UNIVERSIDADE FEDERAL DE SANTA CATARINA

Tradutor Java/LLVM: Geração de código para a máquina virtual LLVM a partir de programas escritos na linguagem de programação JAVA

EDUARDO MELLO CANTÚ

Orientador: Olinto José Varella Furtado

Florianópolis-SC
2008/2

UNIVERSIDADE FEDERAL DE SANTA CATARINA
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO DE SISTEMAS DE INFORMAÇÃO

Tradutor Java/LLVM: Geração de código para a máquina virtual LLVM a partir de programas escritos na linguagem de programação JAVA

EDUARDO MELLO CANTÚ

Orientador: Olinto José Varella Furtado

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Sistemas de Informação.

Florianópolis-SC
2008/2

Autoria: Eduardo Mello Cantú

Título: Tradutor Java/LLVM: Geração de código para a máquina virtual LLVM a partir de programas escritos na linguagem de programação JAVA

Os componentes da banca de avaliação, abaixo listados, consideram este trabalho aprovado.				
	Nome	**Titulação**	**Assinatura**	**Instituição**
1	Olinto José Varella Furtado	Doutor		UFSC
2	José Eduardo De Lucca	Doutor		UFSC
3	Ricardo Azambuja Silveira	Doutor		UFSC

Data da aprovação: ..... **de** ..... **de** .....

Sumário

Lista de Figuras

Resumo

A crescente necessidade de código otimizado em diferentes áreas da computação e em particular na área de sistemas embutidos motivou a criação e o uso da máquina virtual LLVM (Low Level Virtual Machine), para qual, até o momento, só existe front-ends C e C++ e uma versão experimental para Java. Por outro lado, a plataforma Java tem se destacado na preferência de diversos segmentos de desenvolvedores, apesar de suas conhecidas limitações de eficiência. Assim sendo, a idéia deste projeto é encontrar uma maneira de permitir que desenvolvedores e pesquisadores que trabalham com a plataforma Java possam tirar proveito dos benefícios da LLVM.

1 Introdução

A infra-estrutura para qual geraremos o código alvo – LLVM - é uma estratégia de compilação desenvolvida para efetuar otimizações eficientes em aplicativos ao longo de suas vidas. Possui um um conjunto virtual de instruções que permite transformações sofisticadas e códigos escritos em outras linguagens de programação (como C e C++) que são traduzidos para sua representação através de um front-end específico. Estas características vêm dando muita notoriedade para a infra-estrutura da LLVM no mundo acadêmico e conseqüentemente o número de artigos e projetos baseados nela vêm crescendo cada vez mais.

Neste trabalho usamos uma tecnologia muito conhecida, a linguagem Java, que ao longo dos anos vem se consolidando com a linguagem preferida dos desenvolvedores de sistemas tanto pela sua portabilidade quanto facilidade.

Apresentamos uma maneira viável de traduzir o código gerado para a máquina virtual Java. Aproveitando que o mesmo já tenha passado por todas as fases de análises ao ser submetido ao compilador Java, em um código na representação LLVM, tornando assim possível a realização de diversas otimizações.

2 Objetivos

O presente trabalho está incluso no contexto de otimização de código, focado nos desenvolvedores que utilizam a linguagem de programação Java. Para isso se pretende integrar os benefícios da plataforma Java com a eficiência buscada por LLVM.

2.1 Objetivo Geral

O objetivo deste trabalho é estudar detalhadamente a JVM (Java Virtual Machine) e a LLVM para poder montar um esquema de tradução do bytecode gerado por Java para o código interpretado pela LLVM, buscando integrar os benefícios da plataforma Java com a eficiência da LLVM.

2.2 Objetivos Específicos

Para atingir o objetivo proposto é necessário realizar alguns passos intermediários antes de partir para o desenvolvimento do esquema de tradução. Desta forma, este trabalho tem como objetivo:

1. Estudar o framework LLVM juntamente com suas vantagens de utilização;
2. Realizar um estudo aprofundado de Java, dando maior ênfase a sua máquina virtual e a estrutura do arquivo interpretado pelo *class;*
3. Propor uma forma de tradução para código da máquina LLVM a partir de um programa Java compilado em arquivo *class*.

3 Low Level Virtual Machine (LLVM)

Linguagens de programação modernas visam a criação de aplicações mais confiáveis, modulares e dinâmicas , aumentando a produtividade do programador e provendo informações semânticas de alto nível para o compilador. Entretanto, estas características acabam prejudicando a performance da execução de aplicações compiladas.

Situado entre as linguagens de programação modernas e a arquitetura, o compilador é responsável por fazer a aplicação executar da melhor maneira possível. Fazem isto eliminando processamentos desnecessários e fazendo efetivo uso dos recursos do processador. A solução para os problemas é conceitualmente simples: aumentar o escopo da análise e otimização.

Neste capítulo descrevemos a Low-Level Virtual Machine (LLVM), concebida por Lattner em sua tese de mestrado, como sendo uma infra-estrutura de compilador compatível com as arquiteturas e linguagens de programação modernas desenvolvida para alcançar três objetivos:

1. Propor uma estratégia agressiva de otimização de múltiplos estágios;

2. Ser um host (hospedeiro) de pesquisa e desenvolvimento provendo uma fundação robusta para projetos atuais e futuros;

3. Operar de forma transparente para o desenvolvedor comportando-se exatamente como um compilador comum.

A LLVM provê uma excelente performance ao usuário final, um bom tempo de compilação e um ambiente de pesquisa produtivo para desenvolvedores de compiladores.

Um estudo feito com os compiladores existentes que tinha como intuito avaliar os métodos de produção de executáveis de alta performance, constatou que as técnicas apresentam um alto tempo de compilação.

3.1 Arquitetura do sistema LLVM

De acordo com Lattner, a compilação do sistema LLVM é baseada na estratégia de múltiplos passos, sendo esta estratégia de compilação única no sentido de permitir uma otimização agressiva ao longo da vida da aplicação.

3.1.1 O design de alto nível de um compilador baseado na LLVM

Comparado aos atuais sistemas de compilação, o sistema LLVM é desenvolvido para efetuar transformações mais sofisticadas no tempo de ligação (do inglês, link-time), execução e após a instalação do software.

Com o intuito de ser realisticamente aplicável, o compilador LLVM deve se integrar com esquemas de montagem existentes e deve ser suficientemente eficaz para ser utilizável em situações corriqueiras.

Na figura 1, vemos o funcionamento geral do sistema LLVM.

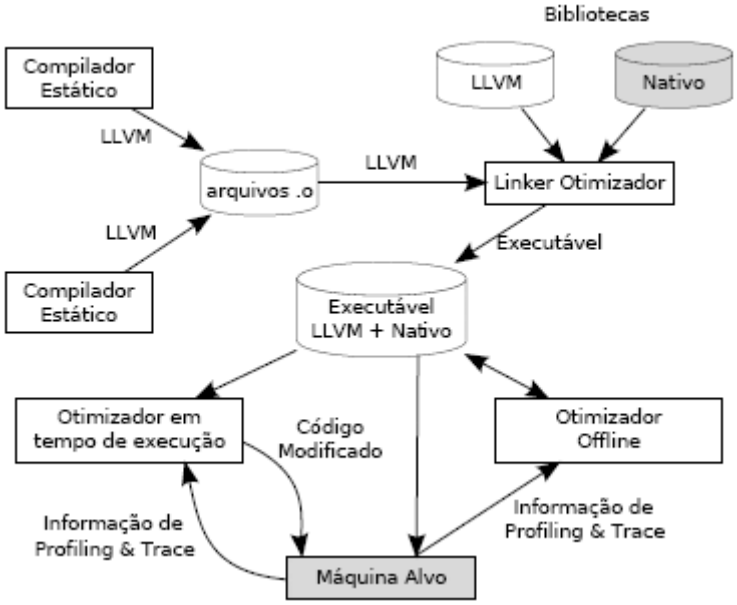

Figura 1: Arquitetura LLVM

Compiladores tradicionais separam o processo de compilação em dois passos: compilação e ligação. Em duas fases teremos os benefícios da compilação separada: apenas as unidades de tradução modificadas devem ser recompiladas. Um compilador tradicional compila o código fonte para um arquivo objeto (.o), contendo código de máquina, e o ligador combina estes arquivos objetos com bibliotecas para formar o programa executável. Em um sistema simples tipicamente o ligador faz pouca coisa além de concatenar os arquivos objetos e resolver as referências simbólicas.

Lattner explica que a estratégia da LLVM diferencia o tempo de compilação e ligação, aproveitando as vantagens da compilação separada. Ao invés de compilar diretamente para código de máquina, o front-end do compilador estático emite código para um conjunto virtual de instruções da LLVM (do inglês, LLVM virtual instruction set). O ligador-otimizador da LLVM combina os arquivos objeto da LLVM, otimiza-os e então os integra com código nativo executável. Esta organização permite que otimizações inter procedurais sofisticadas sejam executadas no tempo de ligação.

O executável produzido pelo ligador-otimizador contém código de máquina nativo executável diretamente na arquitetura da máquina host juntamente com um cópia do código LLVM para a própria aplicação. Quando a aplicação é executada, o otimizador de tempo execução monitora a execução do programa coletando informações características que dizem respeito aos

padrões de uso da aplicação.

Oportunidades de otimização detectáveis do comportamento da aplicação fazem com que o otimizador de execução re-compile e re-otimize dinamicamente partes da aplicação (usando o código LLVM armazenado). Entretanto, algumas transformações são muito custosas para se efetuar diretamente em tempo de execução, pois para estas o tempo de espera é usado pelo otimizador offline (fora da execução) para recompilar a aplicação usando técnicas interprocedurais agressivas e às informações baseadas nos padrões de utilização do usuário final.

A chave do sistema alto nível LLVM é que o conjunto de instruções virtuais LLVM é usado para comunicação entre diferentes ferramentas, que se encaixam em frameworks de desenvolvimento padrão. Operar sobre uma representação comum, permite que as transformações sejam compartilhadas entre os diferentes componentes do sistema.

3.1.2 Tempo de compilação: Front-end e otimizador estático

De acordo com Lattner, o sistema LLVM foi desenvolvido para suportar múltiplos front-ends de linguagens cada qual traduzindo o código fonte da linguagem suportada para o conjunto de instruções virtuais LLVM. Cada compilador estático efetua tanto quanto possível, otimizações nas unidades de tradução, para reduzir a quantidade de trabalho do otimizador de tempo de ligação.

O trabalho inicial do front-end de uma linguagem específica é traduzir o código fonte em questão para o conjunto virtual de instruções da LLVM. Adicionalmente, pode também efetuar otimizações específicas da linguagem.

Como todas as transformações da LLVM são modulares e compartilhadas, os compiladores estáticos podem escolher a utilização de algumas (ou todas) as transformações da estrutura LLVM para melhorar sua capacidade de geração de código.

Com este trabalho, após traduzir o código Java para o conjunto de instruções da LLVM, realizaremos as otimizações através da otimizador e analisador modular da LLVM, acessível através do comando *opt*.

Abaixo, na tabela 1, podemos contemplar uma lista de algumas das possíveis otimizações presentes no comando *opt*.

adce	Eliminação agressiva de código inútil.
codegenprepare	Otimização para geração de código.
condprop	Propagação condicional.
deadargelim	Eliminação de argumentos inúteis.
deadtypeelim	Eliminação de tipos inúteis.
die	Eliminação de instruções inúteis.
globalopt	Otimizador de variáveis globais.
loop-deletion	Exclusão de blocos de repetição inúteis.
memcpyopt	Otimização de *MemCpy*.
mem2reg	Promoção de memória para registrador.

Tabela 1: Tabela de algmas otimizações do comando opt.

Um aspecto importante do conjunto de instruções virtuais da LLVM é a habilidade de suportar código fonte arbitrário, e isto graças a um sistema de tipos de baixo nível. Diferente de máquinas virtuais de alto nível, onde o sistema de tipos da LLVM não especifica um modelo de objeto, um sistema de gerenciamento de memória ou semânticas específicas de exceções. Ao contrário, a LLVM apenas suporta os tipos de mais baixo nível como ponteiros, estruturas e arrays, confiando na linguagem fonte para efetuar o correto mapeamento entre os tipos de mais alto nível.

3.1.3 Tempo de ligação: Ligador e otimizador inter procedural

A ligação é a primeira fase do processo de compilação onde a maior parte do programa fica disponível para análises e transformações e graças a tal aspecto, o ligador é o local mais indicado para que as análises inter procedurais agressivas sejam realizadas.

Finalizado o processo de otimização do ligador, selecionamos um gerador de código apropriado para a plataforma desejada que traduzirá o código da LLVM para o código da máquina em questão.

3.1.4 Tempo de execução: Profiling e reotimização

Um dos objetivos de pesquisa do projeto LLVM é de desenvolver uma nova estratégia de otimização durante o tempo de execução. Esta estratégia consiste em agrupar informações peculiares durante a execução para usá-las na reotimização e recompilação do programa a partir do bytecode LLVM.

3.1.5 Tempo ocioso: Reotimzador offline

Nem todos os aplicativos são passíveis de otimização em tempo de execução. Com o propósito de suportar essas aplicações, temos o reotimizador offline que foi desenvolvido para executar durante o tempo ocioso do computador, permitindo otimizações mais agressivas que aquelas realizadas pelo otimizador de tempo de execução, que combina informações levantadas com o bytecode LLVM para reotimizar e recompilar a aplicação.

3.2 Conjunto de instruções virtuais da LLVM (LLVM Virtual Instruction Set, ou LVIS)

Um dos fatores que diferencia a LLVM de outros sistemas é a representação de programa que ele usa. Este deve ser suficientemente *baixo nível* para permitir a otimização nas fases iniciais da compilação e, suficientemente *alto nível* para suportar as otimizações agressivas feitas nos tempos de ligação e pós-ligação.

O LVIS, como será chamado de agora em diante, foi desenvolvido como sendo uma representação de *baixo nível* com informações de tipos de *alto nível*, este representa uma arquitetura virtual que captura operações de processadores comuns, mas evita restrições de máquina específicas como registradores físicos, pipelines, chamadas de processos de *baixo nível*, etc. A LLVM permite um conjunto infinito de registadores virtuais capazes de armazenar valores de tipos primitivos. Esses registradores virtuais se encontram na forma de Static Single Assignement (SSA), que é muito usual nos algoritmos de análise de fluxo de dados.

A maioria das operações da LLVM estão no formato de três endereços, ou seja, com um ou dois operandos geram um resultado. A representação de três endereços é a representação escolhida pela arquitetura RISC, que pode ser facilmente comprimida, permitindo uma grande densidade nos arquivos LLVM.

Como o LVIS é a *cola* que une o sistema LLVM, previamente descrito neste capítulo como sendo utilizado na comunicação entre diferentes ferramentas, a eficiência e facilidade de uso do sistema depende de muitos aspectos do design do conjunto de instruções. Uma importante característica do LVIS é que ele é uma linguagem de primeira classe, com um formato textual um formato binário comprimido e um formato *em memória* suscetível a transformações.

Abaixo podemos ver a lista das instruções previstas pelo LVIS:

- Instruções terminadoras (indicam qual bloco será executado após o fim do bloco atual): *ret, br, switch, invoke, unwind, unreachable*;
- Instruções de operações binárias (são usadas para fazer a maioria das computações em um programa): *add, sub, mul, udiv, sdiv, fdiv, urem, srem, frem*;
- Instruções de operações binarias *bitwise* (para executar várias formas de movimentações de bits): *shl, lshr, ashr, and, or, xor*;
- Instruções de operações vetoriais (utilizadas para acessos de elementos e outras operações específicas de vetores): *extractelement, insertelement, shufflevector*;
- Instruções de operações agregadas (para trabalhar com valores agregados): *extractvalue,*

insertvalue;

- Instruções de acesso de memória e operações de endereçamento: *malloc, free, alloca, load, store, getelementptr*;

- Instruções de operações de conversão (usadas para casts): *trunc .. to, zext .. to, sext .. to, fptrunc .. to, fpext .. to, fptoui .. to, fptosi .. to, uitofp .. to, sitofp .. to, ptrtoint .. to, inttoptr .. to, bitcast .. to*;

- Instruções para outros tipos de operações: *icmp, fcmp, vicmp, vfcmp, phi, select, call, va\underline{ }arg, getresult*.

4 JAVA

A linguagem de programação Java era chamada de Oak e foi desenvolvida por James Gosling para ser embutida em aparelhos eletrodomésticos. Após alguns anos de experiência com a linguagem e grandes contribuições foi redirecionada para a Internet, renomeada e substancialmente revisada.

Neste capítulo apresentaremos um pouco sobre a história, porém o foco principal será sua máquina virtual (especialmente a estrutura do arquivo *class*), cuja compreensão será de vital importância para este trabalho.

É importante ressaltar que toda a fundamentação teórica buscada para a escrita deste capítulo foi retirada tanto da especificação da máquina virtual Java escrita por Lindholm e Yellim, quanto da especificação da própria linguagem escrita por Gosling, Joy, Steele e Bracha.

4.1 História

A linguagem de programação Java é uma linguagem de propósito geral, orientada a objetos e concorrente. Sua sintaxe é similar a do C e C++, omitindo os aspectos que a torna complexa, confusa e insegura. A plataforma Java foi desenvolvida, inicialmente, para solucionar o problema de construir softwares para dispositivos em rede. Tal propósito deveria ser compatível com diferentes arquiteturas e permitir a entrega segura dos componentes do software. Para garantir estes requisitos, o código compilado deve *sobreviver* ao transporte através de redes, operar em qualquer cliente e garantir execução segura.

A popularização da WEB tornou estes atributos ainda mais interessantes. A internet demonstrou como conteúdos ricos em mídia eram acessíveis de maneira simples e a *navegação* na WEB tornou-se popular. Logo se concluiu que o formato dos documentos HTML para WEB eram muito limitados.

O navegador HotJava da Sun demonstrou propriedades interessantes da linguagem e da plataforma Java, tornando possível acoplar *programas* dentro das páginas HTML. Estes *programas* são *baixados* para o navegador HotJava juntamente com a página HTML na qual aparecem e antes de serem aceitos pelo navegador, os *programas* são cautelosamente verificados para garantir que são seguros. Como as páginas HTML, os *programas* compilados são independentes da rede e do provedor e comportamento dos *programas* são iguais, independentemente de onde vieram ou da máquina para a qual foram carregados.

Um navegador da WEB incorporando a plataforma Java ou Java 2 não é mais limitado a um determinado conjunto de capacidades. Programadores podem escrever o programa uma única vez e executá-lo em qualquer máquina que tenha o ambiente de execução (runtime environment) Java ou

Java 2.

4.2 A máquina virtual Java (Java Virtual Machine, ou JVM)

A máquina virtual Java é o pilar para a plataforma Java e Java 2. É o componente da tecnologia responsável pela independência do hardware e do sistema operacional, do tamanho reduzido do código compilado e da capacidade de proteger os usuários de programas maliciosos.

De acordo com Lindholm e Yellin, a máquina virtual Java é uma máquina computacional abstrata e como uma máquina real, ela possui um conjunto de instruções e manipula diversas área de memória durante a execução. É extremamente comum implementar uma linguagem de programação utilizando uma máquina virtual e, como exemplo, podemos citar a mais conhecida de todos, a máquina P-Code do UCSD Pascal.

O primeiro protótipo da implementação da máquina virtual do Java, à partir de agora chamado de JVM (do inglês, Java Virtual Machine), desenvolvido pela Sun Microsystems, Inc., emulou o conjunto de instruções da JVM em um software *hospedado* por um dispositivo do tipo *handheld* semelhante ao atual PDA (do inglês, Personal Digital Assistant). As atuais implementações da Sun da JVM, componentes dos produtos JavaTM 2 SDK e JavaTM 2 Runtime Environment, emulam a JVM em *provedores* Win32 e Solaris de maneiras muito mais sofisticadas. Entretanto, a JVM não assume nenhuma implementação de tecnologias particulares, hardwares ou sistemas operacionais *provedores*. Não é interpretada nativamente, mas pode ser interpretada através da compilação do conjunto de instruções de uma CPU implementada em microcódigo ou diretamente no silício.

A JVM não entende nada da linguagem de programação Java, só compreende um formato binário particular, o arquivo do formato *class*. Este contém instruções da JVM conhecidos como bytecode e uma tabela de símbolos, assim como informações adicionais.

Em nome da segurança, a JVM impõe um formato forte juntamente com amarras estruturais no código em um arquivo *class*. Entretanto, qualquer linguagem com funcionalidades que possa ser expressada em termos de um arquivo *class* válido pode ser *hospedada* pela JVM.

4.2.1 A estrutura da máquina virtual Java

Antes de adentrar na estrutura da máquina virtual, detalharemos a estrutura do arquivo reconhecido pela máquina – o arquivo *class*.

Compilado para ser executado sobre máquina virtual Java, o arquivo *class* é um formato binário independente de hardware e sistema operacional, normalmente armazenado em um arquivo conhecido como *.class*, que define precisamente a representação de uma classe ou interface.

A máquina virtual define várias áreas de dados durante a execução de um programa, como o registrador PC (program counter), sendo que cada thread executada na máquina virtual tem o seu próprio registrador PC; a pilha da máquina virtual (uma para cada thread); o heap, que é compartilhado entre todas as threads da máquina virtual; a área de métodos, que também é

compartilhada entre todas as threads; a constant pool do runtime (tempo de execução), que é uma representação da constant pool para cada classe ou interface; e a pilha para métodos nativos, que é uma pilha convencional, coloquialmente chamada de *pilha C*, para armazenar métodos nativos.

Frames são utilizados para armazenar dados e resultados parciais, assim como para efetuar a ligação dinâmica, retornar valores para métodos e lançar exceções. É criado cada vez que um método é executado, e alocadas na pilha da máquina virtual onde cada frame tem seu array de variáveis locais, pilha de operandos e uma referência para a constant pool da classe proprietária do método em questão.

4.2.2 O formato do arquivo *class*

O arquivo *class* contém a definição de uma única classe ou interface que é composto por uma seqüência de bytes de 8 bits.

Para facilitar a explanação, utilizamos estruturas para mostrar o relacionamento entre os elementos formadores do arquivo *class*. Nas estruturas apresentadas desta parte em diante, utilizaremos a seguinte convenção para indicar o seu tamanho: u4 para representar 4 bytes, u3 para 3 bytes, u2 para 2 bytes e u1 para 1 byte.

O arquivo *class* é representado pela estrutura ClassFile, conforme descrito abaixo.

```
ClassFile
{
        u4 magic;
        u2 minor_version;
        u2 major_version;
        u2 constant_pool_count;
        cp_info constant_pool[constant_pool_count-1];
        u2 access_flags;
        u2 this_class;
        u2 super_class;
        u2 interfaces_count;
        u2 interfaces[interfaces_count];
        u2 fields_count;
        field_info fields[fields_count];
        u2 methods_count;
        method_info methods[methods_count];
        u2 attributes_count;
        attribute_info attributes[attributes_count];
}
```

A definição de cada um dos atributos que formam a estrutura, podemos assim enumerar:

- magic - é número que identifica um arquivo do tipo *class*, que sempre deve conter o valor 0xCAFEBABE;

- minor_version e major_version - formam a versão do arquivo *class*. Se major_version é um número denotado por M, e minor_version denotado por m, a versão do arquivo *class* será M.m;

- constant_pool_count - indica a quantidade de itens presentes no constant pool desta classe;
- constant_pool[] - é um array, que vai da posição 1 até a posição constant_pool_cont - 1, contendo os itens (cp_info) da constant pool;
- access_flags - armazena o valor que representa as permissões de acesso da classe em questão, onde o valor 0x0001 serve para demarcar a classe como public, 0x0010 como final, 0x0020 como super, 0x0200 para caracterizar uma interface e 0x0400 para afirmar que a classe é abstract;
- this_class - é zero ou uma entrada válida no array constant_pool[];
- super_class - é zero ou uma entrada válida no array constant_pool[] que representa a classe que é *mãe* da classe em questão;
- interfaces_count - é o número de superinterfaces diretas desta classe;
- interfaces[] - contém índices válidos do array contant_pool[] que representam as superinterfaces diretas da classe em questão;
- fields_count - é o número de estruturas do tipo field_info contidos dentro do array fields[];
- fields[] - é um array que contém estruturas do tipo field_info;
- method_count - é o número de estruturas do tipo method_info contidos no array methods[];
- methods[] - é um array que contém estruturas do tipo method_info;
- attributes_count - é o número de estruturas do tipo attribute_info contidos no array attributes[];
- attributes[] - é um array que contém estruturas do tipo attribute_info.

Visto os atributos da estrutura principal, podemos agora detalhar cada uma das estruturas que formam esses, que são cp_info, field_info, method_info e attribute_info.

4.2.2.1 Estrutura cp_info

A constant pool é uma tabela composta por um ou mais itens, onde cada item é representado pela estrutura cp_info e tem como propósito armazenar valores referenciados no código.

Abaixo temos a representação desta estrutura:

```
cp_info
{
        u1 tag;
```

```
        u1 info[];
}
```

A definição de cada um dos atributos é:

tag - é utilizado para definir qual o tipo da informação que está contida na constant pool;
info[] - é o conjunto de informações da entrada sendo que cada entrada tem sua própria descrição de atributos. Os tipos de entradas da constant pool variam de acordo com o valor do atributo tag da estrutura cp_info, visto que os mesmos respeitam a seguinte regra (valor da tag e o respectivo tipo de entrada):

- **CONSTANT_Class_info**

```
CONSTANT_Class_info
{
        u1 tag;
        u2 name_index;
}
```

Onde:

- tag - conforme descrito na estrutura cp_info contém o tipo de entrada, neste caso 7, que referencia uma entrada do tipo CONSTANT_Class;
- name_index - é um índice válido da tabela constant_pool[] que, conforme descrito anteriormente é um item cp_info do tipo CONSTANT_Utf8_info que será descrito representando o nome completo de uma ou interface.

- **CONSTANT_Fieldref_info, CONSTANT_Methodref_info e CONSTANT_InterfaceMethodref_info**

```
CONSTANT_X_info
{
        u1 tag;
        u2 class_index;
        u2 name_and_type_index;
}
```

Onde:

- tag - em CONSTANT_Fieldref_info tem o valor 9, CONSTANT_Methodref tem o valor 10 e CONSTANT_InterfaceMethodref o valor 11;
- class_index - é um índice válido dentro do tabela constant_pool[] que nesta posição possui uma estrutura do tipo CONSTANT_Class_info, anteriormente detalhada, e que contém a declaração do campo ou método;

- name_and_type_index - índice da constant pool que possui um CONSTANT_NameAndType_info. Se a estrutura for CONSTANT_Fieldref_info representa um nome e descritor de campo. Se for CONSTANT_Methodref_info ou um CONSTANT_InterfaceMethodref_info representa o nome e o descritor de um método.

- **CONSTANT_String_info**

```
CONSTANT_String_info
{
        u1 tag;
        u2 string_index;
}
```

Onde:

- tag - o valor para este tipo específico é 8;

- string_index - índice válido em constant_pool[] associada a CONSTANT_Utf8_info, representando uma seqüência de caracteres que inicializa o objeto String.

- **CONSTANT_Integer_info e CONSTANT_Float_info**

```
CONSTANT_X_info
{
                u1 tag;
                u4 bytes;
}
```

Novamente, utilizamos a estrutura CONSTANT_X_info, a qual não existe na prática, para representar estruturas semelhantes, onde:

- tag - define o tipo da entrada cp_info, com o valor 3 para CONSTANT_Integer_info e 4 para CONSTANT_Float_info;

- bytes - no caso do tipo CONSTANT_Integer_info, armazena o valor *int* e, no caso do tipo CONSTANT_Float_info, armazenam o valor float no formato simples do ponto-flutuante IEEE 754.

- **CONSTANT_Long_info e CONSTANT_Double_info**

```
CONSTANT_X_info
{
        u1 tag;
        u4 high_bytes;
        u4 low_bytes;
}
```

Onde:

- tag - valor 5 para CONSTANT_Long_info e 6 para CONSTANT_Double_info;
- high_bytes e low_bytes - em CONSTANT_Long_info armazena o valor long respeitando a forma ((long) high_bytes << 32) + low_bytes e em CONSTANT_Double_info armazena o valor double respeitando o formato duplo do ponto-flutuante IEEE 754.

- **CONSTANT_NameAndType_info**

```
CONSTANT_NameAndType_info
{
        u1 tag;
        u2 name_index;
        u2 descriptor_index;
}
```

Onde:

- tag - neste caso, com o valor 12;
- name_index - é um índice válido da tabela constant_pool[] associado a uma estrutura do tipo CONSTANT_Utf8_info que representa o nome de um campo ou método válido, sendo um identificador válido da linguagem Java ou o nome especial de método <init>;
- descriptor_index - é também um índice válido da tabela constant_pool[] associado a estrutura CONSTANT_Utf8_info, só que este representa o descritor de uma campo ou método válido.

- **CONSTANT_Utf8_info**

```
CONSTANT_Utf8_info
{
        u1 tag;
        u2 length;
        u1 bytes[length];
}
```

Onde:

- tag - que neste caso está com o valor 1;
- length - contém o número de bytes dentro do array de bytes;

- bytes[] - é um array de bytes que guarda uma informação no formato de string.

4.2.2.2 Estrutura field_info

Cada campo vindo da linguagem fonte Java é descrito por uma estrutura do tipo field_info, sendo que cada item representado por essa esta estrutura é armazenado na tabela fileds[] e que em uma classe não se pode ter dois campos com o mesmo nome e descritor.

Abaixo temos a representação da estrutura:

```
field_info
{
        u2 access_flags;
        u2 name_index;
        u2 descriptor_index;
        u2 attributes_count;
        attribute_info attributes[attributes_count];
}
```

A definição de cada um dos atributos pode ser assim elencada:

- access_flags - representa as permissões de acesso do campo e pode assumir os seguintes valores: 0x0001 para public, 0x0002 para private, 0x0004 para protected, 0x0008 para static, 0x0010 para final, 0x0040 para *volatile* e 0x0080 para *transient*;
- name_index - é uma índice válido dentro da tabela constant_pool[] associado a uma estrutura CONSTANT_Utf8_info que representa o nome do campo;
- descriptor_index - é um índice válido dentro da tabela constant_pool[] associado a uma estrutura CONSTANT_Utf8_info que representa a descrição do campo;
- attributes_count - é o número de atributos contidos no array attributes[];
- attributes[] - contém um lista de estruturas do tipo attribute_info que serão descrita mais adiante.

4.2.2.3 Estrutura method_info

Cada método derivado de uma classe Java é descrito pela estrutura method_info e estes são armazenados na tabela methods[], observado que uma classe não pode ter mais de um método com o mesmo nome e descritor.

A representação desta estrutura é a seguinte:

```
method_info
```

```
{
        u2 access_flags;
        u2 name_index;
        u2 descriptor_index;
        u2 attributes_count;
        attribute_info attributes[attributes_count];
}
```

Onde:

- access_flags - define as permissões de acesso do método onde os valores assumidos foram: 0x0001 (public), 0x0002 (private), 0x0004 (protected), 0x0008 (static), 0x0010 (final), 0x0020 (synchronized), 0x0100 (native), 0x0400 (abstract) e 0x0800 (strict);

- name_index - é um índice válido dentro da tabela constant_pool[] da classe em questão, que representa ou os métodos especiais <init> e <clinit> ou um método válido da linguagem Java. Este índice é associado a uma estrutura do tipo CONSTANT_Utf8_info;

- descriptor_index - é um índice válido da tabela constant_pool[] (também associado a uma estrutura CONSTANT_Utf8_info) que representa a descrição do método;

- attributes_count - indica o número de atributos deste método armazenados no array attributes[];

- attributes[] - é um array de atributos no formato de estruturas do tipo atttribute_info que será descrita ao longo deste trabalho.

4.2.2.4 Estrutura attribute_info

Atributos são utilizados nas estruturas ClassFile, field_info, method_info e Code_attribute.

A representação da estrutura segue o modelo abaixo.

```
attribute_info
{
        u2 attribute_name_index;
        u4 attribute_length;
        u1 info[attribute_length];
}
```

Várias estruturas são na verdade especializações da estrutura básica attribute_info, funcionando mais ou menos com uma herança entre classe. Para todas estas estruturas especializadas, os atributos attribute_name_index e attribute_length têm basicamenete a mesma funcionalidade, variando apenas no propósito da informação neles contida.

Defineremos outras estruturas, que como comentando herdam as propriedades da estrutura attribute_info, juntamente com a descrição detalhada dos atributos que as compõem.

- **ConstantValue_attribute**

```
ConstantValue_attribute
{
        u2 attribute_name_index;
        u4 attribute_length;
        u2 constantvalue_index;
}
```

Onde:

- attribute_name_index - índice válido associado a CONSTANT_Utf8_info no constant pool representando a string *ConstantValue*;
- attribute_length - deve ser 2, neste caso;
- constantvalue_index - índice válido em constant pool representando um dos tipos long (CONSTANT_Long), float (CONSTANT_Float), double (CONSTANT_Double), int/short/char/byte/boolean (CONSTANT_Integer) ou String (CONSTANT_String).

- **Code_attribute**

```
Code_attribute
{
        u2 attribute_name_index;
        u4 attribute_length;
        u2 max_stack;
        u2 max_locals;
        u4 code_length;
        u1 code[code_length];
        u2 exception_table_length;
        {
                u2 start_pc;
                u2 end_pc;
                u2  handler_pc;
                u2  catch_type;
        }
        exception_table[exception_table_length];
        u2 attributes_count;
        attribute_info attributes[attributes_count];
}
```

Onde:

- attribute_name_index - índice válido associado a um CONSTANT_Utf8_info da tabela constant_pool[] represetando a string *Code*;
- attribute_length - indica o tamanho do atributo;

- max_stack - indica o tamanho máximo da pilha de operações deste método durante sua execução;
- max_locals - indica o número máximo de variáveis alocadas no array de variáveis locais;
- code_length - indica o tamanho do array de bytes (code[]);
- code[] - é o array de bytes contedo os códigos de operações reais da máquina virtual;
- exception_table_length - entradas da tabela de exceções (exception_table[]);
- exception_table[] - é uma tabela contendo as entradas do tipo exception_table. Cada entrada representa um tratador de exceções do código que é formada pelos atributos start_pc e end_pc e marcam dentro do array de código (code[]) as faixas onde a exceção é ativada, o handler_pc indica o início do tratamento da exceção, e o catch_type, quando houver, deve ser um índice válido no array constant_pool[] para representar uma classe de exceção;
- attributes_count - indica o número de atributos contidos na lista de attributos internos;
- attributes[] - é o array contendo estruturas do tipo attribute_info.

- **Exceptions_attribute**

```
Exceptions_attribute
{
        u2 attribute_name_index;
        u4 attribute_length;
        u2 number_of_exceptions;
        u2 exception_index_table[number_of_exceptions];
}
```

Onde:

- attribute_name_index - item associado a estrutura CONSTANT_Utf8_info na tabela constant_pool[] que contém a string *Exceptions*;
- attribute_length - indica o tamanho do atributo;
- number_of_exceptions - indica a quantidade de itens do array exception_index_table[];
- exception_index_table[] - contém diversos itens do tipo exception_index_table, sendo que cada um destes é um índice válido dentro do constant_pool[] representado por uma classe que é declarada para capturar as exceções lançadas por este método;

- **InnerClasses_attribute**

```
InnerClasses_attribute
{
        u2 attribute_name_index;
        u4 attribute_length;
        u2 number_of_classes;
        {
                u2 inner_class_info_index;
                u2 outer_class_info_index;
                u2 inner_name_index;
                u2 inner_class_access_flags;
        }
        classes[number_of_classes];
}
```

Onde:

- attribute_name_index - índice de constant pool associado a CONSTANT_Utf8_info contendo a string *InnerClasses*;
- attribute_length - informa o tamanho do atributo;
- number_of_classes - indica o número de entradas em classes[];
- classes[] - classes internas à classe proprietária. Os itens do tipo classes possuem os atributos inner_class_info_index, se existir, deve ser um item válido em constant pool representando uma classe; outer_class_info_index, se existir, deve ser um item válido em constant pool representando uma classe; inner_name_index, caso exista, deve ser um item válido em constant pool representando o nome original da classe; inner_class_access_flags, que marcam as permissões das classes, sendo os possíveis valores 0x0001 (public), 0x0002 (private), 0x0004 (protected), 0x0008 (static), 0x0010 (final), 0x0200 (interface) e 0x0400 (abstract).

- **SourceFile_attribute**

```
SourceFile_attribute
{
        u2 attribute_name_index;
        u4 attribute_length;
        u2 sourcefile_index;
}
```

Onde:

- attribute_name_index - index é um índice em constant_pool[] associado a uma estrutura CONSTANT_Utf8_info, a qual contém a string *SourceFile*;
- attribute_length - indica o tamanho do atributo, neste caso 2 (dois);

- sourcefile_index - índice válido ligado a um CONSTANT_Utf8_info em constant pool representando uma string.

- **LineNumberTable_attribute**

```
LineNumberTable_attribute
{
        u2 attribute_name_index;
        u4 attribute_length;
        u2 line_number_table_length;
        {
                u2 start_pc;
                u2 line_number;
        }
        line_number_table[line_number_table_length];
}
```

Onde:

- attribute_name_index - é um item válido da tabela constant_pool[] que contém uma estrutura do tipo CONSTANT_Utf8_info com a string *LineNumberTable*;

- attribute_length - indica o tamanho do atributo;

- line_number_table_length - indica o tamanho do array line_number_table[];

- line_number_table[] - contém itens do tipo line_number_table com o número da linha no código original indicando uma posição do array code[]. São formados pelos atributos start_pc;

- line_number - contém o número da linha do código original.

- **LocalVariableTable_attribute**

```
LocalVariableTable_attribute
{
        u2 attribute_name_index;
        u4 attribute_length;
        u2 local_variable_table_length;
        {
                u2 start_pc;
                u2 length;
                u2 name_index;
                u2 descriptor_index;
                u2 index;
        }
        local_variable_table[local_variable_table_length];
}
```

Onde:

- attribute_name_index - é um item válido da tabela constant_pool[] associado a uma estrutura do tipo CONSTANT_Utf8_info, que contém a string *LocalVariableTable*;
- attribute_length indica - o tamanho do atributo;
- local_variable_table_length - indica o número de itens dentro de local_variable_table[];
- local_variable_table[] - contém os itens do tipo local_variable_table que representam uma faixa dentro do array code[] que possui o valor de uma variável local. Cada item é composto pelos atributos start_pc que juntamente com length indica a faixa dentro do array code[] onde se encontra a variável; name_index indica um índice válido dentro de constant_pool[] que representa o nome de uma variável local; descriptor_index indica o índice da constant_pool[] que contém a representação da descrição do campo; e index, que indica a posição dentro do array de variáveis locais do frame atual que se encontra variável.

- **Deprecated_attribute**

```
Deprecated_attribute
{
        u2 attribute_name_index;
        u4 attribute_length;
}
```

Onde:

- attribute_name_index - é um item válido da tabela constant_pool[] que contém a string *Deprecated* e está presente em uma estrutura do tipo CONSTANT_Utf8_info;
- attribute_length - neste caso tem valor 0 (zero).

Detalhada a estrutura do arquivo *class*, que é considerado o pilar da máquina virtual, podemos assim compreender como são feitos os processos de carga, ligação e inicialização.

4.2.3 O processo de carga, ligação e inicialização

A máquina virtual Java automaticamente carrega, liga e inicializa classes e interfaces. Carregar é o processo de encontrar uma representação binária de uma classe ou interface com um nome particular e criá-la com esta representação. Ligação é o processo de combinar uma classe ou interface com o estado de execução de uma máquina virtual Java para que assim seja executada. Inicialização de uma classe ou interface consiste em executar seu método de inicialização <clinit>.

A máquina virtual começa o processo criando uma classe inicial. Em seguida ela liga a

classe inicial, inicializa-a e invoca o método público void main(String[]). A invocação deste método leva a toda execução subseqüente.

A criação de uma classe ou interface *C* denotada pelo nome *N* consiste na construção da área de métodos da máquina virtual de uma implementação específica da representação interna de *C*. A criação de uma classe ou interface é ativada por outra classe ou interface *D*, que referencia *C* através de sua constant pool de execução.

Se *C* não é uma classe do tipo array é criada através da carga da representação binária de *C* usando um class loader (carregador de classes), que pode ser o Bootstrap class loader ou um class loader definido pelo usuário, devendo este ser uma subclasse de ClassLoader.

Carregar uma classe através do Bootstrap class loader consiste nos seguintes passos:

1. A máquina virtual procura por uma representação de *C* de maneira independente da plataforma. Nesta fase, deve-se detectar o erro *NoClassDefFoundError* caso nenhuma representação de *C* seja encontrada. Na seqüência a máquina virtual tenta derivar uma classe denotada por *N* usando o Bootstrap class loader, através da representação encontrada usando o algoritmo específico.

2. O Bootstrap class loader pode delegar a carga de *C* para algum class loader definido pelo usuário como *L*, por exemplo, passando *N* para uma invocação do método loadClass em *L*. O resultado é a invocação de *C*, pois a máquina virtual armazena a informação de que o Bootstrap class loader é o carregador inicial de *C*.

Para carregar uma classe ou interface através de um class loader definido pelo usuário temos os seguintes passos:

1. O class loader *L* pode criar um array de bytes representando *C* como uma estrutura ClassFile; onde deve ser invocado o método defineClass da classe ClassLoader, que ao invocá-lo faz com que a máquina virtual derive uma classe ou interface denotada por *N* usando *L* do array de bytes através do algorítimo específico.

2. O class loader *L* pode delegar a carga de *C* para outro class loader *L`*. Isto é feito passando *N* como argumento diretamente ou indiretamente, para um método de invocação de *L`*, tipicamente loadClass, onde o resultado é a invocação de *C*.

O próximo passo para a máquina virtual é fazer a ligação (linking). A ligação (linking) de uma classe ou interface envolve fazer a verificação e preparação da classe ou interface, de suas superclasses diretas, super interfaces diretas e seus elementos caso seja necessário. A resolução das referências simbólicas da classe ou interface é uma parte opcional da ligação.

A representação de uma classe ou interface é verificada para garantir que a representação binária seja estruturalmente válida, e pode fazer com que classes e interfaces adicionais sejam carregadas.

Uma classe ou interface deve ser verificada com sucesso antes de ser inicializada, em seguida, faz-se a preparação, que envolve a criação dos campos estáticos (statics) da classe ou interface e a inicialização desses campos com valores padrão e não deve ser confundida com a execução de inicializadores estáticos, pois diferentemente destes, essa não requer a execução de nenhum código da máquina da virtual.

O processo de determinação dinâmica de valores concretos para referências simbólicas na constant pool de execução é chamado de resolução.

Algumas instruções da máquina virtual (anewarray, checkcast, getfield, getstatic,instanceof, invokeinterface, invokespecial, invokestatic, invokevirtual, multianewarray, new, putfield e putstatic) fazem referências simbólicas para a constant pool de execução, e a execução destas instruções requer a resolução destas referências.

A inicialização de um classe ou interface consiste em invocar seus inicializadores estáticos e dos campos estáticos declarados na classe.

Uma classe ou interface só deve ser inicializada se uma das instruções (new, getstatic, putstatic e invokestatic) da máquina virtual referenciar a classe ou interface, ou se algum método reflectivo for invocado, ou se uma de suas subclasses for inicializada, ou ainda se é designada como a classe inicial pela inicialização da máquina virtual.

4.2.4 A lista de instruções da Máquina Virtual Java

A lista de instruções da máquina virtual Java consiste em um opcode (código de operação) especificando alguma operação a ser efetuada, seguida de zero ou mais operandos encorporando valores a serem operados.

Na tabela 2 podemos ver a lista dos códigos de operações e seus respectivos significados.

00 (0x00) nop	42 (0x2a) aload_0	84 (0x54) bastore	126 (0x7e) iand	168 (0xa8) jsr
01 (0x01) aconst_null	43 (0x2b) aload_1	85 (0x55) castore	127 (0x7f) land	169 (0xa9) ret
02 (0x02) iconst_m1	44 (0x2c) aload_2	86 (0x56) sastore	128 (0x80) ior	170 (0xaa) tableswitch
03 (0x03) iconst_0	45 (0x2d) aload_3	87 (0x57) pop	129 (0x81) lor	171 (0xab) lookupswitch
04 (0x04) iconst_1	46 (0x2e) iaload	88 (0x58) pop2	130 (0x82) ixor	172 (0xac) ireturn
05 (0x05) iconst_2	47 (0x2f) laload	089 (0x59) dup	131 (0x83) lxor	173 (0xad) lreturn
06 (0x06) iconst_3	48 (0x30) faload	090 (0x5a) dup_x1	132 (0x84) iinc	174 (0xae) freturn
07 (0x07) iconst_4	49 (0x31) daload	091 (0x5b) dup_x2	133 (0x85) i2l	175 (0xaf) dreturn
08 (0x08) iconst_5	50 (0x32) aaload	092 (0x5c) dup2	134 (0x86) i2f	176 (0xb0) areturn
09 (0x09) lconst_0	51 (0x33) baload	093 (0x5d) dup2_x1	135 (0x87) i2d	177 (0xb1) return
10 (0x0a) lconst_1	52 (0x34) caload	094 (0x5e) dup2_x2	136 (0x88) l2i	178 (0xb2) getstatic
11 (0x0b) fconst_0	53 (0x35) saload	095 (0x5f) swap	137 (0x89) l2f	179 (0xb3) putstatic
12 (0x0c) fconst_1	54 (0x36) istore	096 (0x60) iadd	138 (0x8a) l2d	180 (0xb4) getfield
13 (0x0d) fconst_2	55 (0x37) lstore	097 (0x61) ladd	139 (0x8b) f2i	181 (0xb5) putfield
14 (0x0e) dconst_0	56 (0x38) fstore	098 (0x62) fadd	140 (0x8c) f2l	182 (0xb6) invokevirtual
15 (0x0f) dconst_1	57 (0x39) dstore	099 (0x63) dadd	141 (0x8d) f2d	183 (0xb7) invokespecial
16 (0x10) bipush	58 (0x3a) astore	100 (0x64) isub	142 (0x8e) d2i	184 (0xb8) invokestatic
17 (0x11) sipush	59 (0x3b) istore_0	101 (0x65) lsub	143 (0x8f) d2l	185 (0xb9) invokeinterface
18 (0x12) ldc	60 (0x3c) istore_1	102 (0x66) fsub	144 (0x90) d2f	186 (0xba) xxxunusedxxx1
19 (0x13) ldc_w	61 (0x3d) istore_2	103 (0x67) dsub	145 (0x91) i2b	187 (0xbb) new
20 (0x14) ldc2_w	62 (0x3e) istore_3	104 (0x68) imul	146 (0x92) i2c	188 (0xbc) newarray
21 (0x15) iload	63 (0x3f) lstore_0	105 (0x69) lmul	147 (0x93) i2s	189 (0xbd) anewarray
22 (0x16) lload	64 (0x40) lstore_1	106 (0x6a) fmul	148 (0x94) lcmp	190 (0xbe) arraylength
23 (0x17) fload	65 (0x41) lstore_2	107 (0x6b) dmul	149 (0x95) fcmpl	191 (0xbf) athrow
24 (0x18) dload	66 (0x42) lstore_3	108 (0x6c) idiv	150 (0x96) fcmpg	192 (0xc0) checkcast
25 (0x19) aload	67 (0x43) fstore_0	109 (0x6d) ldiv	151 (0x97) dcmpl	193 (0xc1) instanceof
26 (0x1a) iload_0	68 (0x44) fstore_1	110 (0x6e) fdiv	152 (0x98) dcmpg	194 (0xc2) monitorenter
27 (0x1b) iload_1	69 (0x45) fstore_2	111 (0x6f) ddiv	153 (0x99) ifeq	195 (0xc3) monitorexit
28 (0x1c) iload_2	70 (0x46) fstore_3	112 (0x70) irem	154 (0x9a) ifne	196 (0xc4) wide
29 (0x1d) iload_3	71 (0x47) dstore_0	113 (0x71) lrem	155 (0x9b) iflt	197 (0xc5) multianewarray
30 (0x1e) lload_0	72 (0x48) dstore_1	114 (0x72) frem	156 (0x9c) ifge	198 (0xc6) ifnull
31 (0x1f) lload_1	73 (0x49) dstore_2	115 (0x73) drem	157 (0x9d) ifgt	199 (0xc7) ifnonnull
32 (0x20) lload_2	74 (0x4a) dstore_3	116 (0x74).....ineg	158 (0x9e) ifle	200 (0xc8) goto_w
33 (0x21) lload_3	75 (0x4b) astore_0	117 (0x75) lneg	159 (0x9f) if_icmpeq	201 (0xc9) jsr_w
34 (0x22) fload_0	76 (0x4c) astore_1	118 (0x76) fneg	160 (0xa0) if_icmpne	Reserved opcodes:
35 (0x23) fload_1	77 (0x4d) astore_2	119 (0x77) dneg	161 (0xa1) if_icmplt	202 (0xca) breakpoint
36 (0x24) fload_2	78 (0x4e) astore_3	120 (0x78) ishl	162 (0xa2) if_icmpge	254 (0xfe) impdep1
37 (0x25) fload_3	79 (0x4f) iastore	121 (0x79) lshl	163 (0xa3) if_icmpgt	255 (0xff) impdep2
38 (0x26) dload_0	80 (0x50) lastore	122 (0x7a) ishr	164 (0xa4) if_icmple	
39 (0x27) dload_1	81 (0x51) fastore	123 (0x7b) lshr	165 (0xa5) if_acmpeq	
40 (0x28) dload_2	82 (0x52) dastore	124 (0x7c) iushr	166 (0xa6) if_acmpne	
41 (0x29) dload_3	83 (0x53) aastore	125 (0x7d) lushr	167 (0xa7) goto	

Tabela 2: Tabela de instruções da JVM

5 *O processo de tradução*

Este capítulo compreenderá a teoria por trás do processo da tradução entre linguagens e na obtenção do conhecimento necessário para realizar a estruturação de uma estratégia viável de desenvolvimento.

O capítulo foi particionado da seguinte forma:

- A Primeira parte é dedicada ao entendimento do processo de tradução através de outros trabalhos semelhantes.
- Na segunda parte pretende-se explanar o esforço feito para que fosse possível a realização da leitura do bytecode Java (representado pelo arquivo *class*), tarefa de extrema importância para o processo de tradução, pois ele será o código fonte.
- Na terceira e última etapa, propomos a implementação do tradutor responsável pela transformação do código fonte (bytecode Java) em código objeto (código LLVM).

5.1 Tradução

A tradução é definida como uma atividade que abrange a interpretação do significado de um texto em uma língua (o texto fonte) e a produção de um novo texto em outra língua mas que exprima o texto original da forma mais exata possível na língua destino, o texto resultante também se chama tradução.

Na área da computação, um tradutor é um programa que converte um programa ou algo específico (textos principalmente) em outra linguagem ou em atividades.

Este trabalho insere-se no contexto da tradução entre duas linguagens computacionais.

Encontrou-se muitos trabalhos sobre o processo de tradução da linguagem Java para código nativo da máquina alvo através de compiladores C, entre eles, podemos citar o TOBA, que é baseado no artigo homônimo, um sistema para gerar aplicativos Java standalone eficientes.

De acordo com Proebsting, Townsend, Bridges, Newsham e Watterson, juntamente com o TOBA temos um compilador bytecode Java para C, um coletor de lixo, um pacote de threads e o suporte para a API Java. A promessa do TOBA é que os aplicativos gerados por ele rodem entre 1.5 e 10 vezes mais rápido que aplicativos interpretados e compilados pelo compilador Just-in-time. Para obter este feito, o TOBA traduz o arquivo *class* em código C para então compilá-lo em código de máquina. Os objetos resultantes são então ligados através do sistema de execução do TOBA para

criar os tradicionais arquivos executáveis.

O processo de tradução do TOBA consiste em traduzir cada método encontrado no arquivo *class* em uma função C correspondente, conforme a seqüência de passos a seguir:

1. Lê as instruções do arquivo *class*;
2. Computa o estado da pilha para cada intrução;
3. Verifica instruções que são pontos de entrada de exceções e define labels para elas;
4. Verifica intruções alvos de jumps (saltos condicionais) e define labels para elas;
5. Gera cabeçalho de função C e
6. Gera código C para cada instrução.

Além do TOBA, foram encontrados trabalhos referentes a tradução do bytecode Java diretamente para código assembly (código de montagem) para X86. Em um dos trabalhos, *Translating Java bytecode to X86 assembly code*, os pesquisadores usaram a estratégia de deixar a criação e manipulação dos objetos para a máquina virtual Java. O problema encontrado foi que o código assembly continuava fazendo chamadas freqüêntes para funções da máquina virtual.

Segundo Chen, Chen e Yang, uma alternativa para o problema seria realizar a criação e a manipulação dos objetos através de código assembly, mas isso acabaria com a facilidade do uso das funções da máquina virtual. A saída encontrada por eles foi um tradutor off-line que permitisse otimizações precisas. A máquina virtual Java é uma máquina de pilha e os pesquisadores utilizaram a pilha nativa da arquitetura X86 para emulá-la. Os objetos são alocados na heap da máquina virtual e as estruturas de dados auxiliares criadas pelo tradutor, juntamente com constantes do constant pool do arquivo class, são alocados em um arquivo '.data'.

O processo de tradução consiste basicamente em extrair as informações do arquivo *class* e construir tabelas para calcular endereços e gerar instruções assembly.

Chen, Chen e Yang concluíram que o código gerado, mesmo obtendo performance superior ao código interpretado, obteve performance inferior ao código submetido ao compilador Just-in-time.

5.2 Mapeamento do arquivo *class*

Aqui descreve-se o esforço feito para se obter as informações relevantes provenientes do arquivo *class* que é o código fonte do processo de tradução proposto por este trabalho.

A primeira tentativa foi para desenvolver uma ferramenta experimental, visto que a estrutura formadora do arquivo *class* já foi estudada anteriormente.

O propósito principal da criação foi meramente educacional, pois não se esperava que este atendesse a todas as etapas previstas pela máquina virtual, que são os processos de criação, carga e

ligação.

Com este pensamento buscou-se um biblioteca completa com o propósito não apenas de montar as estruturas da arquivo *class*, mas também realizar todas as etapas necessárias. Assim, optou-se pelo *Byte Code Engineering Library.*

De acordo com estes testes, obtivemos os seguintes resultados:

5.2.1 Ferramenta experimental

Observou-se era possível desenvolver uma ferramenta própria para efetuar a leitura do arquivo que contém o bytecode Java. Portanto, neste momento, descrevemos a experiência realizada através da ferramenta experimental.

Partiu-se da premissa que a estrutura principal do arquivo *class* seria o ClassFile e dela partiríamos para todas as outras estruturas, conforme foi analisado no capítulo referente ao bytecode do Java.

Para desenvolver esta ferramenta, se optou pela linguagem Java pois, não bastando a grande afinidade entre o pesquisador e a linguagem, o arquivo *class* é composto por uma seqüência de bytes de 8 bits que facilmente são lidos através dos métodos readUnsignedByte, readUnsignedShort e readInt da classe java.io.DataInputStream.

O primeiro passo é criar uma classe do tipo ClassFile com auxílio do método estático parse da classe Bytecode2Jvm que recebe como parâmetro um string com o caminho completo do arquivo *class*. O método parse, internamente, criará um novo objeto do tipo ClassFile e invocará o método parseYourSelf implementado da interface InterfaceParsableElemen, passando como parâmetro o objeto java.io.DataInputStream, criado para ler a seqüência de bytes do arquivo *class*.

Para demonstrar o processo detalhado nada melhor do que um trecho de código explicativo. Para tanto, apresentamos um exemplo de uso do método responsável pela leitura do arquivo *class*.

```
ClassFile mainClass = Bytecode2Jvm.parse( "C:\\Simples.class" );
System.out.println(mainClass );
```

O arquivo fonte da classe que será lida segue a seqüência:

```
public class Simples
{
        public static int somar(int x, int y) {
                return x+y;
        }
        public static void main( String[] args ) {
                int x = somar(1,2);
        }
}
```

O código do exemplo citado anteriormente faz exatamente o que foi descrito, ou seja, carrega as estruturas com base no bytecode contido no arquivo *class*.

A seguir podemos ver a saída.

```
magic_number: cafebabe
minor_version: 3
major_version: 45
access_flags: 100001
this_class: 1
super_class: 3
interfaces_count: 0
interfaces[]: [S@186d4c1
fields_count: 0

field info: (INICIO)
field info: (FIM)
methods_count: 3

method info: (INICIO)
index: 19; access_flags: 1001
name_index: 19
descriptor_index: 20
attributes_count: 1

attributes[]: (INICIO)
index: 7; max_stack:2
max_locals:2
code_length:7
code:
    iconst_1
    iconst_2
    invokestatic
    nop
    iload
    istore_1
    return

exception_table_length:0
exception_table[]:
attributes_count:2
index:10; line_number_table_length:2
line_number_table[]:
start_pc:0
line_number:7

start_pc:6
line_number:8

index:11; local_variable_table_length:2
local_variable_table[]:
start_pc:0
length:7
name_index:23
descriptor_index:24
index:0

start_pc:6
```

```
length:1
name_index:16
descriptor_index:17
index:1

attributes[]: (FIM)

index: 5; access_flags: public
name_index: 5
descriptor_index: 6
attributes_count: 1

attributes[]: (INICIO)
index: 7; max_stack:1
max_locals:1
code_length:5
code:
    aload_0
    invokespecial
    nop
    iconst_5
    return

exception_table_length:0
exception_table[]:
attributes_count:2
index:10; line_number_table_length:1
line_number_table[]:
start_pc:0
line_number:1

index:11; local_variable_table_length:1
local_variable_table[]:
start_pc:0
length:5
name_index:12
descriptor_index:13
index:0

attributes[]: (FIM)

index: 14; access_flags: 1001
name_index: 14
descriptor_index: 15
attributes_count: 1

attributes[]: (INICIO)
index: 7; max_stack:2
max_locals:2
code_length:4
code:
    iload_0
    iload_1
    iadd
    ireturn

exception_table_length:0
exception_table[]:
attributes_count:2
index:10; line_number_table_length:1
```

```
line_number_table[]:
start_pc:0
line_number:4

index:11; local_variable_table_length:2
local_variable_table[]:
start_pc:0
length:4
name_index:16
descriptor_index:17
index:0

start_pc:0
length:4
name_index:18
descriptor_index:17
index:1

attributes[]: (FIM)

method info: (FIM)
attributes_count: 1

attributes[]: (INICIO)
index: 25; source_index:26

attributes[]: (FIM)
```

5.2.2 O Byte Code Engineering Library (BCEL)

A API (Application programming interface) do BCEL, conforme especificado no manual de referência disponibilizado na página WEB, abstrai a necessidade de conhecer profundamente a máquina virtual Java e como ler e escrever em arquivos binários (*class*). Constitui-se de três partes:

1. Um pacote que possui classes que descrevem o formato e as estruturas do arquivo *class*. Seu propósito é ler arquivos *class* de um arquivo, ou escrever;

2. Um pacote para dinamicamente gerar ou modificar objetos JavaClass ou Method. Serve para análises de código, remoção de informações necessárias do arquivo *class* ou para implementar um gerador de código;

3. Vários exemplos e códigos.

Para entender melhor o funcionamento da API fez-se um exemplo usando como arquivo *class* o mesmo do exemplo anterior, ou seja, o arquivo Simples.class.

Na seqüência podemos ver o exemplo que monta as estruturas do arquivo *class* utilizando a API BCEL:

```
public static void main(String[] args) throws
        ClassFormatException, IOException {
```

```
        JavaClass mainClass = new ClassParser( "C:\\Simples.class" ).parse();
        System.out.println( mainClass );

        for ( Method method : mainClass.getMethods() ) {
            System.out.println( method );
            System.out.println( method.getCode() );
        }
}
```

O resultado da execução do exemplo pode assim ser analisado:

```
public class Simples extends java.lang.Object
filename        C:\Simples.class
compiled from           Simples.java
compiler version 45.3
access flags            33
constant pool           27 entries
ACC_SUPER flag                  true

Attribute(s):
        SourceFile(Simples.java)

3 methods:
        public void <init>()
        public static int somar(int x, int y)
        public static void main(String[] args)

public void <init>()
Code(max_stack = 1, max_locals = 1, code_length = 5)
0:   aload_0
1:   invokespecial      java.lang.Object.<init> ()V (8)
4:   return

Attribute(s) =
LineNumber(0, 1)
LocalVariable(start_pc = 0, length = 5, index = 0:Simples this)

public static int somar(int x, int y)
Code(max_stack = 2, max_locals = 2, code_length = 4)
0:   iload_0
1:   iload_1
2:   iadd
3:   ireturn

Attribute(s) =
LineNumber(0, 4)
LocalVariable(start_pc = 0, length = 4, index = 0:int x)
LocalVariable(start_pc = 0, length = 4, index = 1:int y)

public static void main(String[] args)
Code(max_stack = 2, max_locals = 2, code_length = 7)
0:   iconst_1
1:   iconst_2
2:   invokestatic  Simples.somar (II)I (21)
5:   istore_1
6:   return
```

```
Attribute(s) =
LineNumber(0, 7), LineNumber(6, 8)
LocalVariable(start_pc = 0, length = 7, index = 0:String[] args)
LocalVariable(start_pc = 6, length = 1, index = 1:int x)
```

Observou-se com os testes realizados que as informações geradas são mais abundantes e de fácil manuseio.

O trabalho apresentado tem como propósito a obtenção de uma maneira eficiente de traduzir o código fonte, no caso o bytecode Java, para o código objeto representado pela LLVM, usando neste caso a biblioteca acima citada para obter-se as instruções da máquina virtual Java.

O próximo passo é julgar as informações obtidas com o uso do BCEL, que é de extrema importância para a organização das informações obtidas, de forma que seja possível gerar o código alvo, ou seja, a representação LLVM.

Aprofundando mais os estudos no uso da API do BCEL, averiguamos que este implementava o padrão de projeto *visitante* e que de acordo com o artigo do Wikipedia é uma solução para separar o algoritmo da estrutura e uma das vantagens é a habilidade de adicionar novas operações.

Assim foi possível "visitar" toda a estrutura de *classfile* sem alterá-lo em nada. Para isso, foi criada uma classe denominada *ClassfileVisitor* (visitante do classfile) que estende e implementa "visitantes" existentes na API, tornando possível passar por todas as estruturas que compõem o arquivo de maneira simples e sem alterar em nada o fluxo.

Nem todas as informações disponibilizadas pelo BCEL nos eram úteis portanto, foi necessário filtrar um pouco as informações, gerando assim um arquivo fonte mais enxuto e conciso. Porém, isso não foi o suficiente para gerar o código alvo para a LLVM. Foi aí que pensamos no uso de um gerador de analisadores léxico e sintáticos, mas para isso era preciso, antes de mais nada, de uma gramática formalizada, para então termos os analisadores e a geração de código.

Infelizmente, a única informação relevante obtida através da especificação da máquina virtual do Java foi a estrutura do arquivo *classfile*, mas isso ainda não foi suficiente para dar de entrada em nenhum gerador de analisadores. Foi assim que criamos nossa própria gramática contendo simplesmente as estruturas essenciais para gerar o código para a máquina alvo.

5.3 Gramática experimental do *Classfile*

Antes de criar uma gramática formal foi necessário escolher um gerador de analisadores. Para isso, nada melhor do que trabalhar com algo já compreendido e dominado. Pela familiaridade com o GALS, e pelo fato do orientando ter utilizado o mesmo durante a disciplina de introdução a compiladores resolveu-se optar por esta ferramenta.

5.3.1 Gerador de analisadores Léxicos e Sintáticos (GALS)

O GALS foi o resultado de um trabalho de conclusão de curso realizado pelo acadêmico Carlos Eduardo Gesser e orientado pelo professor Olinto José Varela Furtado onde nos baseamos para a realização deste descritivo.

De acordo com Gesser "O GALS é uma ferramenta para geração automática de analisadores léxicos e sintáticos, duas importantes fases do processo de compilação. Foi desenvolvida para ser uma ferramenta com propósitos didáticos, mas com características profissionais, podendo ser utilizada tanto no auxílio aos alunos da cadeira de Construção de Compiladores como possivelmente em outros projetos que necessitem processamento de linguagens."

Para que o mesmo gerasse os analisadores necessários era preciso dar de entrada nele quatro conjuntos e informações: definições regulares, *tokens*, não-terminais e a gramática propriamente dita.

De acordo com Gesser, Os *tokens* são fundamentais para se gerar o analisador léxico. O analisador gerado funciona simulando um autômato finito, rodando em cima de uma tabela de transições. O analisador verifica o próximo caractere da entrada e o estado atual do autômato (inicialmente zero) e move o autômato para seu próximo estado. Se, eventualmente, o autômato chegar a um estado final sempre correspondente a um *token*, ele ainda não pode dar a análise deste *token* como concluída, pois é preciso tentar identificar a seqüência de caracteres mais longa possível. Assim, o analisador somente pára quando não consegue mais prosseguir na tabela de transições. Se durante este processo ele encontrou algum *token*, ele produz o *token* correspondente ao último estado final alcançado (a seqüência mais longa de caracteres). Se nenhum *token* foi encontrado então um erro léxico é gerado.

A especificação sintática é constituída de três partes: definição dos símbolos terminais (*tokens*), definição dos símbolos não-terminais, e definição das produções gramaticais.

Abaixo podemos ver as informações de entrada fornecidas a ferramenta GALS para gerar os analisadores necessários para geração do código alvo.

- **Definições regulares**

```
J : [a-zA-ZáàâéêíóôüçÁÀÂÉÊÍÓÔÜÇ]
L : [a-z]
D : [0-9]
E : "_"
S : "-"
P : "."
B : "/" | "(" | ")" | "<" | ">"
ASPA : \"
esp_char : [\ \t\n\r]
```

- **Tokens**

```
identificador: {L}+ {D}?{E}?(({L}|{D})+)?
literal: {ASPA} (({J}|{B}|{E}|{S}|{D})+)? {ASPA}
```

```
inteiro : {S}?{D}+
real : {S}?{D}+ ({P} ({D})+)?

inicio = identificador : "inicio"
fim = identificador : "fim"
inicioprograma = identificador : "inicioprograma"
fimprograma = identificador : "fimprograma"

int = identificador : "int"
double = identificador : "double"
float = identificador : "float"
bool = identificador : "bool"
string = identificador : "string"
void = identificador : "void"

nop = identificador : "nop"
aconst_null = identificador : "aconst_null"
iconst_m1 = identificador : "iconst_m1"
iconst_0 = identificador : "iconst_0"
iconst_1 = identificador : "iconst_1"
iconst_2 = identificador : "iconst_2"
iconst_3 = identificador : "iconst_3"
iconst_4 = identificador : "iconst_4"
iconst_5 = identificador : "iconst_5"
lconst_0 = identificador : "lconst_0"
lconst_1 = identificador : "lconst_1"
fconst_0 = identificador : "fconst_0"
fconst_1 = identificador : "fconst_1"
fconst_2 = identificador : "fconst_2"
dconst_0 = identificador : "dconst_0"
dconst_1 = identificador : "dconst_1"
bipush = identificador : "bipush"
sipush = identificador : "sipush"
ldc = identificador : "ldc"
ldc_w = identificador : "ldc_w"
ldc2_w = identificador : "ldc2_w"
iload = identificador : "iload"
lload = identificador : "lload"
fload = identificador : "fload"
dload = identificador : "dload"
aload = identificador : "aload"
iload_0 = identificador : "iload_0"
iload_1 = identificador : "iload_1"
iload_2 = identificador : "iload_2"
iload_3 = identificador : "iload_3"
lload_0 = identificador : "lload_0"
lload_1 = identificador : "lload_1"
lload_2 = identificador : "lload_2"
lload_3 = identificador : "lload_3"
fload_0 = identificador : "fload_0"
fload_1 = identificador : "fload_1"
fload_2 = identificador : "fload_2"
fload_3 = identificador : "fload_3"
dload_0 = identificador : "dload_0"
dload_1 = identificador : "dload_1"
dload_2 = identificador : "dload_2"
dload_3 = identificador : "dload_3"
aload_0 = identificador : "aload_0"
aload_1 = identificador : "aload_1"
aload_2 = identificador : "aload_2"
```

aload_3 = identificador : "aload_3"
iaload = identificador : "iaload"
laload = identificador : "laload"
faload = identificador : "faload"
daload = identificador : "daload"
aaload = identificador : "aaload"
baload = identificador : "baload"
caload = identificador : "caload"
saload = identificador : "saload"
istore = identificador : "istore"
lstore = identificador : "lstore"
fstore = identificador : "fstore"
dstore = identificador : "dstore"
astore = identificador : "astore"
istore_0 = identificador : "istore_0"
istore_1 = identificador : "istore_1"
istore_2 = identificador : "istore_2"
istore_3 = identificador : "istore_3"
lstore_0 = identificador : "lstore_0"
lstore_1 = identificador : "lstore_1"
lstore_2 = identificador : "lstore_2"
lstore_3 = identificador : "lstore_3"
fstore_0 = identificador : "fstore_0"
fstore_1 = identificador : "fstore_1"
fstore_2 = identificador : "fstore_2"
fstore_3 = identificador : "fstore_3"
dstore_0 = identificador : "dstore_0"
dstore_1 = identificador : "dstore_1"
dstore_2 = identificador : "dstore_2"
dstore_3 = identificador : "dstore_3"
astore_0 = identificador : "astore_0"
astore_1 = identificador : "astore_1"
astore_2 = identificador : "astore_2"
astore_3 = identificador : "astore_3"
iastore = identificador : "iastore"
lastore = identificador : "lastore"
fastore = identificador : "fastore"
dastore = identificador : "dastore"
aastore = identificador : "aastore"
bastore = identificador : "bastore"
castore = identificador : "castore"
sastore = identificador : "sastore"
pop = identificador : "pop"
pop2 = identificador : "pop2"
dup = identificador : "dup"
dup_x1 = identificador : "dup_x1"
dup_x2 = identificador : "dup_x2"
dup2 = identificador : "dup2"
dup2_x1 = identificador : "dup2_x1"
dup2_x2 = identificador : "dup2_x2"
swap = identificador : "swap"
iadd = identificador : "iadd"
ladd = identificador : "ladd"
fadd = identificador : "fadd"
dadd = identificador : "dadd"
isub = identificador : "isub"
lsub = identificador : "lsub"
fsub = identificador : "fsub"
dsub = identificador : "dsub"
imul = identificador : "imul"

lmul = identificador : "lmul"
fmul = identificador : "fmul"
dmul = identificador : "dmul"
idiv = identificador : "idiv"
ldiv = identificador : "ldiv"
fdiv = identificador : "fdiv"
ddiv = identificador : "ddiv"
irem = identificador : "irem"
lrem = identificador : "lrem"
frem = identificador : "frem"
drem = identificador : "drem"
ineg = identificador : "ineg"
lneg = identificador : "lneg"
fneg = identificador : "fneg"
dneg = identificador : "dneg"
ishl = identificador : "ishl"
lshl = identificador : "lshl"
ishr = identificador : "ishr"
lshr = identificador : "lshr"
iushr = identificador : "iushr"
lushr = identificador : "lushr"
iand = identificador : "iand"
land = identificador : "land"
ior = identificador : "ior"
lor = identificador : "lor"
ixor = identificador : "ixor"
lxor = identificador : "lxor"
iinc = identificador : "iinc"
i2l = identificador : "i2l"
i2f = identificador : "i2f"
i2d = identificador : "i2d"
l2i = identificador : "l2i"
l2f = identificador : "l2f"
l2d = identificador : "l2d"
f2i = identificador : "f2i"
f2l = identificador : "f2l"
f2d = identificador : "f2d"
d2i = identificador : "d2i"
d2l = identificador : "d2l"
d2f = identificador : "d2f"
i2b = identificador : "i2b"
i2c = identificador : "i2c"
i2s = identificador : "i2s"
lcmp = identificador : "lcmp"
fcmpl = identificador : "fcmpl"
fcmpg = identificador : "fcmpg"
dcmpl = identificador : "dcmpl"
dcmpg = identificador : "dcmpg"
ifeq = identificador : "ifeq"
ifne = identificador : "ifne"
iflt = identificador : "iflt"
ifge = identificador : "ifge"
ifgt = identificador : "ifgt"
ifle = identificador : "ifle"
if_icmpeq = identificador : "if_icmpeq"
if_icmpne = identificador : "if_icmpne"
if_icmplt = identificador : "if_icmplt"
if_icmpge = identificador : "if_icmpge"
if_icmpgt = identificador : "if_icmpgt"
if_icmple = identificador : "if_icmple"

```
if_acmpeq = identificador : "if_acmpeq"
if_acmpne = identificador : "if_acmpne"
goto = identificador : "goto"
jsr = identificador : "jsr"
ret = identificador : "ret"
tableswitch = identificador : "tableswitch"
lookupswitch = identificador : "lookupswitch"
ireturn = identificador : "ireturn"
lreturn = identificador : "lreturn"
freturn = identificador : "freturn"
dreturn = identificador : "dreturn"
areturn = identificador : "areturn"
return = identificador : "return"
getstatic = identificador : "getstatic"
putstatic = identificador : "putstatic"
getfield = identificador : "getfield"
putfield = identificador : "putfield"
invokevirtual = identificador : "invokevirtual"
invokespecial = identificador : "invokespecial"
invokestatic = identificador : "invokestatic"
invokeinterface = identificador : "invokeinterface"
new = identificador : "new"
newarray = identificador : "newarray"
anewarray = identificador : "anewarray"
arraylength = identificador : "arraylength"
athrow = identificador : "athrow"
checkcast = identificador : "checkcast"
instanceof = identificador : "instanceof"
monitorenter = identificador : "monitorenter"
monitorexit = identificador : "monitorexit"
wide = identificador : "wide"
multianewarray = identificador : "multianewarray"
ifnull = identificador : "ifnull"
ifnonnull = identificador : "ifnonnull"
goto_w = identificador : "goto_w"
jsr_w = identificador : "jsr_w"
breakpoint = identificador : "breakpoint"
impdep1 = identificador : "impdep1"
impdep2 = identificador : "impdep2"

"\n"
":"
";"
","
"("
")"
"."
"'"
"["
"]"

: {esp_char}*
```

- **Não-terminais**

```
<corpo>
<bloco>
<listafuncao>
<funcao>
```

<nome_funcao>
<parametros>
<argumentos>
<rep_argumentos>
<tipo>
<retorno>
<rep_parametros>
<listacomando>
<chamada_metodo>
<comando>
<resto_comando>
<constante_explicita>
<operacao>
<opcode>
<label>
<desvio>
<incremento>

- **Gramática**

<corpo> ::= inicioprograma #1 <bloco> fimprograma #2;

<bloco> ::= <listafuncao>;

<listafuncao> ::= <funcao> <listafuncao> | î;

<funcao> ::= inicio #3 <nome_funcao> <listacomando> fim #4;

<nome_funcao> ::= identificador #5 "("<parametros>")" ":" <retorno> #23 ;

<retorno> ::= #22 <tipo> | #22 void #6;

<tipo> ::= int #7 | double #8 | bool #9 | string #10 | float #11;

<parametros> ::= #21 <tipo> identificador #12 <rep_parametros> | î;

<rep_parametros> ::= "," #21 <tipo> identificador #12 <rep_parametros> | î;

<listacomando> ::= <operacao> <listacomando> | î ;

<operacao> ::= <comando> ";";

<comando> ::= <label> <opcode> <resto_comando>;

<label> ::= "[" inteiro #24 "]";

<resto_comando> ::= #13 <constante_explicita> |
#14 <chamada_metodo> |
#25 <desvio> |
<incremento> |
î;

<incremento> ::= "[" inteiro #27 ";" inteiro #28 "]";

<desvio> ::= "(" inteiro #26 ")";

<constante_explicita> ::= inteiro #15 | literal #16 | real #17;

<chamada_metodo> ::= identificador #18 "(" <argumentos> ")";

<argumentos> ::= <tipo> #19 <rep_argumentos> #20 | î;

<rep_argumentos> ::= "," <tipo> #19 <rep_argumentos> #20 | î;

<opcode> ::= nop #100 | aconst_null #101 | iconst_m1 #102 | iconst_0 #103 | iconst_1 #104 |
iconst_2 #105 | iconst_3 #106 | iconst_4 #107 | iconst_5 #108 | lconst_0 #109 |
lconst_1 #110 | fconst_0 #111 | fconst_1 #112 | fconst_2 #113 | dconst_0 #114 |
dconst_1 #115 | bipush #116 | sipush #117 | ldc #118 | ldc_w #119 | ldc2_w #120 |
iload #121 | lload #122 | fload #123 | dload #124 | aload #125 | iload_0 #126 |
iload_1 #127 | iload_2 #128 | iload_3 #129 | lload_0 #130 | lload_1 #131 |
lload_2 #132 | lload_3 #133 | fload_0 #134 | fload_1 #135 | fload_2 #136 |
fload_3 #137 | dload_0 #138 | dload_1 #139 | dload_2 #140 | dload_3 #141 |
aload_0 #142 | aload_1 #143 | aload_2 #144 | aload_3 #145 | iaload #146 |
laload #147 | faload #148 | daload #149 | aaload #150 | baload #151 |
caload #152 | saload #153 | istore #154 | lstore #155 | fstore #156 |
dstore #157 | astore #158 | istore_0 #159 | istore_1 #160 | istore_2 #161 |
istore_3 #162 | lstore_0 #163 | lstore_1 #164 | lstore_2 #165 | lstore_3 #166 |
fstore_0 #167 | fstore_1 #168 | fstore_2 #169 | fstore_3 #170 | dstore_0 #171 |
dstore_1 #172 | dstore_2 #173 | dstore_3 #174 | astore_0 #175 | astore_1 #176 |
astore_2 #177 | astore_3 #178 | iastore #179 | lastore #180 | fastore #181 |
dastore #182 | aastore #183 | bastore #184 | castore #185 |sastore #186 |
pop #187 | pop2 #188 | dup #189 | dup_x1 #190 | dup_x2 #191 | dup2 #192 |
dup2_x1 #193 | dup2_x2 #194 | swap #195 | iadd #196 | ladd #197 | fadd #198 |
dadd #199 | isub #200 | lsub #201 | fsub #202 | dsub #203 | imul #204 | lmul #205 |
fmul #206 | dmul #207 | idiv #208 | ldiv #209 | fdiv #210 | ddiv #211 | irem #212 |
lrem #213 | frem #214 | drem #215 | ineg #216 | lneg #217 | fneg #218 | dneg #219 |
ishl #220 | lshl #221 | ishr #222 | lshr #223 | iushr #224 | lushr #225 | iand #226 |
land #227 | ior #228 | lor #229 | ixor #230 | lxor #231 | iinc #232 | i2l #233 |
i2f #234 | i2d #235 | l2i #236 | l2f #237 | l2d #238 | f2i #239 | f2l #240 |
f2d #241 | d2i #242 | d2l #243 | d2f #244 | i2b #245 | i2c #246 | i2s #247 |
lcmp #248 | fcmpl #249 | fcmpg #250 | dcmpl #251 | dcmpg #252 | ifeq #253 | ifne #254 |
iflt #255 | ifge #256 | ifgt #257 | ifle #258 | if_icmpeq #259 | if_icmpne #260 |
if_icmplt #261 | if_icmpge #262 | if_icmpgt #263 | if_icmple #264 | if_acmpeq #265 |
if_acmpne #266 | goto #267 | jsr #268 | ret #269 | tableswitch #270 | lookupswitch #271 |
ireturn #272 | lreturn #273 | freturn #274 | dreturn #275 | areturn #276 | return #277 |
getstatic #278 | putstatic #279 | getfield #280 | putfield #281 | invokevirtual #282 |
invokespecial #283 | invokestatic #284 | invokeinterface #285 | new #286 | newarray #287 |
anewarray #288 | arraylength #289 | athrow #290 | checkcast #291 | instanceof #292 |
monitorenter #293 | monitorexit #294 | wide #295 | multianewarray #296 | ifnull #297 |
ifnonnull #298 | goto_w #299 | jsr_w #300 | breakpoint #301 | impdep1 #302 | impdep2 #303;

Feito isso foi possível gerar os analisadores léxicos e sintáticos restando apenas a necessidade de implementar o analisador semântico que na prática seria nosso gerador de código.

5.3.2 Geração do código fonte

Não bastava apenas definir uma gramática para o *classfile* e submeter de entrada para os analisadores as informações geradas pelo *ClassfileVisitor*. Era necessário adaptar nossas construções gramaticais ao mesmo, pois vários símbolos foram acrescentados com o intuito de auxiliar na geração do código.

Para exemplificar as mudanças sofridas no *ClassfileVisitor* mostraremos um arquivo fonte escrito na linguagem Java e a saída correspondente gerada pelo *ClassfileVisitor.*

Abaixo, uma simples classe java (*Simples.java*) com dois métodos estáticos um que retorna o valor de uma posição de um *array* e outro que efetua duas operações matemáticas.

```
public class Simples {

        public static int teste2() {

                int[] vars = new int[3];
                vars[0] = 2;
                vars[1] = 2;
                vars[2] = 2;

                return vars[2];
        }

        public static int teste( int x, int y ) {

                int z = 19;
                int w = 19;

                return (x-y)*(z-w);
        }

        public static void main( String[] args ) {

                teste( teste2(), 42 );
        }
}
```

Resulado gerado pelo *ClassfileVisitor:*

```
inicioprograma

inicio teste2():int
[0]iconst_3 ;
[1]newarray ;
[3]astore_0 ;
[4]aload_0 ;
[5]iconst_0 ;
[6]iconst_2 ;
[7]iastore ;
[8]aload_0 ;
[9]iconst_1 ;
[10]iconst_2 ;
[11]iastore ;
[12]aload_0 ;
[13]iconst_2 ;
[14]iconst_2 ;
[15]iastore ;
[16]aload_0 ;
[17]iconst_2 ;
```

```
[18]iaload ;
[19]ireturn ;
fim

inicio teste(int x,int y):int
[0]bipush 19;
[2]istore_2 ;
[3]bipush 19;
[5]istore_3 ;
[6]iload_0 ;
[7]iload_1 ;
[8]isub ;
[9]iload_2 ;
[10]iload_3 ;
[11]isub ;
[12]imul ;
[13]ireturn ;
fim

inicio main():void
[0]invokestatic teste2();
[3]bipush 42;
[5]invokestatic teste(int,int);
[8]pop ;
[9]return ;
fim

fimprograma
```

Como podemos ver várias modificações foram realizadas se compararmos o exemplo acima com o exemplo inicialmente demostrado da capacidade de geração do BCEL. Isso porque o adaptamos totalmente as nossas necessidades, visando a gramática criada no GALS.

Com o código fonte em mãos bastava trabalhar na ferramenta capaz de realizar a transição, ou seja, gerar o código específico para máquina LLVM. Como visto anteriormente, nos detalhamentos do GALS, o responsável por isso seria o analisador semântico.

5.3.3 Geração do código alvo

O GALS com dois poderosos analisadores:

- Analisador Léxico: de acordo com Gesser, trata os aspectos léxicos do programa. Entra em cena o analisador léxico, também conhecido como *scanner*. Ele lê o texto do programa e o divide em *tokens*, que são os símbolos básicos de uma linguagem de programação e representam palavras reservadas, identificadores, literais numéricos e de texto, operadores e pontuações.

- Analisador Sintático: para Gesser, trabalha lendo os tokens gerados pelo analisador léxico e os agrupando de acordo com a estrutura sintática da linguagem. O resultado final é chamado de árvore de derivação.

Cabe a nós apenas implementar um analisador, o semântico, que leva em consideração o sentido do programa. É através da semântica que são extraídas as informações que possibilitam a

posterior geração de código.

Os aspectos semânticos não são facilmente especificáveis. Os principais mecanismos formais para a especificação dos aspectos semânticos são as gramáticas de atributos, as semânticas denotacionais e as semânticas de ações. Porém, sua complexidade os torna inviáveis na prática e acaba-se partindo para mecanismos semi-formais, ou até mesmo informais.

Uma implementação muito comum é a utilização de ações semânticas que são inseridas no meio da especificação sintática sob a forma de sub-rotinas que são invocadas quando o parser as atinge. Observando a gramática fornecida ao GALS (anteriormente neste capítulo), podemos vê-las.

Nosso analisador recebe um inteiro identificando a ação a ser executada e um *token* reconhecido pelo analisador léxico e passado pelo sintático. A maneira escolhida para tratar cada ação foi criar uma classe para cada.

Para traduzir a seqüência de operações da máquina virtual em código LLVM criamos diversas pilhas:

- de *labels*: para idendificar alvos de desvios;
- de inteiros: para carregar constantes que serão usadas nas chamadas de métodos, operações aritméticas, etc;
- de literais: para carregar constantes;
- de reais: também para carregar constantes e,
- de instruções: para carregar variavéis e para identificar ações, pois algumas vezes são necessários mais de um código de operação da máquina virtual Java para gerar um única operação da máquina LLVM.

Além das pilhas descritas também trabalhamos com uma lista de métodos, sendo que cada um destes itens é composto por um lista de comandos a serem executados (o corpo do método), uma lista de atributos aceitos (nome e tipo de cada um) e o retorno esperado.

Antes de gerar o código motamos a classe como um todo, ou seja, cada método, um a um, para depois gerar todo o código observado.

Abaixo podemos verificar o código alvo gerado para a máquina LLVM que tem como fonte o exemplo listado anteriormente.

```
define i32 @teste()
{
entry:
   %intvar2 = alloca i32
   %ptr2 = getelementptr i32* %intvar2
   store i32 19, i32* %ptr2
   %intvar3 = alloca i32
   %ptr3 = getelementptr i32* %intvar3
   store i32 19, i32* %ptr3
   %ptr1temp1 = load i32* %ptr1
   %ptr0temp2 = load i32* %ptr0
   %temp0 = sub i32   %ptr0temp2,%ptr1temp1
   %ptr3temp4 = load i32* %ptr3
   %ptr2temp5 = load i32* %ptr2
   %temp3 = sub i32   %ptr2temp5,%ptr3temp4
```

```
   %temp6 = mul i32   %temp0,%temp3
   ret i32   %temp6
}

define i32 @teste2()
{
entry:
   %array0 =    alloca i32, i32 3
   %arrayptr0= getelementptr i32* %array0, i32 0
   store i32 2, i32* %arrayptr0
   %arrayptr1= getelementptr i32* %array0, i32 1
   store i32 2, i32* %arrayptr1
   %arrayptr2= getelementptr i32* %array0, i32 2
   store i32 2, i32* %arrayptr2
   %arrayptr3 = getelementptr i32* %array0, i32 2
   %temp1 = load i32* %arrayptr3
   ret i32   %temp1
}

define void @main()
{
entry:
   %retorno0 = tail call i32 @teste2()
   %retorno1 = tail call i32 @teste()
   ret void
}
```

5.4 Ferramenta final, o Tradutor

Com a realização de todas as partes necessárias para desempenhar o processo completo da tradução (a interpretação do bytecode java e tradução do mesmo para o código da representação LLVM), bastava fazer uma ferramenta que fizesse a integração de todas as partes envolvidas.

O resultado foi uma ferramenta muito simples que permite a entrada do código java de duas formas: via interface ou através da abertura de um arquivo. Com a entrada definida seria possível executar o processo de tradução e na seqüência observar o código gerado para a gramática experimental do *classfile,* o código gerado para a máquina alvo na representação LLVM, e o mesmo otimizado com auxílio das ferramentas disponíveis.

Na figura 2 podemos ver a interface durante a execução de uma tradução simples.

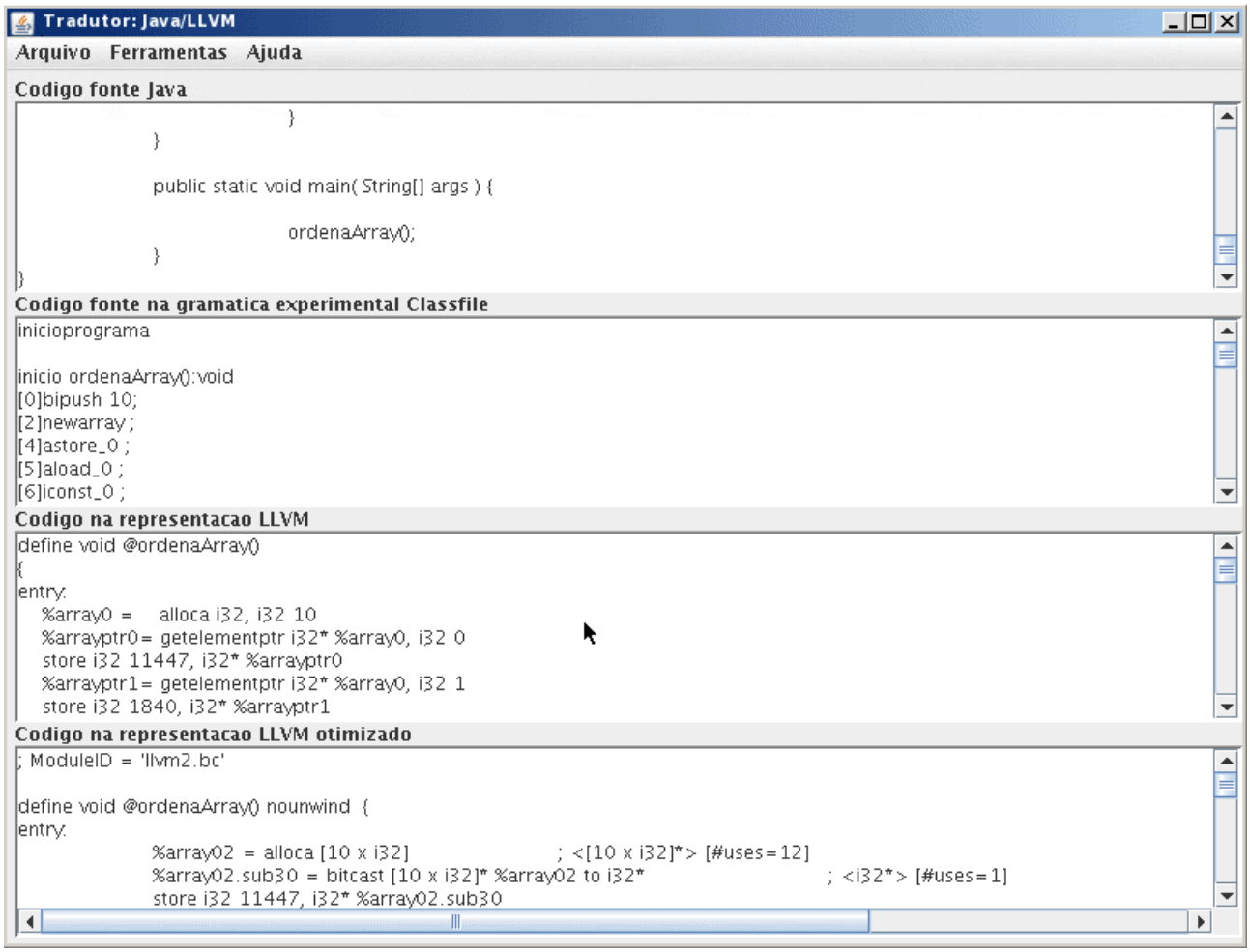

Figura 2: Interface do Tradutor

É possível observar na figura 2 as distintas áreas da interface gráfica:

- *Código fonte Java* sendo a área da interface onde o usuário escreve seu programa na linguagem Java.

- *Código fonte na gramática experimental Classfile* é a área onde o aplicativo adiciona o código fonte gerado pelo tradutor que será usado para gerar o código na representação LLVM.

- *Código na representação LLVM* sendo a área onde a ferramenta disponibilizará o código gerado na representação LLVM.

- *Código na representação LLVM otimizado* é a área onde a ferramenta disponibilizará o código LLVM gerado e otimizado.

O fluxo de funcionamento do tradutor, simplificando a integração entre todas as partes envolvidas, pode ser observado na figura 3.

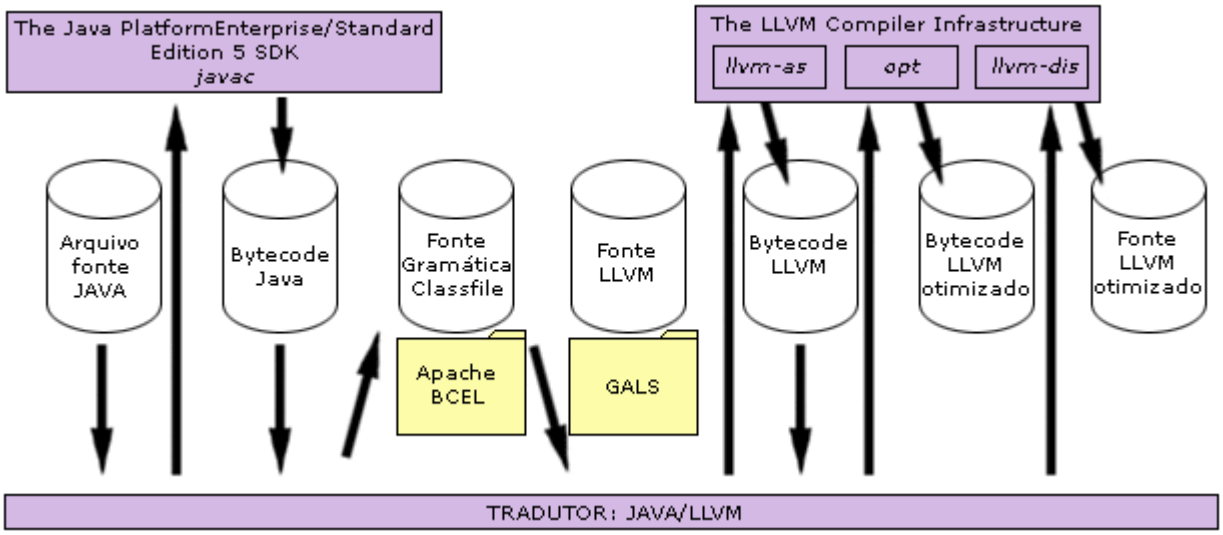

Figura 3: Fluxo da tradução

Ainda sobre figura 3 podemos ressaltar alguns aspectos:

- *Arquivo fonte JAVA* se refere tanto a um programa escrito diretamente na ferramenta quanto algum arquivo *.java* aberto. É importante ressaltar que como o propósito deste trabalho foi meramente apresentar a viabilidade da geração de código na representação LLVM, tendo como fonte um programa escrito na linguagem Java, esta não apresenta suporte para todas as características previstas pela linguagem, mas apenas o mínimo necessário para validar o código gerado.

- *The Java Platform Enterprise/Standard Edition 5 SDK* se refere ao kit de desenvolvimento fornecediro pela Sun Microsystems para criação, compilação e depuração (entre outros propósitos) dos arquivos Java. A ferramenta *javac* disponibilizada pelo tal kit é a ferramenta responsável pela compilação do arquivo fonte Java em bytecode Java. É imprescindível que este kit suporte a versão 1.5 do Java.

- *Fonte Gramática Classfile* é o arquivo gerado pela a classe (*ClassfileVisitor)* que utiliza as funcionalidades da *API* do *Apache BCEL* para gerar o código intermediário na gramática experimental do classfile.

- *Fonte LLVM* é o arquivo que foi gerado utilizando os analisadores gerados pela ferramenta *GALS*. Ressalta-se que o analisador semântico foi inteiramente desenvolvido, pois o *GALS* gera somente o léxico e o sintático.

- *Bytecode LLVM* é gerado pelo uso da ferramenta *llvm-as* da infra-estrutura do LLVM. O propósito é compilar o arquivo fonte (em assembly próprio) para bytecode LLVM.

- *Bytecode LLVM otimizado* é gerado pela ferramenta *opt* que também é disponibilizada pela infra-estrutura da LLVM. Sua finalidade é otimizar um um arquivo no formato bytecode LLVM aplicando as otimizações informadas e gerando um outro arquivo bytecode LLVM. Cabe explicar que dentre as opções da ferramenta de otimização *opt* estamos usando

somente *adce*, *codegenprepare* e std-compile-opts.

- *Fonte LLVM otimizado* é gerado pela ferramenta *llvm-dis*, também da infra-estrutura LLVM, que tem como propósito gerar um arquivo no formato assembly LLVM a partir de um arquivo em bytecode LLVM.

Com a ferramenta criada foi possível escrever programas na linguagem Java e analisar como o mesmo ficou na representação da linguagem da máquina virtual LLVM.

5.4.1 Validação do código gerado

Com o intuito de validar a qualidade do código gerado na representação LLVM foi decidido fazer um programa muito simples de ordenação de um array de 10 posições utilizando o algoritmo *bubble sort* que, de acordo com o a enciclopédia Wikipédia, tem como idéia base "percorrer o vetor diversas vezes e a cada passagem fazer flutuar para o topo o menor elemento da seqüência".

Abaixo podemos ver o código fonte escrito em Java que foi submetido como entrada para a validação da ferramenta.

```
public class OrdenacaoArray {

	public static void ordenaArray() {

		int array[] = new int[10];
		array[0] = 11447;
		array[1] = 1840;
		array[2] = 2754;
		array[3] = 4637;
		array[4] = 2128;
		array[5] = 1573;
		array[6] = 2816;
		array[7] = 1770;
		array[8] = 2014;
		array[9] = 1657;

		for ( int i = 0 ; i < array.length -1; i ++ ) {

			for ( int j = 0 ; j < array.length -i-1 ; j ++ ) {

				if ( array[j] > array[j+1] ) {

					int temp = array[j];
					array[j] = array[j+1];
					array[j+1] = temp;
				}
			}
		}
	}

	public static void main( String[] args ) {

		ordenaArray();
	}
```

```
}
```

Dada a entrada seleciou-se a opção de tradução (no menu de ferramentas) disponibilizada pelo aplicativo. O resultado foi o código na representação LLVM otimizado o qual segue abaixo.

```
; ModuleID = 'llvm2.bc'

define void @ordenaArray() nounwind  {
entry:
        %array02 = alloca [10 x i32]                    ; <[10 x i32]*> [#uses=12]
        %array02.sub30 = bitcast [10 x i32]* %array02 to i32*                   ;       <i32*>
[#uses=1]
        store i32 11447, i32* %array02.sub30
        %arrayptr1 = getelementptr [10 x i32]* %array02, i32 0, i32 1                   ;
<i32*> [#uses=1]
        store i32 1840, i32* %arrayptr1
        %arrayptr2 = getelementptr [10 x i32]* %array02, i32 0, i32 2                   ;
<i32*> [#uses=1]
        store i32 2754, i32* %arrayptr2
        %arrayptr3 = getelementptr [10 x i32]* %array02, i32 0, i32 3                   ;
<i32*> [#uses=1]
        store i32 4637, i32* %arrayptr3
        %arrayptr4 = getelementptr [10 x i32]* %array02, i32 0, i32 4                   ;
<i32*> [#uses=1]
        store i32 2128, i32* %arrayptr4
        %arrayptr5 = getelementptr [10 x i32]* %array02, i32 0, i32 5                   ;
<i32*> [#uses=1]
        store i32 1573, i32* %arrayptr5
        %arrayptr6 = getelementptr [10 x i32]* %array02, i32 0, i32 6                   ;
<i32*> [#uses=1]
        store i32 2816, i32* %arrayptr6
        %arrayptr7 = getelementptr [10 x i32]* %array02, i32 0, i32 7                   ;
<i32*> [#uses=1]
        store i32 1770, i32* %arrayptr7
        %arrayptr8 = getelementptr [10 x i32]* %array02, i32 0, i32 8                   ;
<i32*> [#uses=1]
        store i32 2014, i32* %arrayptr8
        %arrayptr9 = getelementptr [10 x i32]* %array02, i32 0, i32 9                   ;
<i32*> [#uses=1]
        store i32 1657, i32* %arrayptr9
```

```
        br label %Seq0

Seq0:           ; preds = %Label126.Seq0_crit_edge, %entry
        %indvar18 = phi i32 [ 0, %entry ], [ %indvar.next28, %Label126.Seq0_crit_edge ]
                ; <i32> [#uses=2]
        %intvar2 = alloca i32                   ; <i32*> [#uses=2]
        store i32 0, i32* %intvar2
        %temp69 = sub i32 9, %indvar18          ; <i32> [#uses=3]
        %cond811 = icmp sgt i32 %temp69, 0              ; <i1> [#uses=1]
        br i1 %cond811, label %bb.nph, label %Label126

Label81.Label126_crit_edge:             ; preds = %Label81.backedge
        store i32 %smax23, i32* %intvar2
        br label %Label126

bb.nph:         ; preds = %Seq0
        %tmp22 = icmp slt i32 %temp69, 1                ; <i1> [#uses=1]
        %smax23 = select i1 %tmp22, i32 1, i32 %temp69          ; <i32> [#uses=2]
        br label %Seq1

Seq1:           ; preds = %Label81.backedge.Seq1_crit_edge, %bb.nph
        %indvar15 = phi i32 [ 0, %bb.nph ], [ %indvar.next16,
%Label81.backedge.Seq1_crit_edge ]              ; <i32> [#uses=3]
        %arrayptr10 = getelementptr [10 x i32]* %array02, i32 0, i32 %indvar15
        ; <i32*> [#uses=2]
        %temp9 = load i32* %arrayptr10          ; <i32> [#uses=2]
        %temp11 = add i32 %indvar15, 1          ; <i32> [#uses=1]
        %arrayptr11 = getelementptr [10 x i32]* %array02, i32 0, i32 %temp11
        ; <i32*> [#uses=2]
        %temp13 = load i32* %arrayptr11         ; <i32> [#uses=2]
        %cond14 = icmp sgt i32 %temp9, %temp13          ; <i1> [#uses=1]
        br i1 %cond14, label %Seq2, label %Label81.backedge

Seq2:           ; preds = %Seq1
        store i32 %temp13, i32* %arrayptr10
        store i32 %temp9, i32* %arrayptr11
        br label %Label81.backedge

Label81.backedge:               ; preds = %Seq2, %Seq1
        %indvar.next16 = add i32 %indvar15, 1           ; <i32> [#uses=2]
        %exitcond = icmp eq i32 %indvar.next16, %smax23         ; <i1> [#uses=1]
```

```
        br i1 %exitcond, label %Label81.Label126_crit_edge, label %Label81.backedge.Seq1_crit_edge

Label81.backedge.Seq1_crit_edge:              ; preds = %Label81.backedge
        br label %Seq1

Label126:                                     ; preds = %Label81.Label126_crit_edge, %Seq0
        %indvar.next28 = add i32 %indvar18, 1                 ; <i32> [#uses=2]
        %exitcond29 = icmp eq i32 %indvar.next28, 9           ; <i1> [#uses=1]
        br i1 %exitcond29, label %Label132, label %Label126.Seq0_crit_edge

Label126.Seq0_crit_edge:              ; preds = %Label126
        br label %Seq0

Label132:                                     ; preds = %Label126
        ret void
}

define void @main() nounwind  {
entry:
        %array02.i = alloca [10 x i32]                ; <[10 x i32]*> [#uses=12]
        %array02.sub.i25 = bitcast [10 x i32]* %array02.i to i32*     ; <i32*> [#uses=1]
        store i32 11447, i32* %array02.sub.i25
        %arrayptr1.i = getelementptr [10 x i32]* %array02.i, i32 0, i32 1     ; <i32*> [#uses=1]
        store i32 1840, i32* %arrayptr1.i
        %arrayptr2.i = getelementptr [10 x i32]* %array02.i, i32 0, i32 2     ; <i32*> [#uses=1]
        store i32 2754, i32* %arrayptr2.i
        %arrayptr3.i = getelementptr [10 x i32]* %array02.i, i32 0, i32 3     ; <i32*> [#uses=1]
        store i32 4637, i32* %arrayptr3.i
        %arrayptr4.i = getelementptr [10 x i32]* %array02.i, i32 0, i32 4     ; <i32*> [#uses=1]
        store i32 2128, i32* %arrayptr4.i
        %arrayptr5.i = getelementptr [10 x i32]* %array02.i, i32 0, i32 5     ; <i32*> [#uses=1]
        store i32 1573, i32* %arrayptr5.i
        %arrayptr6.i = getelementptr [10 x i32]* %array02.i, i32 0, i32 6     ; <i32*> [#uses=1]
        store i32 2816, i32* %arrayptr6.i
```

```
	%arrayptr7.i = getelementptr [10 x i32]* %array02.i, i32 0, i32 7		; <i32*> [#uses=1]
	store i32 1770, i32* %arrayptr7.i
	%arrayptr8.i = getelementptr [10 x i32]* %array02.i, i32 0, i32 8		; <i32*> [#uses=1]
	store i32 2014, i32* %arrayptr8.i
	%arrayptr9.i = getelementptr [10 x i32]* %array02.i, i32 0, i32 9		; <i32*> [#uses=1]
	store i32 1657, i32* %arrayptr9.i
	%intvar2.i5 = alloca i32		; <i32*> [#uses=2]
	store i32 0, i32* %intvar2.i5
	br label %bb.nph.i

Seq0.i:		; preds = %Label126.i.Seq0.i_crit_edge, %Label81.Label126_crit_edge.i.Seq0.i_crit_edge
	%indvar = phi i32 [ 0, %Label81.Label126_crit_edge.i.Seq0.i_crit_edge ], [ %indvar.next23, %Label126.i.Seq0.i_crit_edge ]		; <i32> [#uses=2]
	%tmp = add i32 %indvar18.i.reg2mem.0, 1		; <i32> [#uses=1]
	%indvar.next28.i.reg2mem.0 = add i32 %indvar, %tmp		; <i32> [#uses=3]
	%intvar2.i = alloca i32		; <i32*> [#uses=2]
	store i32 0, i32* %intvar2.i
	%temp69.i = sub i32 9, %indvar.next28.i.reg2mem.0		; <i32> [#uses=2]
	%cond811.i = icmp sgt i32 %temp69.i, 0		; <i1> [#uses=1]
	br i1 %cond811.i, label %Seq0.i.bb.nph.i_crit_edge, label %Label126.i

Seq0.i.bb.nph.i_crit_edge:		; preds = %Seq0.i
	br label %bb.nph.i

Label81.Label126_crit_edge.i:		; preds = %Label81.backedge.i, %Seq2.i
	store i32 %smax23.i, i32* %intvar2.i.reg2mem.0
	%exitcond29.i15 = icmp eq i32 %indvar18.i.reg2mem.0, 8		; <i1> [#uses=1]
	br i1 %exitcond29.i15, label %ordenaArray.exit, label %Label81.Label126_crit_edge.i.Seq0.i_crit_edge

Label81.Label126_crit_edge.i.Seq0.i_crit_edge:		; preds = %Label81.Label126_crit_edge.i
	br label %Seq0.i

bb.nph.i:		; preds = %Seq0.i.bb.nph.i_crit_edge, %entry
	%indvar18.i.reg2mem.0 = phi i32 [ 0, %entry ], [ %indvar.next28.i.reg2mem.0, %Seq0.i.bb.nph.i_crit_edge ]		; <i32> [#uses=2]
```

```
        %intvar2.i.reg2mem.0 = phi i32* [ %intvar2.i5, %entry ], [ %intvar2.i, %Seq0.i.bb.nph.i_crit_edge ]          ; <i32*> [#uses=1]
        %temp69.i.reg2mem.0 = phi i32 [ 9, %entry ], [ %temp69.i, %Seq0.i.bb.nph.i_crit_edge ]          ; <i32> [#uses=2]
        %tmp22.i = icmp slt i32 %temp69.i.reg2mem.0, 1          ; <i1> [#uses=1]
        %smax23.i = select i1 %tmp22.i, i32 1, i32 %temp69.i.reg2mem.0          ; <i32> [#uses=3]
        br label %Seq1.i

Seq1.i:         ; preds = %Seq2.i.Seq1.i_crit_edge, %bb.nph.i
        %indvar15.i = phi i32 [ 0, %bb.nph.i ], [ %temp11.i, %Seq2.i.Seq1.i_crit_edge ]          ; <i32> [#uses=2]
        %arrayptr10.i = getelementptr [10 x i32]* %array02.i, i32 0, i32 %indvar15.i          ; <i32*> [#uses=2]
        %temp9.i = load i32* %arrayptr10.i          ; <i32> [#uses=2]
        %temp11.i = add i32 %indvar15.i, 1          ; <i32> [#uses=4]
        %arrayptr11.i = getelementptr [10 x i32]* %array02.i, i32 0, i32 %temp11.i          ; <i32*> [#uses=2]
        %temp13.i = load i32* %arrayptr11.i          ; <i32> [#uses=2]
        %cond14.i = icmp sgt i32 %temp9.i, %temp13.i          ; <i1> [#uses=1]
        br i1 %cond14.i, label %Seq2.i, label %Label81.backedge.i

Seq2.i:         ; preds = %Seq1.i
        icmp eq i32 %temp11.i, %smax23.i          ; <i1>:0 [#uses=1]
        store i32 %temp13.i, i32* %arrayptr10.i
        store i32 %temp9.i, i32* %arrayptr11.i
        br i1 %0, label %Label81.Label126_crit_edge.i, label %Seq2.i.Seq1.i_crit_edge

Seq2.i.Seq1.i_crit_edge:        ; preds = %Label81.backedge.i, %Seq2.i
        br label %Seq1.i

Label81.backedge.i:         ; preds = %Seq1.i
        icmp eq i32 %temp11.i, %smax23.i          ; <i1>:1 [#uses=1]
        br i1 %1, label %Label81.Label126_crit_edge.i, label %Seq2.i.Seq1.i_crit_edge

Label126.i:         ; preds = %Seq0.i
        %exitcond29.i = icmp eq i32 %indvar.next28.i.reg2mem.0, 8          ; <i1> [#uses=1]
        %indvar.next23 = add i32 %indvar, 1          ; <i32> [#uses=1]
        br i1 %exitcond29.i, label %ordenaArray.exit, label %Label126.i.Seq0.i_crit_edge

Label126.i.Seq0.i_crit_edge:        ; preds = %Label126.i
```

```
        br label %Seq0.i

ordenaArray.exit:                 ; preds = %Label126.i, %Label81.Label126_crit_edge.i
        ret void
}
```

6 Conclusões

6.1 Análise comparativa

Com o intuito de averiguar se o código gerado no formato de bytecode da LLVM a partir do bytecode da JVM utilizando as opções disponíveis de otimização do framework LLVM trazia algum benefício no que diz respeito ao uso do processador, escrevemos um programa na linguagem Java que ordenava um array de 6 (seis) mil posições utilizando o algoritmo *bubblesort*.

O primeiro passo foi compilá-lo para o bytecode da JVM e executá-lo.

Durante a execução monitoramos o uso do processador e a média de carga que o sistema realizou. Os resultados obtidos podem ser observados na figura 4 (o uso do processador) e na figura 5 (média de carga).

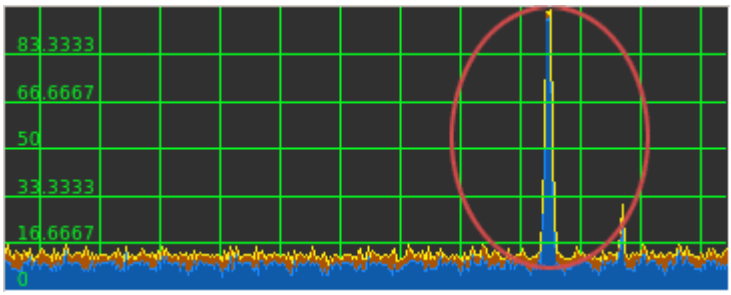

Figura 4: O uso da CPU na execução JVM

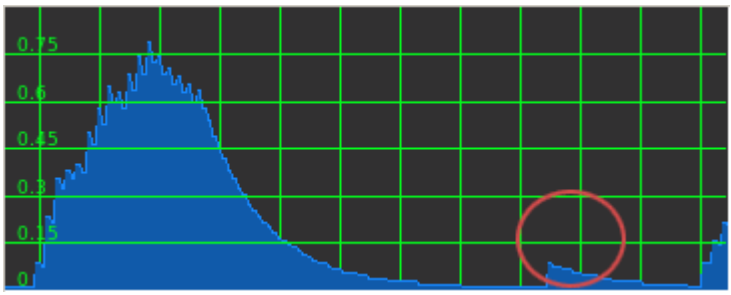

Figura 5: A carga do sistema na execução JVM

Em ambas figuras podemos observar o momento da execução do programa referenciada com a parte circulada em vermelho no gráfico.

O passo seguinte foi utilizar o tradutor para gerar o código no formato de bytecode da LLVM e na seqüência utilizar o comando *lli* disponibilizado pelo framework para executar programas diretamente utilizando um compilador *just-in-time* ou um interpretador dependendo da disponibilidade.

Da mesma forma que foi feito durante a execução do programa sobre a JVM monitoramos a execução do aplicativo utilizando o *lli* sobre o bytecode LLVM. Os resultados obtidos podem ser averiguados, respectivamente, nas figuras 6 e 7.

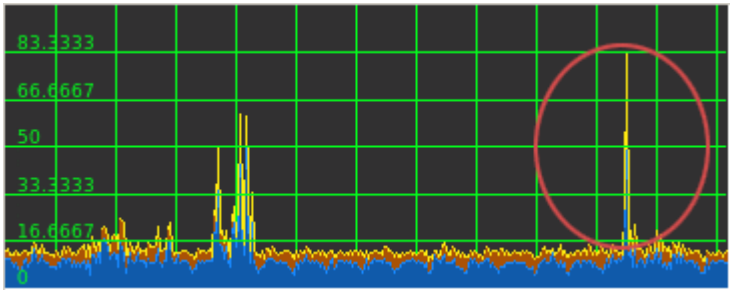

Figura 6: O uso da CPU na execução LLVM

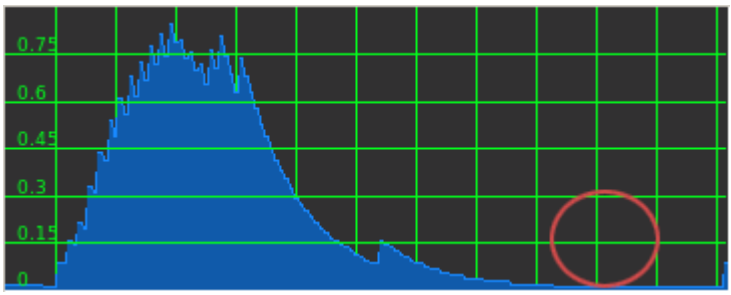

Figura 7: A carga do sistema na execução LLVM

Ao compararmos a figura 6, que representa o uso da CPU durante a execução do programa utilizando o *just-in-time* compiler sobre o bytecode LLVM com a figura 4, que por sua vez representa o mesmo programa sendo executado sobre a máquina JVM, podemos observar uma significativa redução do uso do processador. Enquanto na execução sobre a JVM o pico máximo de uso do processador atingiu o valor 83.33, na execução sobre a LLVM obtivemos um pico máximo de 66.66 o que representa um redução de aproximadamente 20% do uso do processador.

Com relação a carga média do sistema se compararmos a figura 7 (execução sobre a LLVM) com a figura 5 (execução JVM) observamos que a carga do sistema caiu de aproximadamente 0.7 da JVM para 0 da LLVM, o que representou uma queda significativa.

6.2 Considerações Finais

Ao final do presente trabalho constatamos que a ferramenta resultante não realizou todas as possibilidades levantadas pela linguagem Java. Entretanto, consideramos que o presente trabalho foi

de grande valia para tirar conclusões relacionadas à conjectura levantada inicialmente, mas principalmente pelo qual é possível gerar código para a máquina virtual LLVM a partir de códigos escritos na linguagem de programação Java.

Encontramos problemas ao longo do desenvolvimento, principalmente no que diz respeito a geração do código LLVM tendo como fonte a gramática experimental do Classfile, pois precisávamos gerar código tendo como entrada apenas uma seqüência de instruções. Todavia com criatividade e paciência acabamos resolvendo o problema utilizando pilhas e outras estruturas de dados.

6.3 Trabalhos Futuros

Comentado anteriormente da impossibilidade de gerar código para todos os aspectos previstos pela linguagem Java, propomos que um dos trabalhos futuros seria a finalização deste "mapeamento" dando continuidade as ações semânticas que aguardam implementação.

Outra trabalho interessante seria tornar a ferramenta configurável em vários aspectos, principalmente no que diz respeito as otimizações utilizadas na geração do código para a LLVM.

Outra sugestão seria aceitar a outra possível fonte para a geração do código para a máquina LLVM, sendo que este trabalho tratou do bytecode Java e a outra possibilidade seria tratar do próprio código fonte Java.

Referências

LATTNER, C. A. **LLVM:** an infrastructure for multi-stage optimization. 2002. Master of Science in Computer Science - Graduate College of the University of Illinois, Urbana, Illinois.

MACHADO, A. **Análise de Tempo de Execução em alto nível para Sistemas de Tempo Real utilizando-se o framework LLVM**. Trabalho de conclusão do curso de Sistemas de Informação – Universidade Federal de Santa Catarina, Florianópolis, Santa Catarina, Brasil.

AHO, A. V. & LAM, M. S. & SETHI, R. & ULLMAN, J. D. **Compilers:** Principles, Techniques & Tools. Segunda edição. Addison-Wesley, 2007.

LINDHOLM, T. & YELLIN, F. **The JavaTM Virtual Machine Specification Second Edition.** Segunda edição. Addison-Wesley, 1999. (0-201-43294-3).

GOSLING, J. & JOY, B. & STEELE, G. & BRACH, G. **The JavaTM Language Specification Third Edition**. Terceira edição. Santa Clara, California, U.S.A: Addison-Wesley: 2005. (0-321-24678-0).

GEOFFRAY, N. & THOMAS, G. & CLÉMENT, C. & FOLLIOT, B. A Lazzy Developer Approach: Building a JVM with Third Party Software. Université Pierre et Marie Curie, Paris, France.

LATTNER, C. & ADVE, V. A Compilation Framework for Lifelong Program Analysis and Transformation. University of Illinois at Urbana-Champaign.

CHEN, C. & CHEN, Z. Q. & YANG, W. Translating Java bytecode to X86 assembly code. National Chiao-Tung University, Hsin-Chu, Taiwan.

PROEBSTING, T. A. & TOWNSEND, G. & BRIDGES. P. & HARTMAN, J. H. & NEWSHAM, T. & WATTERSON, S. A. Toba: Java For Applications A Way Ahead of Time (WAT) Compiler. The University of Arizona.

Ankush Varma and Shuvra S. Bhattacharyya
VARMA, ª & BHATTACHARYYA, S. S. Java-through-C Compilation: An Enabling Technology for Java in Embedded Systems. In: Proceedings of the Design Automation and Test in Europe Conference and Exhibition, Designer's Forum, 2004, Paris, France. p. 161-167.

The Byte Code Engineering Library – BCEL. Disponível em: <http://jakarta.apache.org/bcel/index.html>. Acesso em: 15 de abril de 2008

LLVM Language Reference Manual. Disponível em: <http://www.llvm.org/docs/LangRef.html>. Acesso em: 15 de abril de 2008.

Getting Started with the LLVM System. Disponível em: <http://www.llvm.org/docs/GettingStarted.html>. Acesso em: 10 de junho de 2008.

Wikipédia, a enciclopédia livre. Visitor Pattern. Disponível em: <http://pt.wikipedia.org/wiki/Visitor>. Acesso em: 11 de outubro de 2008.

Wikipédia, a enciclopédia livre. Bubble Sort. Disponível em: <http://pt.wikipedia.org/wiki/Bubble_sort>. Acesso em: 19 de outubro de 2008.

Anexo I – Artigo

Tradutor Java – LLVM: Geração de código para a máquina virtual LLVM a partir de programas escritos na linguagem de programação JAVA

Eduardo Mello Cantú

Departamento de Informática e Estatística (INE)
Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil

cantu@inf.ufsc.br

Abstract. *The main point of this paper is to describe the learning process of the architecture of two separete machines: the Java Virtual Machine (JVM) and the Low Level Virtual Machine (LLVM), regarding their assembly language, and the development process of a translation tool, that translate code from JVM to LLVM.*

Resumo. *Este artigo tem como objetivo descrever o processo de aprendizado das arquiteturas das máquinas Java Virtual Machine (JVM) e Low Level Virtual Machine (LLVM), juntamente com o código de montagem das mesmas, e o processo de desenvolvimento da ferramenta de tradução de código da máquina JVM para a LLVM.*

1. Introdução

A crescente necessidade de código otimizado em diferentes áreas da computação e em particular na área de sistemas embutidos, motivou a criação e o uso da máquina virtual LLVM (Low Level Virtual Machine), para qual, até o momento, só existem front-ends C e C++ e uma versão experimental para Java. Por outro lado a plataforma Java tem se destacado na preferência de diversos segmentos de desenvolvedores, apesar de suas conhecidas limitações de eficiência. Assim sendo, a idéia deste projeto é encontrar uma maneira de permitir que desenvolvedores e pesquisadores que trabalham com a plataforma Java, possam tirar proveito dos benefícios da LLVM.

2. Low Level Virtual Machine

O LLVM é uma infra-estrutura de compilador compatível com as arquiteturas e linguagens de programação modernas, desenvolvida para alcançar três objetivos:

1. Propor uma estratégia agressiva de otimização de múltiplos estágios;

2. Ser um host (hospedeiro) de pesquisa e desenvolvimento, provendo uma fundação robusta para projetos atuais e futuros, e

3. Operar de forma transparente para o desenvolvedor comportando-se exatamente como um compilador comum.

2.1. Arquitetura do sistema LLVM

A compilação do sistema LLVM é baseada na estratégia de múltiplos passos. Esta estratégia de compilação é única no sentido de permitir uma otimização agressiva ao longo da vida da aplicação.

Comparado aos atuais sistemas de compilação, o sistema LLVM é desenvolvido para efetuar transformações mais sofisticadas no tempo de ligação (do inglês link-time), execução e após a instalação do software. Com o intuito de ser aplicável, o compilador LLVM deve se integrar com esquemas de montagem existentes, e deve ser suficientemente eficiente para ser utilizável em situações corriqueiras.

Na figura 1, vemos o funcionamento geral do sistema LLVM.

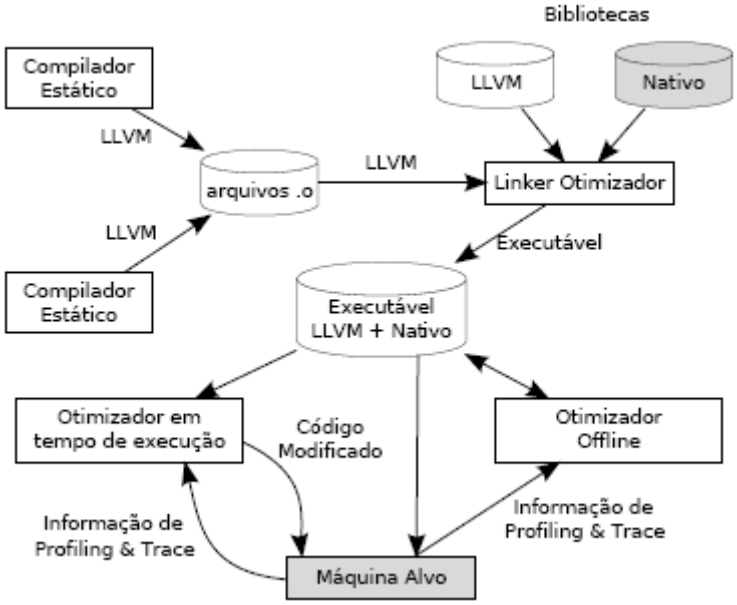

Figura 1: Arquitetura da LLVM

2.2. Conjunto de instruções virtuais do LLVM (LVIS)

Um dos fatores que diferenciam o LLVM de outros sistemas é a representação de programa que ele usa. Deve ser suficientemente *baixo nível* para permitir a otimização nas fases iniciais da compilação e suficientemente *alto nível* para suportar as otimizações agressivas feitas nos tempos de ligação e pós-ligação.

Abaixo podemos ver a lista das instruções previstas pelo LVIS.

- Instruções terminadoras (indicam qual o bloco será executado após o fim do bloco atual): *ret, br, switch, invoke, unwind, unreachable*;
- Instruções de operações binárias (são usadas para fazer a maioria das computações em um programa): *add, sub, mul, udiv, sdiv, fdiv, urem, srem, frem*;
- Instruções de operações binarias *bitwise* (usadas para executar várias formas de movimentações de bits): *shl, lshr, ashr, and, or, xor*;
- Instruções de operações vetoriais (utilizadas para acessos de elementos e outras operações específicas de vetores): *extractelement, insertelement, shufflevector*;
- Instruções de operações agregadas (servem para trabalhar com valores agregados): *extractvalue, insertvalue*;
- Instruções de acesso de memória e operações de endereçamento: *malloc, free, alloca, load,*

store, getelementptr;

- Instruções de operações de conversão (usadas para casts): *trunc .. to, zext .. to, sext .. to, fptrunc .. to, fpext .. to, fptoui .. to, fptosi .. to, uitofp .. to, sitofp .. to, ptrtoint .. to, inttoptr .. to, bitcast .. to*;
- Instruções para outros tipos de operações: *icmp, fcmp, vicmp, vfcmp, phi, select, call, va\underline{ }arg, getresult*.

3. Java

A linguagem de programação Java era chamada de Oak. Foi desenvolvida por James Gosling para ser colocada em aparelhos eletrodomésticos. Após alguns anos de experiência com a linguagem e grandes contribuições foi redirecionada para a Internet, renomeada e substancialmente revisada.

3.1. A máquina virtual Java

A máquina virtual Java é o pilar para a plataforma Java e Java 2. É o componente da tecnologia responsável pela independência do hardware e do sistema operacional, do tamanho reduzido do código compilado e da capacidade de proteger os usuários de programas maliciosos.

A máquina virtual Java é uma máquina computacional abstrata. Como uma máquina real ela possui um conjunto de instruções e manipula diversas áreas de memória durante sua execução.

A JVM não entende nada da linguagem de programação Java ela só compreende um formato binário particular, o arquivo do formato *class*. O arquivo *class* contém instruções da JVM conhecidos como bytecode e uma tabela de símbolos, assim como informações adicionais.

3.1.1. O formato do arquivo *class*

O arquivo *class* contém a definição de uma única classe ou interface e é composto por uma seqüência de bytes de 8 bits.

Também representado pela estrutura *ClassFile*, conforme descrito abaixo.

```
ClassFile
{
        u4 magic;
        u2 minor_version;
        u2 major_version;
        u2 constant_pool_count;
        cp_info constant_pool[constant_pool_count-1];
        u2 access_flags;
        u2 this_class;
        u2 super_class;
        u2 interfaces_count;
        u2 interfaces[interfaces_count];
        u2 fields_count;
        field_info fields[fields_count];
        u2 methods_count;
        method_info methods[methods_count];
        u2 attributes_count;
        attribute_info attributes[attributes_count];
}
```

Como visto, alguns dos atributos também são representados por estruturas, como é o caso da estrutura abaixo, *cp_info*.

```
cp_info
{
        u1 tag;
        u1 info[];
}
```

Na seqüência, a estrutura *method_info*, também parte de *ClassFile*, onde:

```
method_info
{
        u2 access_flags;
        u2 name_index;
        u2 descriptor_index;
        u2 attributes_count;
        attribute_info attributes[attributes_count];
}
```

E por último a estrutura *attribute_info*.

```
attribute_info
{
                u2 attribute_name_index;
                u4 attribute_length;
                u1 info[attribute_length];
}
```

4. Processo de tradução

A tradução é definida como uma atividade que abrange a interpretação do significado de um texto em uma língua (texto fonte) e a produção de um novo texto em outra língua mas que exprima o texto original da forma mais exata possível na língua destino. O texto resultante também se chama tradução.

Na computação tradutor é um programa que traduz um programa ou algo específico (textos principalmente) em outra linguagem ou em atividades.

Observamos no presente estudo que para a realização do processo de tradução seria necessário primeiramente extrair as informações provenientes do arquivo *class*.

4.1. Mapeamento do arquivo *class*

A primeira tentativa foi de desenvolver uma ferramenta experimental sendo que a estrutura formadora do arquivo *class,* com ênfase*,* havia sido estudada.

O propósito principal da criação desta ferramenta experimental foi meramente educacional, pois não se esperava que se atendesse todas as etapas previstas pela máquina virtua (processos de criação, carga e ligação).

Com este propósito buscou-se um biblioteca completa que realizasse não apenas a montagem das estruturas da arquivo *class*, mas também realizar todas as etapas necessárias. Optou-se pelo *Byte*

Code Engineering Library.

O próximo passo foi julgar as informações obtidas com o uso do BCEL, que foi de grande relevância, pois conseguiu que com a organização das informações fosse possível gerar o código alvo, ou seja, a representação LLVM.

Em seguida,criamos uma classe denominada *ClassfileVisitor (*visitante do classfile) que estende e implementa "visitantes" existentes na API tornando possível passar por todas as estruturas que compõem o arquivo de maneira simples e sem alterar em nada o fluxo.

Entretanto, isto não foi suficiente para gerar o código alvo para o LLVM. Precisávamos dar mais um passo. Pensamos no uso de um gerador de analisadores léxicos e sintáticos, mas para isso era preciso uma gramática formalizada para então termos os analisadores e a geração de código.
Infelizmente, a única informação relevante obtida através da especificação da máquina virtual do Java foi a estrutura do arquivo *classfile*, mas isto ainda não era suficiente para dar de entrada em nenhum gerador de analisadores. Foi então que avançamos e criamos nossa própria gramática contendo simplesmente as estruturas essenciais para gerar o código para a máquina alvo.

4.2. Gramática experimental do *Classfile*

Antes de criar uma gramática formal foi necessário escolher um gerador de analisadores. Para isso, nada melhor do que trabalhar com algo já compreendido e dominado. Pela familiaridade com o GALS, e pelo fato do orientando ter utilizado o mesmo durante a disciplina de introdução a compiladores resolveu-se optar por esta ferramenta.

O GALS é o responsável pela geração dos analisadores léxicos e sintáticos nos sendo incumbidos apenas responsabilidade de implementar o analisador semântico (responsável também pela geração do código na representação do LLVM).

4.3. Ferramenta final, o Tradutor

Com todas as necessárias para desempenhar o processo completo da tradução (interpretação do bytecode java e tradução do mesmo para o código da representação LLVM), bastava fazer uma ferramenta que obtivesse a integração de todas as partes envolvidas.

O resultado foi uma ferramenta muito simples que permitia a entrada do código java de duas formas: via interface ou através da abertura de um arquivo. Com a entrada definida foi possível executar o processo de tradução e na seqüência observar o código gerado para a gramática experimental do *classfile,* o código gerado para a máquina alvo na representação LLVM e o mesmo otimizado com auxílio das ferramentas disponíveis.

Na figura 2 podemos observar a interface durante a execução de uma tradução simples.

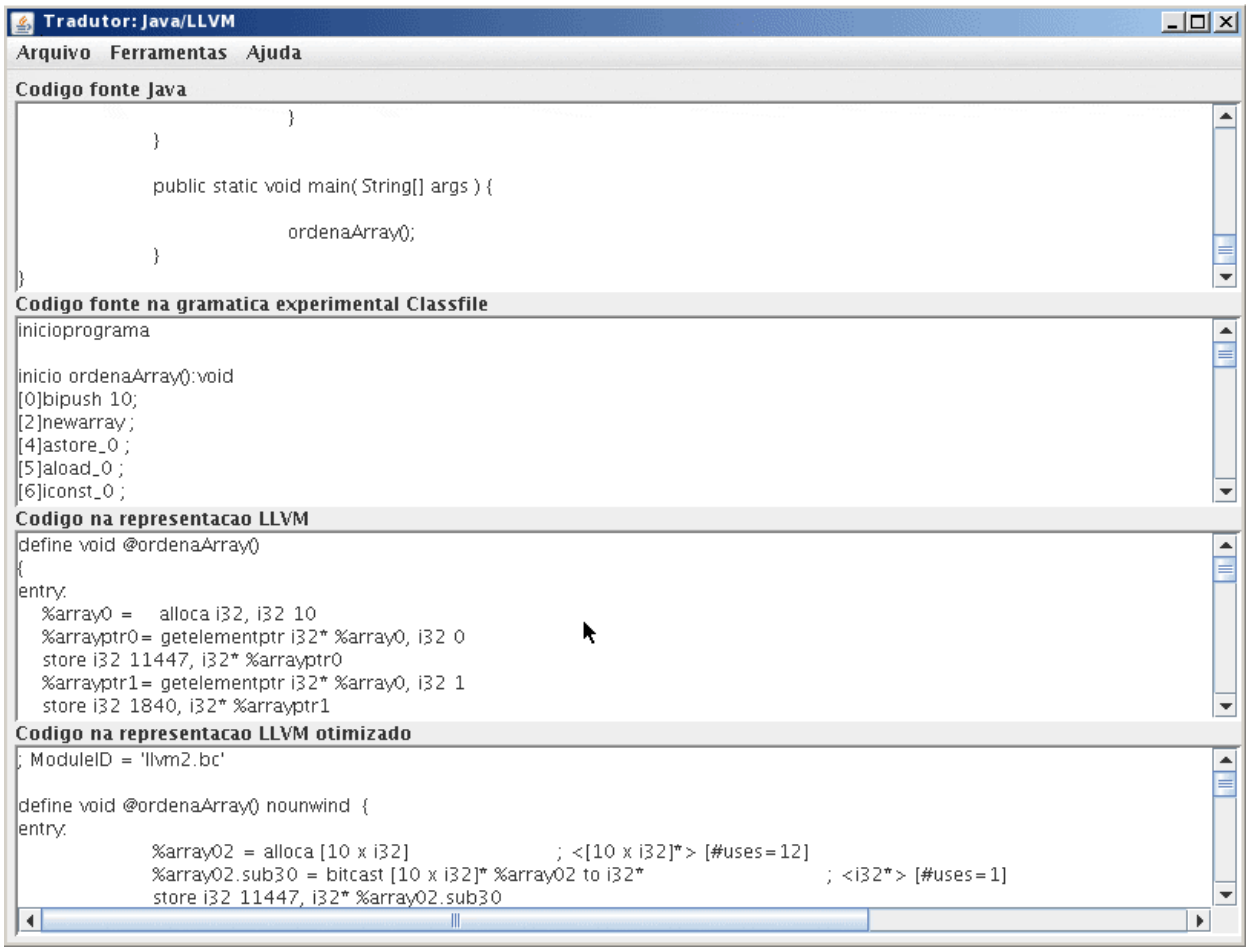

Figura 2: A interface do Tradutor

O fluxo de funcionamento do tradutor simplificando a integração entre todas as partes envolvidas pode ser observado abaixo na figura 3:

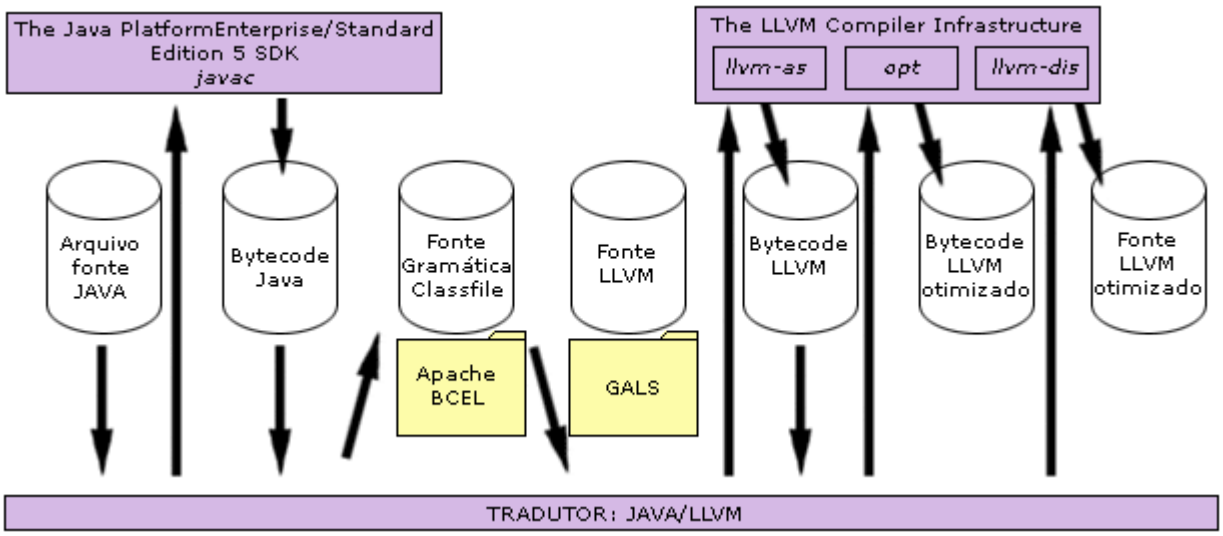

Figura 3: Fluxo de tradução

.

5. Conclusões

Para validar a eficiência do código gerado resolvemos executar um mesmo aplicativo sobre a máquina vitual Java, JVM e sobre a estrutura do LLVM. Durante a execução destes, monitoramos o uso do processador e o consumo de carga do sistema. Abaixo podemos observar os gráficos obtidos:

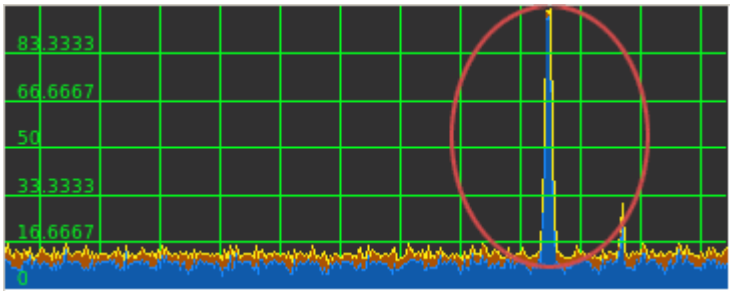

Figura 3: O uso da CPU na execução JVM

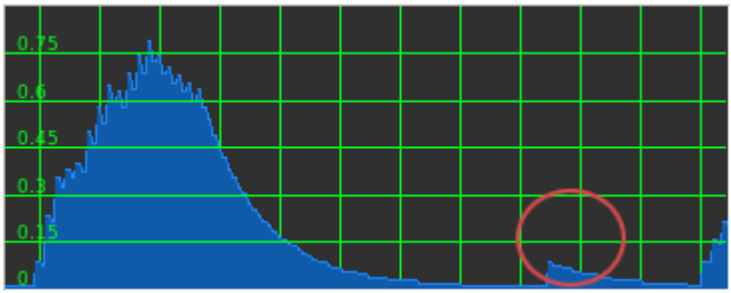

Figura 4: A carga do sistema na execução JVM

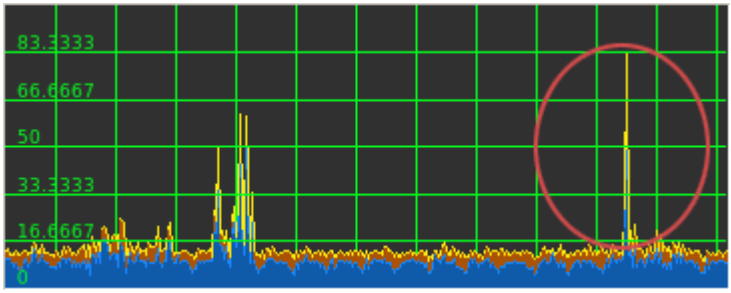

Figura 5: O uso da CPU na execução LLVM

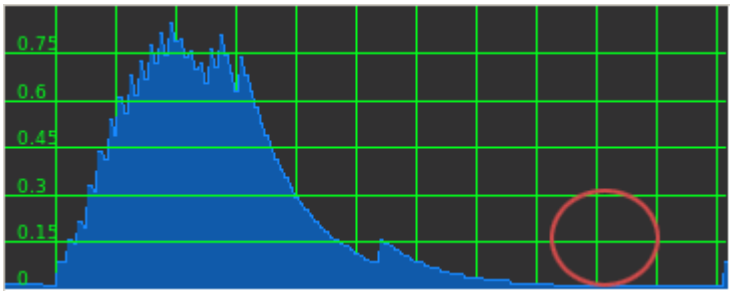

Figura 6: A carga do sistema na execução LLVM

Comparando a figura 5, que representa o uso da CPU durante a execução do programa utilizando o *just-in-time* compiler sobre o bytecode LLVM com a figura 3, que por sua vez representa o mesmo programa sendo executado sobre a máquina JVM, podemos observar uma significativa redução do uso do processador. Enquanto na execução sobre a JVM o pico máximo de uso do processador

atingiu o valor 83.33, na execução sobre a LLVM obtivemos um pico máximo de 66.66, o que representa um redução de aproximadamente 20% do uso do processador.

Com relação a carga média do sistema, se compararmos a figura 6 (execução sobre a LLVM) com a figura 4(execução sobre a JVM), observamos que a carga do sistema caiu de aproximadamente 0.7 da JVM para 0 da LLVM, o que também representa um queda significativa.

Apesar da ferramenta resultante não cobrir todas as possibilidades levantadas pela linguagem Java, consideramos com muito êxito o presente trabalho, pois nos proporcionou realizar conclusões relacionadas com a hipótese levantada inicialmente, mas principalmente por que foi possível gerar código para a máquina virtual LLVM a partir de códigos escritos na linguagem de programação Java.

Problemas foram encontrados ao longo do desenvolvimento principalmente sobre a geração do código LLVM como fonte da gramática experimental do Classfile, onde precisávamos gerar código tendo apenas como entrada seqüência de instruções. Contudo, com criatividade, paciência, disciplina e muita dedicação, finalizamos o problema utilizando pilhas e outras estruturas de dados.

Referências

Lattner, C. A. (2002) “LLVM: an infrastructure for multi-stage optimization.”. Master of Science in Computer Science - Graduate College of the University of Illinois, Urbana, Illinois.

Lindholm, T. & Yellin, F. (1999) The JavaTM Virtual Machine Specification Second Edition**.** Segunda edição. Addison-Wesley, 1999. (0-201-43294-3).

Anexo 2 – Código fonte

```
/**
 * Arquivo TelaAguarde.java
 */
package inf.ufsc.br.tcc.cantu.gui;

import javax.swing.JFrame;
import javax.swing.JLabel;

public class TelaAguarde extends JFrame {

	private static final int WIDTH = 380;
	private static final int HEIGHT = 100;
	private static final long serialVersionUID = 1L;

	private JLabel jLabelTexto;

	public TelaAguarde() {

		setLayout( null );

		int x = (int)
			((getToolkit().getScreenSize().getWidth()/2)-
				(WIDTH/2));

		int y = (int)
			((getToolkit().getScreenSize().getHeight()/2)-
				(HEIGHT/2));

		setLocation( x, y );
		setSize( WIDTH, HEIGHT );
		setResizable( false );
		setVisible( true );

		jLabelTexto = new JLabel(
			"Aguarde enquato as bibliotecas s?o carregadas." );

		jLabelTexto.setBounds( 30, 10, 350, 50 );
		add( jLabelTexto );
	}

	public void mudeTexto() {

		jLabelTexto.setText(
			"Realizando processo de tradu??o. Por favor, AGUARDE!" );
	}

	public void apareca() {

		setVisible( true );
	}

	public void desapareca() {
```

```
                setVisible( false );
        }
}


/**
 * Arquivo TelaPrincipal.java
 */
package inf.ufsc.br.tcc.cantu.gui;

import inf.ufsc.br.tcc.cantu.parser.Tradutor;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

import javax.swing.BorderFactory;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.border.BevelBorder;

public class TelaPrincipal extends JFrame implements ActionListener {

        private static final long serialVersionUID = 1L;

        private TelaAguarde telaAguarde;
        private JMenuBar menuBar;

        private JMenu menuArquivo;
        private JMenu menuFerramentas;
        private JMenu menuAjuda;

        private JMenuItem menuItemArquivoSair;
        private JMenuItem menuItemArquivoAbrir;
        private JMenuItem menuItemArquivoSalvar;
        private JMenuItem menuItemFerramentasTraduzir;
        private JMenuItem menuItemAjudaSobre;

        private JLabel labelCodigoArea;
        private JTextArea codigoTextArea;
        private JScrollPane scrollPaneCodigoTextArea;

        private JLabel labelClassfile;
        private JTextArea classfileTextArea;
        private JScrollPane scrollPaneClassfileTextArea;

        private JLabel labelLlvm;
        private JTextArea llvmTextArea;
        private JScrollPane scrollPaneLlvmTextArea;
```

```
private JLabel labelLlvmOtimizado;
private JTextArea llvmOtimizadoTextArea;
private JScrollPane scrollPaneLlvmOtimizadoTextArea;

private Tradutor tradutor;

private void criarTextAreas() {

	int altura = ( (int)this.getToolkit().getScreenSize().getHeight() / 4 ) - 40;
	int largura = ( (int)this.getToolkit().getScreenSize().getWidth() - 15 );

	int aumento = 0;

	labelCodigoArea = new JLabel( "Codigo fonte Java" );
	labelCodigoArea.setBounds( 5, 5, largura, 15 );
	add( labelCodigoArea );
	codigoTextArea = new JTextArea();
	scrollPaneCodigoTextArea = new JScrollPane( codigoTextArea );
	scrollPaneCodigoTextArea.setBounds( 5, 20, largura, altura );
	scrollPaneCodigoTextArea.setBorder(
		BorderFactory.createBevelBorder(
			BevelBorder.LOWERED ) );
	add( scrollPaneCodigoTextArea );
	aumento += 20;

	labelClassfile = new JLabel( "Codigo fonte na gramatica experimental Classfile" );
	labelClassfile.setBounds( 5, altura+aumento, largura, 15 );
	add( labelClassfile );
	classfileTextArea = new JTextArea();
	classfileTextArea.setEditable( false );
	scrollPaneClassfileTextArea = new JScrollPane( classfileTextArea );
	scrollPaneClassfileTextArea.setBounds( 5, altura+aumento+15, largura, altura );
	scrollPaneClassfileTextArea.setBorder(
		BorderFactory.createBevelBorder(
			BevelBorder.LOWERED ) );
	add( scrollPaneClassfileTextArea );
	aumento += 15;

	labelLlvm = new JLabel( "Codigo na representacao LLVM" );
	labelLlvm.setBounds( 5, (altura*2)+aumento, largura, 15 );
	add( labelLlvm );
	llvmTextArea = new JTextArea();
	llvmTextArea.setEditable( false );
	scrollPaneLlvmTextArea = new JScrollPane( llvmTextArea );
	scrollPaneLlvmTextArea.setBounds( 5, (altura*2)+aumento+15, largura, altura );
	scrollPaneLlvmTextArea.setBorder(
		BorderFactory.createBevelBorder(
			BevelBorder.LOWERED ) );
	add( scrollPaneLlvmTextArea );
	aumento += 15;

	labelLlvmOtimizado = new JLabel( "Codigo na representacao LLVM otimizado" );
	labelLlvmOtimizado.setBounds( 5, (altura*3)+aumento, largura, 15 );
	add( labelLlvmOtimizado );
	llvmOtimizadoTextArea = new JTextArea();
	llvmOtimizadoTextArea.setEditable( false );
	scrollPaneLlvmOtimizadoTextArea = new JScrollPane( llvmOtimizadoTextArea );
	scrollPaneLlvmOtimizadoTextArea.setBounds( 5, (altura*3)+aumento+15, largura, altura );
	scrollPaneLlvmOtimizadoTextArea.setBorder(
```

```
                BorderFactory.createBevelBorder(
                        BevelBorder.LOWERED ) );
        add( scrollPaneLlvmOtimizadoTextArea );
}

private void criarMenu() {

        menuItemArquivoSair = new JMenuItem( "Sair" );
        menuItemArquivoSair.addActionListener( this );
        menuItemArquivoAbrir = new JMenuItem( "Abrir" );
        menuItemArquivoAbrir.addActionListener( this );
        menuItemArquivoSalvar = new JMenuItem( "Salvar" );
        menuItemArquivoSalvar.setEnabled( false );
        menuItemArquivoSalvar.addActionListener( this );

        menuItemAjudaSobre = new JMenuItem( "Sobre" );
        menuItemAjudaSobre.addActionListener( this );

        menuItemFerramentasTraduzir = new JMenuItem( "Traduzir" );
        menuItemFerramentasTraduzir.addActionListener( this );

        menuAjuda = new JMenu( "Ajuda" );
        menuAjuda.add( menuItemAjudaSobre );

        menuArquivo = new JMenu( "Arquivo" );
        menuArquivo.add( menuItemArquivoAbrir );
        menuArquivo.add( menuItemArquivoSalvar );
        menuArquivo.addSeparator();
        menuArquivo.add( menuItemArquivoSair );

        menuFerramentas = new JMenu( "Ferramentas" );
        menuFerramentas.addSeparator();
        menuFerramentas.add( menuItemFerramentasTraduzir );

        menuBar = new JMenuBar();
        menuBar.add( menuArquivo );
        menuBar.add( menuFerramentas );
        menuBar.add( menuAjuda );

        setJMenuBar( menuBar );
}

public TelaPrincipal( Tradutor tradutor ) {

        telaAguarde = new TelaAguarde();

        try {

                Thread.sleep( 2000 );

        } catch ( InterruptedException e ) {
        }
        telaAguarde.mudeTexto();

        this.tradutor = tradutor;

        setLayout( null );

        setSize( this.getToolkit().getScreenSize() );
        setDefaultCloseOperation( EXIT_ON_CLOSE );
```

```
        setTitle( "Tradutor: Java/LLVM" );
        setResizable( true );

        criarMenu();
        criarTextAreas();

        telaAguarde.desapareca();
}

public void apareca() {

        setVisible( true );
}

private boolean checkCodigoTextArea() {

        return ( ( codigoTextArea.getText() != null )
                && ( !codigoTextArea.getText().trim().equals( "" ) ) );
}

@Override
public void actionPerformed( ActionEvent event ) {

        if ( event.getSource() == menuItemArquivoSair ) {

                System.exit( 0 );

        } else if ( event.getSource() == menuItemAjudaSobre ) {

                JOptionPane.showMessageDialog( null,
                        "Tradutor: Java/LLVM\n" +
                                "Trabalho de Conclus?o de Curso (2008-2)\n"+
                                        "Eduardo Mello Cant? (00238520)\n" +
                                                "Orientador: Olinto Jos? Varela Furtado" );

        } else if ( event.getSource() == menuItemFerramentasTraduzir ) {

                if ( checkCodigoTextArea() ) {

                        try {

                                telaAguarde.apareca();
                                tradutor.gerarBytecodeLLVM(
                                        codigoTextArea.getText() );

                                classfileTextArea.setText(
                                        tradutor.getFonteClassFile() );

                                llvmTextArea.setText(
                                        tradutor.getFonteLLVM() );

                                llvmOtimizadoTextArea.setText(
                                        tradutor.getFonteLLVMOtimizado() );

                                telaAguarde.desapareca();

                        } catch ( Exception e ) {

                                telaAguarde.desapareca();
                                //e.printStackTrace();
```

```
                                        JOptionPane.showMessageDialog( null,
                                                "Erro: "+ e.getLocalizedMessage() );
                                }
                        }

                } else if ( event.getSource() == menuItemArquivoAbrir ) {

                        JFileChooser fileChooser = new JFileChooser();
                        int action = fileChooser.showOpenDialog( null );

                        if ( action == JFileChooser.APPROVE_OPTION ) {

                                File arquivo = fileChooser.getSelectedFile();

                                if ( arquivo != null ) {

                                        try {

                                                FileReader reader = new FileReader( arquivo );
                                                BufferedReader buffer = new BufferedReader( reader );

                                                String texto = "";

                                                while ( buffer.ready() ) {

                                                        texto += buffer.readLine() +"\n";
                                                }

                                                codigoTextArea.setText( texto );

                                        } catch ( FileNotFoundException e ) {

                                                e.printStackTrace();

                                        } catch (IOException e) {

                                                e.printStackTrace();
                                        }
                                }
                        }
                }
        }

}

/**
 * Arquivo Main.java
 */
package inf.ufsc.br.tcc.cantu.main;

import inf.ufsc.br.tcc.cantu.gui.TelaPrincipal;
import inf.ufsc.br.tcc.cantu.parser.Tradutor;

public class Main {

        public static void main( String[] args ) {

                Tradutor tradutor = new Tradutor();
                TelaPrincipal tela = new TelaPrincipal( tradutor );
                tela.apareca();
```

```
		}
}

/**
 * Arquivo Tradutor.java
 */
package inf.ufsc.br.tcc.cantu.parser;

import inf.ufsc.br.tcc.cantu.parser.java2jvm.ClassfileVisitor;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Gerador;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.LexicalError;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Lexico;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.SemanticError;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Semantico;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Sintatico;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.SyntaticError;

import java.io.BufferedInputStream;
import java.io.BufferedReader;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Properties;
import java.util.StringTokenizer;

import org.apache.bcel.classfile.ClassFormatException;
import org.apache.bcel.classfile.ClassParser;
import org.apache.bcel.classfile.JavaClass;

public class Tradutor {

		private static final String ROOT_PATH =
			(String)System.getProperties().get( "user.dir" );;

		private static final String QUOTATION_LINUX = "";
		private static final String QUOTATION_WINDOWS = "\"";
		private static final String FILE_SEPARATOR_LINUX = "/";
		private static final String FILE_SEPARATOR_WINDOWS = "\\";

		private static final String JVM_COMPILER_LINUX =
			ROOT_PATH +"/javac.sh";

		private static final String JVM_COMPILER_WINDOWS =
			"\""+ ROOT_PATH +"\\javac.bat\"";

		private static final String LLVM_COMPILER_LINUX =
			ROOT_PATH +"/llvm.sh";

		private String temporaryFileName;

		private String javacDiretory;
		private StringBuilder fonteClassFile;
		private StringBuilder fonteLLVM;
		private StringBuilder fonteLLVMOtimizado;
		private JavaClass javaClass;
		private ClassfileVisitor classfileVisitor;
```

```
private Sintatico sintatico;
private Semantico semantico;
private Lexico lexico;
private Gerador gerador;

public Tradutor() {

	classfileVisitor = new ClassfileVisitor();
	fonteLLVM = new StringBuilder();
	fonteLLVMOtimizado = new StringBuilder();
	fonteClassFile = new StringBuilder();
	sintatico = new Sintatico();
	semantico = new Semantico();
	lexico = new Lexico();
	gerador = new Gerador();
}

private String getQuotation() {

	if ( isOsLinux() ) {

		return QUOTATION_LINUX;
	}

	return QUOTATION_WINDOWS;
}

private String getFileSeparator() {

	if ( isOsLinux() ) {

		return FILE_SEPARATOR_LINUX;
	}

	return FILE_SEPARATOR_WINDOWS;
}

private boolean isOsLinux() {

	if ( System.getenv().get( "TERM" ).
		toLowerCase().equals( "linux" ) ) {

		return true;
	}

	return false;
}

private String getLlvmCompiler() {

	return LLVM_COMPILER_LINUX;
}

private String getJvmCompiler() {

	if ( isOsLinux() ) {

		return JVM_COMPILER_LINUX;
	}
```

```
		return JVM_COMPILER_WINDOWS;
}

private void setTemporaryFileName( String temporaryFileName ) {

		this.temporaryFileName = temporaryFileName;
}

private String getTemporaryFileName() {

		return temporaryFileName;
}

private String getTemporaryJavafileName() {

		return ROOT_PATH+ getFileSeparator() +
			getTemporaryFileName() +".java";
}

private String getTemporaryClassfileName() {

		return ROOT_PATH+ getFileSeparator() +
			getTemporaryFileName() +".class";
}

private String getTemporaryLlvmName() {

		return ROOT_PATH+ getFileSeparator() +
			getTemporaryFileName() +".llvm";
}

private String getJavacDiretory() {

		if ( javacDiretory == null ) {

			File file = new File(
				ROOT_PATH + getFileSeparator() +
					"tradutor.properties" );

			if ( file.exists() ) {

				try {

					Properties properties = new Properties();
					properties.load(
						new FileInputStream( file ) );

					javacDiretory = (String)properties.
						get( "javac.directory" );

				} catch ( FileNotFoundException e ) {

					e.printStackTrace();

				} catch ( IOException e ) {

					e.printStackTrace();
				}
			}
		}
```

```
        return javacDiretory;
}

private void gerarFonteClassFile() {

        classfileVisitor.conhecaJavaClass(
                javaClass, fonteClassFile );

        classfileVisitor.disassemble();
}

private void gerarEstruturaClassFile() throws ClassFormatException, IOException {

        javaClass =
                new ClassParser( getTemporaryClassfileName() ).parse();
}

private void compilarJavaFile() throws Exception {

        Process process = Runtime.getRuntime().exec(
                getJvmCompiler() +" "+
                getQuotation()+getJavacDiretory()+getQuotation() +" "+
                        getQuotation()+getTemporaryJavafileName()+getQuotation()+ " "+
                                getQuotation()+ROOT_PATH+getQuotation());

        BufferedReader error = new BufferedReader(
                new InputStreamReader( process.getErrorStream() ) );

        String line = "";
        while ( ( line = error.readLine() ) != null ) {

                System.out.println( line );
        }

        process.waitFor();

        File classFile = new File( getTemporaryClassfileName() );
        if ( !classFile.exists() ) {

                throw new Exception( "Arquivo n?o foi criado, " +
                        "verifique o arquivo fonte." );
        }
}

private String getJavafileName( String fonteJava ) throws Exception {

        String begining = "public class";
        String end = " {";

        try {

                String firstChunk = fonteJava.substring(
                        fonteJava.indexOf( begining ),
                                fonteJava.indexOf( end ) );

                StringTokenizer tokenizer = new StringTokenizer( firstChunk );
                String token = null;
                while ( tokenizer.hasMoreTokens() ) {
```

```
                        token = tokenizer.nextToken();
                }

                return token;

        } catch ( Exception e ) {

                throw new Exception( "N?o foi definida uma classe." );
        }
}

private void clearFonteClassFile() {

        fonteClassFile.delete( 0,
                fonteClassFile.length() );
}

public void gerarBytecodeJVM( String fonteJava ) throws Exception {

        clearFonteClassFile();
        String filename = getJavafileName( fonteJava );

        if ( filename.equals( "" ) ) {

                throw new Exception(
                        "N?o foi poss?vel recuperar o nome do arquivo." );
        }

        setTemporaryFileName( filename );

        File javaFile = new File( getTemporaryJavafileName() );
        javaFile.createNewFile();

        /* Cria??o do arquivo. */
        if ( !javaFile.exists() ) {

                throw new Exception(
                        "N?o foi poss?vel criar o arquivo." );
        }

        FileWriter writer = new FileWriter( javaFile );
        for ( char c : fonteJava.toCharArray() ) {

                writer.append( c );
        }
        writer.close();

        /* Compilando o fonte java. */
        try {

                compilarJavaFile();

        } catch ( InterruptedException ioe ) {

                throw new Exception(
                        "N?o foi poss?vel compilar o arquivo." );

        } catch ( IOException ioe ) {

                throw new Exception(
```

```
                        "N?o foi poss?vel compilar o arquivo." );

        } catch ( Exception e ) {

                throw new Exception(
                        e.getLocalizedMessage() );
        }

        /* Gerando a estrutura do classfile. */
        try {

                gerarEstruturaClassFile();

        } catch ( IOException ioe ) {

                throw new Exception(
                        "N?o foi poss?vel gerar o fonte " +
                                "no formato bytecode java." );

        } catch ( ClassFormatException cfe ) {

                throw new Exception(
                        "N?o foi poss?vel gerar o fonte " +
                                "no formato bytecode java." );
        }

        /* Gerando o arquivo fonte classfile. */
        gerarFonteClassFile();
}

public String getFonteLLVMOtimizado() {

        return fonteLLVMOtimizado.toString();
}

public String getFonteLLVM() {

        return fonteLLVM.toString();
}

public String getFonteClassFile() {

        return fonteClassFile.toString();
}

private void deleteTempFiles() {

        File tempfile = null;

        try {

                tempfile = new File( getTemporaryJavafileName() );
                tempfile.delete();

        } catch ( Exception e ) {
        }

        try {

                tempfile = new File( getTemporaryClassfileName() );
```

```
                tempfile.delete();

        } catch ( Exception e ) {
        }

        try {

                tempfile = new File( getTemporaryLlvmName() );
                tempfile.delete();

        } catch ( Exception e ) {
        }
}

private void clearFonteLLVM() {

        fonteLLVM.delete( 0,
                        fonteLLVM.length() );
}

private void clearFonteLLVMOtimizado() {

        fonteLLVMOtimizado.delete( 0,
                        fonteLLVMOtimizado.length() );
}

private void setLLVMOtimizado() throws Exception {

        compilarLlvmFile();

        clearFonteLLVMOtimizado();

        File llvmFile = new File( getTemporaryLlvmName() );
        if ( !llvmFile.exists() ) {

                throw new Exception( "Arquivo n?o foi criado, " +
                        "verifique o arquivo fonte." );
        }

        DataInputStream dataInputStream = new DataInputStream(
                new BufferedInputStream( new FileInputStream( llvmFile ) ) );

        byte b = (byte) dataInputStream.read();

        while ( b != -1 ) {

                fonteLLVMOtimizado.append( (char)b );
                b = (byte) dataInputStream.read();
        }

        dataInputStream.close();
}

private void compilarLlvmFile() throws Exception {

        saveBytecodeLLVM();

        Process process = Runtime.getRuntime().exec(
                getLlvmCompiler() +" "+ getTemporaryLlvmName() );
```

```
        BufferedReader error = new BufferedReader(
                new InputStreamReader( process.getErrorStream() ) );

        String line = "";
        while ( ( line = error.readLine() ) != null ) {

                System.out.println( line );
        }

        process.waitFor();

}

private void saveBytecodeLLVM() throws Exception {

        String filename = getTemporaryLlvmName();

        if ( filename.equals( "" ) ) {

                throw new Exception(
                        "N?o foi poss?vel recuperar o nome do arquivo." );
        }

        File llvmFile = new File( getTemporaryLlvmName());
        llvmFile.createNewFile();

        /* Cria??o do arquivo. */
        if ( !llvmFile.exists() ) {

                throw new Exception(
                        "N?o foi poss?vel criar o arquivo." );
        }

        FileWriter writer = new FileWriter( llvmFile );
        for ( char c : fonteLLVM.toString().toCharArray() ) {

                writer.append( c );
        }
        writer.close();
}

public void gerarBytecodeLLVM( String fonteJava ) throws Exception {

        gerarBytecodeJVM( fonteJava );

        clearFonteLLVM();
        lexico.setInput( getFonteClassFile() );

        try {

                semantico.clear();
                sintatico.parse( lexico, semantico );
                gerador.setClasse( semantico.getClasse() );
                fonteLLVM.append( gerador.gerarCodigo() );

                setLLVMOtimizado();

        } catch ( LexicalError le ) {

                le.printStackTrace();
```

```
                    throw new Exception( "Erro l?xico." );

                } catch ( SyntaticError sye ) {

                    sye.printStackTrace();
                    throw new Exception( "Erro sint?tico:");

                } catch ( SemanticError see ) {

                    see.printStackTrace();
                    throw new Exception( "Erro sem?ntico." );

                } catch ( Exception e ) {

                    e.printStackTrace();
                    throw new Exception( "Erro desconhecido." );

                } finally {

                    deleteTempFiles();
                }
            }
}

/**
 * Arquivo ClassfileVisitor.java
 */
package inf.ufsc.br.tcc.cantu.parser.java2jvm;

import inf.ufsc.br.tcc.cantu.parser.utils.Util;

import org.apache.bcel.Constants;
import org.apache.bcel.classfile.Attribute;
import org.apache.bcel.classfile.Code;
import org.apache.bcel.classfile.ConstantClass;
import org.apache.bcel.classfile.ConstantDouble;
import org.apache.bcel.classfile.ConstantFieldref;
import org.apache.bcel.classfile.ConstantFloat;
import org.apache.bcel.classfile.ConstantInteger;
import org.apache.bcel.classfile.ConstantInterfaceMethodref;
import org.apache.bcel.classfile.ConstantLong;
import org.apache.bcel.classfile.ConstantMethodref;
import org.apache.bcel.classfile.ConstantNameAndType;
import org.apache.bcel.classfile.ConstantPool;
import org.apache.bcel.classfile.ConstantString;
import org.apache.bcel.classfile.ConstantUtf8;
import org.apache.bcel.classfile.ConstantValue;
import org.apache.bcel.classfile.Deprecated;
import org.apache.bcel.classfile.DescendingVisitor;
import org.apache.bcel.classfile.EmptyVisitor;
import org.apache.bcel.classfile.ExceptionTable;
import org.apache.bcel.classfile.Field;
import org.apache.bcel.classfile.JavaClass;
import org.apache.bcel.classfile.LocalVariable;
import org.apache.bcel.classfile.Method;
import org.apache.bcel.classfile.Synthetic;
import org.apache.bcel.classfile.Utility;
import org.apache.bcel.generic.*;
/**
 *
```

```
 * @author Eduado Cant?
 *
 */
public class ClassfileVisitor extends EmptyVisitor implements Visitor {

	private JavaClass		clazz;
	private Method			method;
	private StringBuilder		out;
	private String			class_name;
	private ConstantPoolGen		cp;

	public ClassfileVisitor() {
	}

	public ClassfileVisitor(JavaClass clazz, StringBuilder out ) {

		this.clazz = clazz;
		this.out = out;
		this.class_name = clazz.getClassName();
		this.cp = new ConstantPoolGen( clazz.getConstantPool() );
	}

	public void conhecaJavaClass( JavaClass clazz, StringBuilder out  ) {

		this.clazz = clazz;
		this.out = out;
		this.class_name = clazz.getClassName();
		this.cp = new ConstantPoolGen( clazz.getConstantPool() );
	}

	private boolean isMethodInit() {

		if ( this.method == null ) {

			return false;
		}

		return this.method.getName().equals( "<init>" );
	}

	public void disassemble() {

		out.append( "inicioprograma\n" );

		DescendingVisitor descendingVisitor =
			new DescendingVisitor( clazz, this );

		descendingVisitor.visit();

		out.append("\n");
		out.append( "fimprograma" );
	}

	public void visitJavaClass(JavaClass clazz) {
	}

	public void visitField( Field field ) {
	}

	public void visitExceptionTable( ExceptionTable e ) {
```

```
	}
	private final void printEndMethod( Attribute attr ) {
		Attribute[] attributes = method.getAttributes();
		if( attr == attributes[attributes.length - 1] ) {
			out.append("fim\n");
		}
		this.method = null;
	}
	public void visitDeprecated(Deprecated attribute) {
		printEndMethod( attribute );
	}
	public void visitSynthetic(Synthetic attribute) {
		printEndMethod( attribute );
	}
	public void visitMethod( Method method ) {
		this.method = method;
		if ( !isMethodInit() ) {
			out.append("\n");
			out.append("inicio "+ method.getName() + "(" );
			String attrs = "";
			if ( !method.getName().equals( "main" ) ) {
				for ( Attribute attr : method.getAttributes() ) {
					if ( Constants.ATTRIBUTE_NAMES[attr.getTag()].equals( "Code" ) ) {
						Code codeAttr = (Code)attr;
						if ( codeAttr.getLocalVariableTable() != null ) {
							if ( codeAttr.getLocalVariableTable().
								getLocalVariableTable() != null ) {
								for ( LocalVariable lv :
	codeAttr.getLocalVariableTable().getLocalVariableTable() ) {
									if ( lv.getStartPC() == 0 ) {
		attrs += Util.javaType2JvmType(lv.getSignature())
			+"  "+  lv.getName() +",";
									}
```

```
                                                                        }
                                                                }
                                                        }
                                                }
                                        }
                                }

                                if ( attrs.length() > 0 ) {

                                        attrs = attrs.substring( 0,
                                                attrs.length()-1);
                                }

                                out.append( attrs+ ")" );
                                out.append( ":"+ method.getReturnType() );

                                Attribute[] attributes = method.getAttributes();

                                if((attributes == null) || (attributes.length == 0)) {

                                        out.append("fim\n");
                                }
                        }
                }

public void visitConstantClass( ConstantClass obj ) {
}

public void visitConstantDouble( ConstantDouble obj ) {

        if ( this.method != null ) {

                out.append( obj.getBytes() );
        }
}

public void visitConstantFieldref( ConstantFieldref obj ) {
}

public void visitConstantFloat( ConstantFloat obj ) {

        if ( this.method != null ) {

                out.append( obj.getBytes() );
        }
}

public void visitConstantInteger( ConstantInteger obj ) {

        if ( this.method != null ) {

                out.append( obj.getBytes() );
        }
}

public void visitConstantInterfaceMethodref( ConstantInterfaceMethodref obj ) {
}

public void visitConstantLong( ConstantLong obj ) {
}
```

```
public void visitConstantMethodref( ConstantMethodref obj ) {

        if ( this.method != null ) {

                cp.getConstantPool().getConstant(
                        obj.getNameAndTypeIndex() ).accept( this );
        }
}

public void visitConstantNameAndType( ConstantNameAndType obj ) {

        if ( this.method != null ) {

                out.append( obj.getName( cp.getConstantPool() ) );

                out.append( Util.javaArgsToJvmArgs(
                        obj.getSignature( cp.getConstantPool() ) ));
        }
}

public void visitConstantPool( ConstantPool obj ) {

}

public void visitConstantString( ConstantString obj ) {

        if ( this.method != null ) {

                cp.getConstantPool().
                                getConstant( obj.getStringIndex() ).accept( this );
        }
}

public void visitConstantUtf8( ConstantUtf8 obj ) {

        if ( this.method != null ) {

                out.append("\"" +
                        Utility.replace(obj.getBytes(), "\n", "\\n")
                                        + "\"" );
        }
}

public void visitConstantValue( ConstantValue obj ) {
}

        public void visitCode( Code code ) {

                MethodGen          mg  = new MethodGen(method, class_name, cp);
                InstructionList    il  = mg.getInstructionList();

                if ( !isMethodInit() ) {

                        out.append("\n");

                        for ( InstructionHandle ih : il.getInstructionHandles() ) {

                                Instruction inst = ih.getInstruction();
                                out.append( "["+ ih.getPosition() +"]" );
```

```
                        out.append( inst.getName() +" " );
                        inst.accept( this );
                        out.append( ";\n" );
                    }

                    printEndMethod( code );
            }
    }

    @Override
    public void visitAALOAD(AALOAD obj) {
            // TODO Auto-generated method stub
    }

    @Override
    public void visitAASTORE(AASTORE obj) {
            // TODO Auto-generated method stub
    }

    @Override
    public void visitACONST_NULL(ACONST_NULL obj) {
            // TODO Auto-generated method stub
    }

    @Override
    public void visitALOAD(ALOAD obj) {
            // TODO Auto-generated method stub
    }

    @Override
    public void visitANEWARRAY(ANEWARRAY obj) {
            // TODO Auto-generated method stub
    }

    @Override
    public void visitARETURN(ARETURN obj) {
            // TODO Auto-generated method stub
    }

    @Override
    public void visitARRAYLENGTH(ARRAYLENGTH obj) {
    }

    @Override
    public void visitASTORE(ASTORE obj) {
            // TODO Auto-generated method stub
    }

    @Override
    public void visitATHROW(ATHROW obj) {
            // TODO Auto-generated method stub
    }

    @Override
    public void visitAllocationInstruction(AllocationInstruction obj) {
            // TODO Auto-generated method stub
    }

    @Override
    public void visitArithmeticInstruction(ArithmeticInstruction obj) {
```

```
		// TODO Auto-generated method stub

	}

	@Override
	public void visitArrayInstruction(ArrayInstruction obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitBALOAD(BALOAD obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitBASTORE(BASTORE obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitBIPUSH(BIPUSH obj) {

		out.append( obj.getValue() );
	}

	@Override
	public void visitBREAKPOINT(BREAKPOINT obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitBranchInstruction(BranchInstruction obj) {

		out.append( "("+ obj.getTarget().getPosition() +")" );
	}

	@Override
	public void visitCALOAD(CALOAD obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitCASTORE(CASTORE obj) {
		// TODO Auto-generated method stub
	}

	@Override
	public void visitCHECKCAST(CHECKCAST obj) {
		// TODO Auto-generated method stub
	}

	@Override
	public void visitCPInstruction(CPInstruction inst) {

		cp.getConstantPool().getConstant(
```

```
                    inst.getIndex() ).accept( this );
}

@Override
public void visitConstantPushInstruction(ConstantPushInstruction obj) {
        // TODO Auto-generated method stub
}

@Override
public void visitConversionInstruction(ConversionInstruction obj) {
        // TODO Auto-generated method stub
}

@Override
public void visitD2F(D2F obj) {
        // TODO Auto-generated method stub
}

@Override
public void visitD2I(D2I obj) {
        // TODO Auto-generated method stub
}

@Override
public void visitD2L(D2L obj) {
        // TODO Auto-generated method stub

}

@Override
public void visitDADD(DADD obj) {
        // TODO Auto-generated method stub

}

@Override
public void visitDALOAD(DALOAD obj) {
        // TODO Auto-generated method stub

}

@Override
public void visitDASTORE(DASTORE obj) {
        // TODO Auto-generated method stub
}

@Override
public void visitDCMPG(DCMPG obj) {
        // TODO Auto-generated method stub
}

@Override
public void visitDCMPL(DCMPL obj) {
        // TODO Auto-generated method stub
}

@Override
public void visitDCONST(DCONST obj) {
        // TODO Auto-generated method stub
```

```
}

@Override
public void visitDDIV(DDIV obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDLOAD(DLOAD obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDMUL(DMUL obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDNEG(DNEG obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDREM(DREM obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDRETURN(DRETURN obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDSTORE(DSTORE obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDSUB(DSUB obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDUP(DUP obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDUP2(DUP2 obj) {
	// TODO Auto-generated method stub
```

```
}

@Override
public void visitDUP2_X1(DUP2_X1 obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDUP2_X2(DUP2_X2 obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDUP_X1(DUP_X1 obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitDUP_X2(DUP_X2 obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitExceptionThrower(ExceptionThrower obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitF2D(F2D obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitF2I(F2I obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitF2L(F2L obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFADD(FADD obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFALOAD(FALOAD obj) {
	// TODO Auto-generated method stub
```

```
}

@Override
public void visitFASTORE(FASTORE obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFCMPG(FCMPG obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFCMPL(FCMPL obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFCONST(FCONST obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFDIV(FDIV obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFLOAD(FLOAD obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFMUL(FMUL obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFNEG(FNEG obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFREM(FREM obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFRETURN(FRETURN obj) {
	// TODO Auto-generated method stub
```

```
}

@Override
public void visitFSTORE(FSTORE obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFSUB(FSUB obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFieldInstruction(FieldInstruction obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitFieldOrMethod(FieldOrMethod obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitGETFIELD(GETFIELD obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitGETSTATIC(GETSTATIC obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitGOTO(GOTO obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitGOTO_W(GOTO_W obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitGotoInstruction(GotoInstruction obj) {
}

@Override
public void visitI2B(I2B obj) {
}

@Override
public void visitI2C(I2C obj) {
```

```
		// TODO Auto-generated method stub

	}

	@Override
	public void visitI2D(I2D obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitI2F(I2F obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitI2L(I2L obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitI2S(I2S obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIADD(IADD obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIALOAD(IALOAD obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIAND(IAND obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIASTORE(IASTORE obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitICONST(ICONST obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIDIV(IDIV obj) {
```

```
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIFEQ(IFEQ obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIFGE(IFGE obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIFGT(IFGT obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIFLE(IFLE obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIFLT(IFLT obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIFNE(IFNE obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIFNONNULL(IFNONNULL obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIFNULL(IFNULL obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIF_ACMPEQ(IF_ACMPEQ obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIF_ACMPNE(IF_ACMPNE obj) {
```

```
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIF_ICMPEQ(IF_ICMPEQ obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIF_ICMPGE(IF_ICMPGE obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIF_ICMPGT(IF_ICMPGT obj) {
		// TODO Auto-generated method stub
	}

	@Override
	public void visitIF_ICMPLE(IF_ICMPLE obj) {
	}

	@Override
	public void visitIF_ICMPLT(IF_ICMPLT obj) {
		// TODO Auto-generated method stub
	}

	@Override
	public void visitIF_ICMPNE(IF_ICMPNE obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIINC(IINC obj) {

		out.append( "[" );
		out.append( obj.getIndex()  );
		out.append( ";" );
		out.append( obj.getIncrement() );
		out.append( "]" );
	}

	@Override
	public void visitILOAD(ILOAD obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIMPDEP1(IMPDEP1 obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIMPDEP2(IMPDEP2 obj) {
```

```
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIMUL(IMUL obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitINEG(INEG obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitINSTANCEOF(INSTANCEOF obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitINVOKEINTERFACE(INVOKEINTERFACE obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitINVOKESPECIAL(INVOKESPECIAL obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitINVOKESTATIC(INVOKESTATIC obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitINVOKEVIRTUAL(INVOKEVIRTUAL obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIOR(IOR obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIREM(IREM obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIRETURN(IRETURN obj) {
```

```
		// TODO Auto-generated method stub

	}

	@Override
	public void visitISHL(ISHL obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitISHR(ISHR obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitISTORE(ISTORE obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitISUB(ISUB obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIUSHR(IUSHR obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIXOR(IXOR obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitIfInstruction(IfInstruction obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitInvokeInstruction(InvokeInstruction obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitJSR(JSR obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitJSR_W(JSR_W obj) {
```

```
		// TODO Auto-generated method stub

	}

	@Override
	public void visitJsrInstruction(JsrInstruction obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitL2D(L2D obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitL2F(L2F obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitL2I(L2I obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLADD(LADD obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLALOAD(LALOAD obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLAND(LAND obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLASTORE(LASTORE obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLCMP(LCMP obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLCONST(LCONST obj) {
```

```
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLDC(LDC obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLDC2_W(LDC2_W obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLDIV(LDIV obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLLOAD(LLOAD obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLMUL(LMUL obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLNEG(LNEG obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLOOKUPSWITCH(LOOKUPSWITCH obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLOR(LOR obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLREM(LREM obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLRETURN(LRETURN obj) {
```

```
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLSHL(LSHL obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLSHR(LSHR obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLSTORE(LSTORE obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLSUB(LSUB obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLUSHR(LUSHR obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLXOR(LXOR obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLoadClass(LoadClass obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLoadInstruction(LoadInstruction obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitLocalVariableInstruction(LocalVariableInstruction obj) {
	}

	@Override
	public void visitMONITORENTER(MONITORENTER obj) {
		// TODO Auto-generated method stub
```

```
}

@Override
public void visitMONITOREXIT(MONITOREXIT obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitMULTIANEWARRAY(MULTIANEWARRAY obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitNEW(NEW obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitNEWARRAY(NEWARRAY obj) {
}

@Override
public void visitNOP(NOP obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitPOP(POP obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitPOP2(POP2 obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitPUTFIELD(PUTFIELD obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitPUTSTATIC(PUTSTATIC obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitPopInstruction(PopInstruction obj) {
	// TODO Auto-generated method stub

}
```

```
@Override
public void visitPushInstruction(PushInstruction obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitRET(RET obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitRETURN(RETURN obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitReturnInstruction(ReturnInstruction obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitSALOAD(SALOAD obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitSASTORE(SASTORE obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitSIPUSH(SIPUSH obj) {

	out.append( obj.getValue() );
}

@Override
public void visitSWAP(SWAP obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitSelect(Select obj) {
	// TODO Auto-generated method stub

}

@Override
public void visitStackConsumer(StackConsumer obj) {
	// TODO Auto-generated method stub

}
```

```
	@Override
	public void visitStackInstruction(StackInstruction obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitStackProducer(StackProducer obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitStoreInstruction(StoreInstruction obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitTABLESWITCH(TABLESWITCH obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitTypedInstruction(TypedInstruction obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitUnconditionalBranch(UnconditionalBranch obj) {
		// TODO Auto-generated method stub

	}

	@Override
	public void visitVariableLengthInstruction(VariableLengthInstruction obj) {
		// TODO Auto-generated method stub
	}
}


/**
 * Arquivo AnalysisError.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

public class AnalysisError extends Exception
{
	private static final long serialVersionUID = 1L;

	private int position;

   public AnalysisError(String msg, int position)
   {
      super(msg);
      this.position = position;
   }
```

```
    public AnalysisError(String msg)
    {
        super(msg);
        this.position = -1;
    }

    public int getPosition()
    {
        return position;
    }

    public String toString()
    {
        return super.toString() + ", @ "+position;
    }
}


/**
 * Arquivo Constants.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

public interface Constants extends ScannerConstants, ParserConstants
{
    int EPSILON  = 0;
    int DOLLAR   = 1;

    int t_identificador = 2;
    int t_literal = 3;
    int t_inteiro = 4;
    int t_real = 5;
    int t_inicio = 6;
    int t_fim = 7;
    int t_inicioprograma = 8;
    int t_fimprograma = 9;
    int t_int = 10;
    int t_double = 11;
    int t_float = 12;
    int t_bool = 13;
    int t_string = 14;
    int t_void = 15;
    int t_nop = 16;
    int t_aconst_null = 17;
    int t_iconst_m1 = 18;
    int t_iconst_0 = 19;
    int t_iconst_1 = 20;
    int t_iconst_2 = 21;
    int t_iconst_3 = 22;
    int t_iconst_4 = 23;
    int t_iconst_5 = 24;
    int t_lconst_0 = 25;
    int t_lconst_1 = 26;
    int t_fconst_0 = 27;
    int t_fconst_1 = 28;
    int t_fconst_2 = 29;
    int t_dconst_0 = 30;
    int t_dconst_1 = 31;
    int t_bipush = 32;
    int t_sipush = 33;
```

```
int t_ldc = 34;
int t_ldc_w = 35;
int t_ldc2_w = 36;
int t_iload = 37;
int t_lload = 38;
int t_fload = 39;
int t_dload = 40;
int t_aload = 41;
int t_iload_0 = 42;
int t_iload_1 = 43;
int t_iload_2 = 44;
int t_iload_3 = 45;
int t_lload_0 = 46;
int t_lload_1 = 47;
int t_lload_2 = 48;
int t_lload_3 = 49;
int t_fload_0 = 50;
int t_fload_1 = 51;
int t_fload_2 = 52;
int t_fload_3 = 53;
int t_dload_0 = 54;
int t_dload_1 = 55;
int t_dload_2 = 56;
int t_dload_3 = 57;
int t_aload_0 = 58;
int t_aload_1 = 59;
int t_aload_2 = 60;
int t_aload_3 = 61;
int t_iaload = 62;
int t_laload = 63;
int t_faload = 64;
int t_daload = 65;
int t_aaload = 66;
int t_baload = 67;
int t_caload = 68;
int t_saload = 69;
int t_istore = 70;
int t_lstore = 71;
int t_fstore = 72;
int t_dstore = 73;
int t_astore = 74;
int t_istore_0 = 75;
int t_istore_1 = 76;
int t_istore_2 = 77;
int t_istore_3 = 78;
int t_lstore_0 = 79;
int t_lstore_1 = 80;
int t_lstore_2 = 81;
int t_lstore_3 = 82;
int t_fstore_0 = 83;
int t_fstore_1 = 84;
int t_fstore_2 = 85;
int t_fstore_3 = 86;
int t_dstore_0 = 87;
int t_dstore_1 = 88;
int t_dstore_2 = 89;
int t_dstore_3 = 90;
int t_astore_0 = 91;
int t_astore_1 = 92;
int t_astore_2 = 93;
```

```
int t_astore_3 = 94;
int t_iastore = 95;
int t_lastore = 96;
int t_fastore = 97;
int t_dastore = 98;
int t_aastore = 99;
int t_bastore = 100;
int t_castore = 101;
int t_sastore = 102;
int t_pop = 103;
int t_pop2 = 104;
int t_dup = 105;
int t_dup_x1 = 106;
int t_dup_x2 = 107;
int t_dup2 = 108;
int t_dup2_x1 = 109;
int t_dup2_x2 = 110;
int t_swap = 111;
int t_iadd = 112;
int t_ladd = 113;
int t_fadd = 114;
int t_dadd = 115;
int t_isub = 116;
int t_lsub = 117;
int t_fsub = 118;
int t_dsub = 119;
int t_imul = 120;
int t_lmul = 121;
int t_fmul = 122;
int t_dmul = 123;
int t_idiv = 124;
int t_ldiv = 125;
int t_fdiv = 126;
int t_ddiv = 127;
int t_irem = 128;
int t_lrem = 129;
int t_frem = 130;
int t_drem = 131;
int t_ineg = 132;
int t_lneg = 133;
int t_fneg = 134;
int t_dneg = 135;
int t_ishl = 136;
int t_lshl = 137;
int t_ishr = 138;
int t_lshr = 139;
int t_iushr = 140;
int t_lushr = 141;
int t_iand = 142;
int t_land = 143;
int t_ior = 144;
int t_lor = 145;
int t_ixor = 146;
int t_lxor = 147;
int t_iinc = 148;
int t_i2l = 149;
int t_i2f = 150;
int t_i2d = 151;
int t_l2i = 152;
int t_l2f = 153;
```

```
int t_l2d = 154;
int t_f2i = 155;
int t_f2l = 156;
int t_f2d = 157;
int t_d2i = 158;
int t_d2l = 159;
int t_d2f = 160;
int t_i2b = 161;
int t_i2c = 162;
int t_i2s = 163;
int t_lcmp = 164;
int t_fcmpl = 165;
int t_fcmpg = 166;
int t_dcmpl = 167;
int t_dcmpg = 168;
int t_ifeq = 169;
int t_ifne = 170;
int t_iflt = 171;
int t_ifge = 172;
int t_ifgt = 173;
int t_ifle = 174;
int t_if_icmpeq = 175;
int t_if_icmpne = 176;
int t_if_icmplt = 177;
int t_if_icmpge = 178;
int t_if_icmpgt = 179;
int t_if_icmple = 180;
int t_if_acmpeq = 181;
int t_if_acmpne = 182;
int t_goto = 183;
int t_jsr = 184;
int t_ret = 185;
int t_tableswitch = 186;
int t_lookupswitch = 187;
int t_ireturn = 188;
int t_lreturn = 189;
int t_freturn = 190;
int t_dreturn = 191;
int t_areturn = 192;
int t_return = 193;
int t_getstatic = 194;
int t_putstatic = 195;
int t_getfield = 196;
int t_putfield = 197;
int t_invokevirtual = 198;
int t_invokespecial = 199;
int t_invokestatic = 200;
int t_invokeinterface = 201;
int t_new = 202;
int t_newarray = 203;
int t_anewarray = 204;
int t_arraylength = 205;
int t_athrow = 206;
int t_checkcast = 207;
int t_instanceof = 208;
int t_monitorenter = 209;
int t_monitorexit = 210;
int t_wide = 211;
int t_multianewarray = 212;
int t_ifnull = 213;
```

```
    int t_ifnonnull = 214;
    int t_goto_w = 215;
    int t_jsr_w = 216;
    int t_breakpoint = 217;
    int t_impdep1 = 218;
    int t_impdep2 = 219;
    int t_TOKEN_220 = 220; //"\n"
    int t_TOKEN_221 = 221; //":"
    int t_TOKEN_222 = 222; //";"
    int t_TOKEN_223 = 223; //"("
    int t_TOKEN_224 = 224; //")"
    int t_TOKEN_225 = 225; //","
    int t_TOKEN_226 = 226; //"["
    int t_TOKEN_227 = 227; //"]"

}


/**
 * Arquivo Gerador.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator.Class;

public class Gerador {

        private Class classe;

        public Gerador() {
        }

        public void setClasse( Class classe ) {

                this.classe = classe;
        }

        public Class getClasse() {

                return classe;
        }

        public String gerarCodigo() {

                return getClasse().toString();
        }

}


/**
 * Arquivo LexicalError.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

public class LexicalError extends AnalysisError
{
        private static final long serialVersionUID = 1L;

        public LexicalError(String msg, int position)
```

```
            {
        super(msg, position);
    }

    public LexicalError(String msg)
    {
        super(msg);
    }
}


/**
 * Arquivo Lexico.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

public class Lexico implements Constants
{
    private int position;
    private String input;

    public Lexico()
    {
        this("");
    }

    public Lexico(String input)
    {
        setInput(input);
    }

    public void setInput(String input)
    {
        this.input = input;
        setPosition(0);
    }

    public void setPosition(int pos)
    {
        position = pos;
    }

    public Token nextToken() throws LexicalError
    {
        if ( ! hasInput() )
            return null;

        int start = position;

        int state = 0;
        int lastState = 0;
        int endState = -1;
        int end = -1;

        while (hasInput())
        {
            lastState = state;
            state = nextState(nextChar(), state);

            if (state < 0)
```

```
            break;

        else
        {
            if (tokenForState(state) >= 0)
            {
                endState = state;
                end = position;
            }
        }
    }
    if (endState < 0 || (endState != state && tokenForState(lastState) == -2))
        throw new LexicalError(SCANNER_ERROR[lastState], start);

    position = end;

    int token = tokenForState(endState);

    if (token == 0)
        return nextToken();
    else
    {
        String lexeme = input.substring(start, end);
        token = lookupToken(token, lexeme);
        return new Token(token, lexeme, start);
    }
}

private int nextState(char c, int state)
{
    int start = SCANNER_TABLE_INDEXES[state];
    int end   = SCANNER_TABLE_INDEXES[state+1]-1;

    while (start <= end)
    {
        int half = (start+end)/2;

        if (SCANNER_TABLE[half][0] == c)
            return SCANNER_TABLE[half][1];
        else if (SCANNER_TABLE[half][0] < c)
            start = half+1;
        else  //(SCANNER_TABLE[half][0] > c)
            end = half-1;
    }

    return -1;
}

private int tokenForState(int state)
{
    if (state < 0 || state >= TOKEN_STATE.length)
        return -1;

    return TOKEN_STATE[state];
}

public int lookupToken(int base, String key)
{
    int start = SPECIAL_CASES_INDEXES[base];
    int end   = SPECIAL_CASES_INDEXES[base+1]-1;
```

```
      while (start <= end)
      {
         int half = (start+end)/2;
         int comp = SPECIAL_CASES_KEYS[half].compareTo(key);

         if (comp == 0)
            return SPECIAL_CASES_VALUES[half];
         else if (comp < 0)
            start = half+1;
         else  //(comp > 0)
            end = half-1;
      }

      return base;
   }

   private boolean hasInput()
   {
      return position < input.length();
   }

   private char nextChar()
   {
      if (hasInput())
         return input.charAt(position++);
      else
         return (char) -1;
   }
}


/**
 * Arquivo ParserConstants.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

public interface ParserConstants
{
   int START_SYMBOL = 228;

   int FIRST_NON_TERMINAL    = 228;
   int FIRST_SEMANTIC_ACTION = 249;

   int[][] PARSER_TABLE =
   {
      { -1, -1, -1, -1, -1, -1, -1,  0, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
      { -1, -1, -1, -1, -1,  1, -1, -1,  1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
```

```
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
    { -1, -1, -1, -1, -1,  2, -1, -1,  3, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
    { -1, -1, -1, -1, -1,  4, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
    { -1,  5, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
    { -1, -1, -1, -1, -1, -1, -1, -1, -1, 13, 13, 13, 13, 13, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 14, -1, -1, -1 },
    { -1, -1, -1, -1, -1, -1, -1, -1, -1, 33, 33, 33, 33, 33, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 34, -1, -1, -1 },
    { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 36, 35, -1, -1 },
    { -1, -1, -1, -1, -1, -1, -1, -1, -1,  8,  9, 12, 10, 11, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
    { -1, -1, -1, -1, -1, -1, -1, -1, -1,  6,  6,  6,  6,  6,  7, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
```

-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 16, 15, -1, -1 },
{ -1, -1, -1, -1, -1, -1, 18, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 17, -1 },
{ -1, 32, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 20, -1 },
{ -1, 23, 22, 22, 22, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 26, 24, -1, -1, 25, -1 },
{ -1, -1, 30, 29, 31, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1 },
{ -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,

```
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 19, -1 },
      { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48,
49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75,
76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102,
103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126,
127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150,
151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174,
175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198,
199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222,
223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, -1, -1, -1, -1, -1, -1, -1, -1
},
      { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 21, -1 },
      { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 28, -1, -1, -1, -1 },
      { -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1,
-1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, -1, 27, -1 }
  };

  int[][] PRODUCTIONS =
  {
    {   8, 250, 229,   9, 251 },
    { 230 },
    { 231, 230 },
    {   0 },
    {   6, 252, 232, 239,   7, 253 },
    {   2, 254, 223, 233, 224, 221, 237, 272 },
    { 271, 236 },
    { 271,  15, 255 },
    {  10, 256 },
    {  11, 257 },
    {  13, 258 },
    {  14, 259 },
    {  12, 260 },
    { 270, 236,   2, 261, 238 },
    {   0 },
    { 225, 270, 236,   2, 261, 238 },
    {   0 },
    { 244, 239 },
    {   0 },
    { 241, 222 },
```

```
{ 246, 245, 242 },
{ 226,   4, 273, 227 },
{ 262, 243 },
{ 263, 240 },
{ 274, 247 },
{ 248 },
{   0 },
{ 226,   4, 276, 222,   4, 277, 227 },
{ 223,   4, 275, 224 },
{   4, 264 },
{   3, 265 },
{   5, 266 },
{   2, 267, 223, 234, 224 },
{ 236, 268, 235, 269 },
{   0 },
{ 225, 236, 268, 235, 269 },
{   0 },
{  16, 349 },
{  17, 350 },
{  18, 351 },
{  19, 352 },
{  20, 353 },
{  21, 354 },
{  22, 355 },
{  23, 356 },
{  24, 357 },
{  25, 358 },
{  26, 359 },
{  27, 360 },
{  28, 361 },
{  29, 362 },
{  30, 363 },
{  31, 364 },
{  32, 365 },
{  33, 366 },
{  34, 367 },
{  35, 368 },
{  36, 369 },
{  37, 370 },
{  38, 371 },
{  39, 372 },
{  40, 373 },
{  41, 374 },
{  42, 375 },
{  43, 376 },
{  44, 377 },
{  45, 378 },
{  46, 379 },
{  47, 380 },
{  48, 381 },
{  49, 382 },
{  50, 383 },
{  51, 384 },
{  52, 385 },
{  53, 386 },
{  54, 387 },
{  55, 388 },
{  56, 389 },
{  57, 390 },
{  58, 391 },
```

{ 59, 392 },
{ 60, 393 },
{ 61, 394 },
{ 62, 395 },
{ 63, 396 },
{ 64, 397 },
{ 65, 398 },
{ 66, 399 },
{ 67, 400 },
{ 68, 401 },
{ 69, 402 },
{ 70, 403 },
{ 71, 404 },
{ 72, 405 },
{ 73, 406 },
{ 74, 407 },
{ 75, 408 },
{ 76, 409 },
{ 77, 410 },
{ 78, 411 },
{ 79, 412 },
{ 80, 413 },
{ 81, 414 },
{ 82, 415 },
{ 83, 416 },
{ 84, 417 },
{ 85, 418 },
{ 86, 419 },
{ 87, 420 },
{ 88, 421 },
{ 89, 422 },
{ 90, 423 },
{ 91, 424 },
{ 92, 425 },
{ 93, 426 },
{ 94, 427 },
{ 95, 428 },
{ 96, 429 },
{ 97, 430 },
{ 98, 431 },
{ 99, 432 },
{ 100, 433 },
{ 101, 434 },
{ 102, 435 },
{ 103, 436 },
{ 104, 437 },
{ 105, 438 },
{ 106, 439 },
{ 107, 440 },
{ 108, 441 },
{ 109, 442 },
{ 110, 443 },
{ 111, 444 },
{ 112, 445 },
{ 113, 446 },
{ 114, 447 },
{ 115, 448 },
{ 116, 449 },
{ 117, 450 },
{ 118, 451 },

{ 119, 452 },
{ 120, 453 },
{ 121, 454 },
{ 122, 455 },
{ 123, 456 },
{ 124, 457 },
{ 125, 458 },
{ 126, 459 },
{ 127, 460 },
{ 128, 461 },
{ 129, 462 },
{ 130, 463 },
{ 131, 464 },
{ 132, 465 },
{ 133, 466 },
{ 134, 467 },
{ 135, 468 },
{ 136, 469 },
{ 137, 470 },
{ 138, 471 },
{ 139, 472 },
{ 140, 473 },
{ 141, 474 },
{ 142, 475 },
{ 143, 476 },
{ 144, 477 },
{ 145, 478 },
{ 146, 479 },
{ 147, 480 },
{ 148, 481 },
{ 149, 482 },
{ 150, 483 },
{ 151, 484 },
{ 152, 485 },
{ 153, 486 },
{ 154, 487 },
{ 155, 488 },
{ 156, 489 },
{ 157, 490 },
{ 158, 491 },
{ 159, 492 },
{ 160, 493 },
{ 161, 494 },
{ 162, 495 },
{ 163, 496 },
{ 164, 497 },
{ 165, 498 },
{ 166, 499 },
{ 167, 500 },
{ 168, 501 },
{ 169, 502 },
{ 170, 503 },
{ 171, 504 },
{ 172, 505 },
{ 173, 506 },
{ 174, 507 },
{ 175, 508 },
{ 176, 509 },
{ 177, 510 },
{ 178, 511 },

```
    { 179, 512 },
    { 180, 513 },
    { 181, 514 },
    { 182, 515 },
    { 183, 516 },
    { 184, 517 },
    { 185, 518 },
    { 186, 519 },
    { 187, 520 },
    { 188, 521 },
    { 189, 522 },
    { 190, 523 },
    { 191, 524 },
    { 192, 525 },
    { 193, 526 },
    { 194, 527 },
    { 195, 528 },
    { 196, 529 },
    { 197, 530 },
    { 198, 531 },
    { 199, 532 },
    { 200, 533 },
    { 201, 534 },
    { 202, 535 },
    { 203, 536 },
    { 204, 537 },
    { 205, 538 },
    { 206, 539 },
    { 207, 540 },
    { 208, 541 },
    { 209, 542 },
    { 210, 543 },
    { 211, 544 },
    { 212, 545 },
    { 213, 546 },
    { 214, 547 },
    { 215, 548 },
    { 216, 549 },
    { 217, 550 },
    { 218, 551 },
    { 219, 552 }
};

String[] PARSER_ERROR =
{
    "",
    "Era esperado fim de programa",
    "Era esperado identificador",
    "Era esperado literal",
    "Era esperado inteiro",
    "Era esperado real",
    "Era esperado inicio",
    "Era esperado fim",
    "Era esperado inicioprograma",
    "Era esperado fimprograma",
    "Era esperado int",
    "Era esperado double",
    "Era esperado float",
    "Era esperado bool",
    "Era esperado string",
```

"Era esperado void",
"Era esperado nop",
"Era esperado aconst_null",
"Era esperado iconst_m1",
"Era esperado iconst_0",
"Era esperado iconst_1",
"Era esperado iconst_2",
"Era esperado iconst_3",
"Era esperado iconst_4",
"Era esperado iconst_5",
"Era esperado lconst_0",
"Era esperado lconst_1",
"Era esperado fconst_0",
"Era esperado fconst_1",
"Era esperado fconst_2",
"Era esperado dconst_0",
"Era esperado dconst_1",
"Era esperado bipush",
"Era esperado sipush",
"Era esperado ldc",
"Era esperado ldc_w",
"Era esperado ldc2_w",
"Era esperado iload",
"Era esperado lload",
"Era esperado fload",
"Era esperado dload",
"Era esperado aload",
"Era esperado iload_0",
"Era esperado iload_1",
"Era esperado iload_2",
"Era esperado iload_3",
"Era esperado lload_0",
"Era esperado lload_1",
"Era esperado lload_2",
"Era esperado lload_3",
"Era esperado fload_0",
"Era esperado fload_1",
"Era esperado fload_2",
"Era esperado fload_3",
"Era esperado dload_0",
"Era esperado dload_1",
"Era esperado dload_2",
"Era esperado dload_3",
"Era esperado aload_0",
"Era esperado aload_1",
"Era esperado aload_2",
"Era esperado aload_3",
"Era esperado iaload",
"Era esperado laload",
"Era esperado faload",
"Era esperado daload",
"Era esperado aaload",
"Era esperado baload",
"Era esperado caload",
"Era esperado saload",
"Era esperado istore",
"Era esperado lstore",
"Era esperado fstore",
"Era esperado dstore",
"Era esperado astore",

"Era esperado istore_0",
"Era esperado istore_1",
"Era esperado istore_2",
"Era esperado istore_3",
"Era esperado lstore_0",
"Era esperado lstore_1",
"Era esperado lstore_2",
"Era esperado lstore_3",
"Era esperado fstore_0",
"Era esperado fstore_1",
"Era esperado fstore_2",
"Era esperado fstore_3",
"Era esperado dstore_0",
"Era esperado dstore_1",
"Era esperado dstore_2",
"Era esperado dstore_3",
"Era esperado astore_0",
"Era esperado astore_1",
"Era esperado astore_2",
"Era esperado astore_3",
"Era esperado iastore",
"Era esperado lastore",
"Era esperado fastore",
"Era esperado dastore",
"Era esperado aastore",
"Era esperado bastore",
"Era esperado castore",
"Era esperado sastore",
"Era esperado pop",
"Era esperado pop2",
"Era esperado dup",
"Era esperado dup_x1",
"Era esperado dup_x2",
"Era esperado dup2",
"Era esperado dup2_x1",
"Era esperado dup2_x2",
"Era esperado swap",
"Era esperado iadd",
"Era esperado ladd",
"Era esperado fadd",
"Era esperado dadd",
"Era esperado isub",
"Era esperado lsub",
"Era esperado fsub",
"Era esperado dsub",
"Era esperado imul",
"Era esperado lmul",
"Era esperado fmul",
"Era esperado dmul",
"Era esperado idiv",
"Era esperado ldiv",
"Era esperado fdiv",
"Era esperado ddiv",
"Era esperado irem",
"Era esperado lrem",
"Era esperado frem",
"Era esperado drem",
"Era esperado ineg",
"Era esperado lneg",
"Era esperado fneg",

"Era esperado dneg",
"Era esperado ishl",
"Era esperado lshl",
"Era esperado ishr",
"Era esperado lshr",
"Era esperado iushr",
"Era esperado lushr",
"Era esperado iand",
"Era esperado land",
"Era esperado ior",
"Era esperado lor",
"Era esperado ixor",
"Era esperado lxor",
"Era esperado iinc",
"Era esperado i2l",
"Era esperado i2f",
"Era esperado i2d",
"Era esperado l2i",
"Era esperado l2f",
"Era esperado l2d",
"Era esperado f2i",
"Era esperado f2l",
"Era esperado f2d",
"Era esperado d2i",
"Era esperado d2l",
"Era esperado d2f",
"Era esperado i2b",
"Era esperado i2c",
"Era esperado i2s",
"Era esperado lcmp",
"Era esperado fcmpl",
"Era esperado fcmpg",
"Era esperado dcmpl",
"Era esperado dcmpg",
"Era esperado ifeq",
"Era esperado ifne",
"Era esperado iflt",
"Era esperado ifge",
"Era esperado ifgt",
"Era esperado ifle",
"Era esperado if_icmpeq",
"Era esperado if_icmpne",
"Era esperado if_icmplt",
"Era esperado if_icmpge",
"Era esperado if_icmpgt",
"Era esperado if_icmple",
"Era esperado if_acmpeq",
"Era esperado if_acmpne",
"Era esperado goto",
"Era esperado jsr",
"Era esperado ret",
"Era esperado tableswitch",
"Era esperado lookupswitch",
"Era esperado ireturn",
"Era esperado lreturn",
"Era esperado freturn",
"Era esperado dreturn",
"Era esperado areturn",
"Era esperado return",
"Era esperado getstatic",

```
        "Era esperado putstatic",
        "Era esperado getfield",
        "Era esperado putfield",
        "Era esperado invokevirtual",
        "Era esperado invokespecial",
        "Era esperado invokestatic",
        "Era esperado invokeinterface",
        "Era esperado new",
        "Era esperado newarray",
        "Era esperado anewarray",
        "Era esperado arraylength",
        "Era esperado athrow",
        "Era esperado checkcast",
        "Era esperado instanceof",
        "Era esperado monitorenter",
        "Era esperado monitorexit",
        "Era esperado wide",
        "Era esperado multianewarray",
        "Era esperado ifnull",
        "Era esperado ifnonnull",
        "Era esperado goto_w",
        "Era esperado jsr_w",
        "Era esperado breakpoint",
        "Era esperado impdep1",
        "Era esperado impdep2",
        "Era esperado \"\\n\"",
        "Era esperado \":\"",
        "Era esperado \";\"",
        "Era esperado \"(\"",
        "Era esperado \")\"",
        "Era esperado \",\"",
        "Era esperado \"[\"",
        "Era esperado \"]\"",
        "<corpo> inv?lido",
        "<bloco> inv?lido",
        "<listafuncao> inv?lido",
        "<funcao> inv?lido",
        "<nome_funcao> inv?lido",
        "<parametros> inv?lido",
        "<argumentos> inv?lido",
        "<rep_argumentos> inv?lido",
        "<tipo> inv?lido",
        "<retorno> inv?lido",
        "<rep_parametros> inv?lido",
        "<listacomando> inv?lido",
        "<chamada_metodo> inv?lido",
        "<comando> inv?lido",
        "<resto_comando> inv?lido",
        "<constante_explicita> inv?lido",
        "<operacao> inv?lido",
        "<opcode> inv?lido",
        "<label> inv?lido",
        "<desvio> inv?lido",
        "<incremento> inv?lido"
    };
}


/**
 * Arquivo ScannerConstants.java
```

```
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

public interface ScannerConstants
{
   int[] SCANNER_TABLE_INDEXES =
   {
      0,
      76,
      80,
      170,
      170,
      170,
      170,
      180,
      191,
      191,
      191,
      254,
      254,
      255,
      255,
      255,
      265,
      328,
      390,
      390,
      400
   };

   int[][] SCANNER_TABLE =
   {
      {9, 1},
      {10, 1},
      {13, 1},
      {32, 1},
      {34, 2},
      {40, 3},
      {41, 4},
      {44, 5},
      {45, 6},
      {48, 7},
      {49, 7},
      {50, 7},
      {51, 7},
      {52, 7},
      {53, 7},
      {54, 7},
      {55, 7},
      {56, 7},
      {57, 7},
      {58, 8},
      {59, 9},
      {65, 10},
      {66, 10},
      {67, 10},
      {68, 10},
      {69, 10},
      {70, 10},
      {71, 10},
```

{72, 10},
{73, 10},
{74, 10},
{75, 10},
{76, 10},
{77, 10},
{78, 10},
{79, 10},
{80, 10},
{81, 10},
{82, 10},
{83, 10},
{84, 10},
{85, 10},
{86, 10},
{87, 10},
{88, 10},
{89, 10},
{90, 10},
{91, 11},
{92, 12},
{93, 13},
{97, 10},
{98, 10},
{99, 10},
{100, 10},
{101, 10},
{102, 10},
{103, 10},
{104, 10},
{105, 10},
{106, 10},
{107, 10},
{108, 10},
{109, 10},
{110, 10},
{111, 10},
{112, 10},
{113, 10},
{114, 10},
{115, 10},
{116, 10},
{117, 10},
{118, 10},
{119, 10},
{120, 10},
{121, 10},
{122, 10},
{9, 1},
{10, 1},
{13, 1},
{32, 1},
{34, 14},
{40, 2},
{41, 2},
{45, 2},
{47, 2},
{48, 2},
{49, 2},
{50, 2},

{51, 2},
{52, 2},
{53, 2},
{54, 2},
{55, 2},
{56, 2},
{57, 2},
{60, 2},
{62, 2},
{65, 2},
{66, 2},
{67, 2},
{68, 2},
{69, 2},
{70, 2},
{71, 2},
{72, 2},
{73, 2},
{74, 2},
{75, 2},
{76, 2},
{77, 2},
{78, 2},
{79, 2},
{80, 2},
{81, 2},
{82, 2},
{83, 2},
{84, 2},
{85, 2},
{86, 2},
{87, 2},
{88, 2},
{89, 2},
{90, 2},
{95, 2},
{97, 2},
{98, 2},
{99, 2},
{100, 2},
{101, 2},
{102, 2},
{103, 2},
{104, 2},
{105, 2},
{106, 2},
{107, 2},
{108, 2},
{109, 2},
{110, 2},
{111, 2},
{112, 2},
{113, 2},
{114, 2},
{115, 2},
{116, 2},
{117, 2},
{118, 2},
{119, 2},
{120, 2},

{121, 2},
{122, 2},
{192, 2},
{193, 2},
{194, 2},
{199, 2},
{201, 2},
{202, 2},
{205, 2},
{211, 2},
{212, 2},
{220, 2},
{224, 2},
{225, 2},
{226, 2},
{231, 2},
{233, 2},
{234, 2},
{237, 2},
{243, 2},
{244, 2},
{252, 2},
{48, 7},
{49, 7},
{50, 7},
{51, 7},
{52, 7},
{53, 7},
{54, 7},
{55, 7},
{56, 7},
{57, 7},
{46, 15},
{48, 7},
{49, 7},
{50, 7},
{51, 7},
{52, 7},
{53, 7},
{54, 7},
{55, 7},
{56, 7},
{57, 7},
{48, 16},
{49, 16},
{50, 16},
{51, 16},
{52, 16},
{53, 16},
{54, 16},
{55, 16},
{56, 16},
{57, 16},
{65, 10},
{66, 10},
{67, 10},
{68, 10},
{69, 10},
{70, 10},
{71, 10},

{72, 10},
{73, 10},
{74, 10},
{75, 10},
{76, 10},
{77, 10},
{78, 10},
{79, 10},
{80, 10},
{81, 10},
{82, 10},
{83, 10},
{84, 10},
{85, 10},
{86, 10},
{87, 10},
{88, 10},
{89, 10},
{90, 10},
{95, 17},
{97, 10},
{98, 10},
{99, 10},
{100, 10},
{101, 10},
{102, 10},
{103, 10},
{104, 10},
{105, 10},
{106, 10},
{107, 10},
{108, 10},
{109, 10},
{110, 10},
{111, 10},
{112, 10},
{113, 10},
{114, 10},
{115, 10},
{116, 10},
{117, 10},
{118, 10},
{119, 10},
{120, 10},
{121, 10},
{122, 10},
{110, 18},
{48, 19},
{49, 19},
{50, 19},
{51, 19},
{52, 19},
{53, 19},
{54, 19},
{55, 19},
{56, 19},
{57, 19},
{48, 17},
{49, 17},
{50, 17},

{51, 17},
{52, 17},
{53, 17},
{54, 17},
{55, 17},
{56, 17},
{57, 17},
{65, 17},
{66, 17},
{67, 17},
{68, 17},
{69, 17},
{70, 17},
{71, 17},
{72, 17},
{73, 17},
{74, 17},
{75, 17},
{76, 17},
{77, 17},
{78, 17},
{79, 17},
{80, 17},
{81, 17},
{82, 17},
{83, 17},
{84, 17},
{85, 17},
{86, 17},
{87, 17},
{88, 17},
{89, 17},
{90, 17},
{95, 17},
{97, 17},
{98, 17},
{99, 17},
{100, 17},
{101, 17},
{102, 17},
{103, 17},
{104, 17},
{105, 17},
{106, 17},
{107, 17},
{108, 17},
{109, 17},
{110, 17},
{111, 17},
{112, 17},
{113, 17},
{114, 17},
{115, 17},
{116, 17},
{117, 17},
{118, 17},
{119, 17},
{120, 17},
{121, 17},
{122, 17},

{48, 17},
{49, 17},
{50, 17},
{51, 17},
{52, 17},
{53, 17},
{54, 17},
{55, 17},
{56, 17},
{57, 17},
{65, 17},
{66, 17},
{67, 17},
{68, 17},
{69, 17},
{70, 17},
{71, 17},
{72, 17},
{73, 17},
{74, 17},
{75, 17},
{76, 17},
{77, 17},
{78, 17},
{79, 17},
{80, 17},
{81, 17},
{82, 17},
{83, 17},
{84, 17},
{85, 17},
{86, 17},
{87, 17},
{88, 17},
{89, 17},
{90, 17},
{97, 17},
{98, 17},
{99, 17},
{100, 17},
{101, 17},
{102, 17},
{103, 17},
{104, 17},
{105, 17},
{106, 17},
{107, 17},
{108, 17},
{109, 17},
{110, 17},
{111, 17},
{112, 17},
{113, 17},
{114, 17},
{115, 17},
{116, 17},
{117, 17},
{118, 17},
{119, 17},
{120, 17},

```
    {121, 17},
    {122, 17},
    {48, 19},
    {49, 19},
    {50, 19},
    {51, 19},
    {52, 19},
    {53, 19},
    {54, 19},
    {55, 19},
    {56, 19},
    {57, 19}
  };

  int[] TOKEN_STATE = { 0,  0, -1, 223, 224, 225, -1,  4, 221, 222,  2, 226, -1, 227,  3, -1,  2,  2, 220,  5 };

  int[] SPECIAL_CASES_INDEXES =
    { 0, 0, 0, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214,
214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214,
214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214,
214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214,
214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214,
214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214,
214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214,
214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214,
214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214,
214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214, 214 };

  String[] SPECIAL_CASES_KEYS =
    {  "aaload", "aastore", "aconst_null", "aload", "aload_0", "aload_1", "aload_2", "aload_3", "anewarray", "areturn",
"arraylength", "astore", "astore_0", "astore_1", "astore_2", "astore_3", "athrow", "baload", "bastore", "bipush", "bool",
"breakpoint", "caload", "castore", "checkcast", "d2f", "d2i", "d2l", "dadd", "daload", "dastore", "dcmpg", "dcmpl",
"dconst_0", "dconst_1", "ddiv", "dload", "dload_0", "dload_1", "dload_2", "dload_3", "dmul", "dneg", "double",
"drem", "dreturn", "dstore", "dstore_0", "dstore_1", "dstore_2", "dstore_3", "dsub", "dup", "dup2", "dup2_x1",
"dup2_x2", "dup_x1", "dup_x2", "f2d", "f2i", "f2l", "fadd", "faload", "fastore", "fcmpg", "fcmpl", "fconst_0",
"fconst_1", "fconst_2", "fdiv", "fim", "fimprograma", "fload", "fload_0", "fload_1", "fload_2", "fload_3", "float",
"fmul", "fneg", "frem", "freturn", "fstore", "fstore_0", "fstore_1", "fstore_2", "fstore_3", "fsub", "getfield", "getstatic",
"goto", "goto_w", "i2b", "i2c", "i2d", "i2f", "i2l", "i2s", "iadd", "iaload", "iand", "iastore", "iconst_0", "iconst_1",
"iconst_2", "iconst_3", "iconst_4", "iconst_5", "iconst_m1", "idiv", "if_acmpeq", "if_acmpne", "if_icmpeq",
"if_icmpge", "if_icmpgt", "if_icmple", "if_icmplt", "if_icmpne", "ifeq", "ifge", "ifgt", "ifle", "iflt", "ifne", "ifnonnull",
"ifnull", "iinc", "iload", "iload_0", "iload_1", "iload_2", "iload_3", "impdep1", "impdep2", "imul", "ineg", "inicio",
"inicioprograma", "instanceof", "int", "invokeinterface", "invokespecial", "invokestatic", "invokevirtual", "ior", "irem",
"ireturn", "ishl", "ishr", "istore", "istore_0", "istore_1", "istore_2", "istore_3", "isub", "iushr", "ixor", "jsr", "jsr_w",
"l2d", "l2f", "l2i", "ladd", "laload", "land", "lastore", "lcmp", "lconst_0", "lconst_1", "ldc", "ldc2_w", "ldc_w", "ldiv",
"lload", "lload_0", "lload_1", "lload_2", "lload_3", "lmul", "lneg", "lookupswitch", "lor", "lrem", "lreturn", "lshl",
"lshr", "lstore", "lstore_0", "lstore_1", "lstore_2", "lstore_3", "lsub", "lushr", "lxor", "monitorenter", "monitorexit",
"multianewarray", "new", "newarray", "nop", "pop", "pop2", "putfield", "putstatic", "ret", "return", "saload", "sastore",
"sipush", "string", "swap", "tableswitch", "void", "wide" };

  int[] SPECIAL_CASES_VALUES =
    {  66, 99, 17, 41, 58, 59, 60, 61, 204, 192, 205, 74, 91, 92, 93, 94, 206, 67, 100, 32, 13, 217, 68, 101, 207, 160,
158, 159, 115, 65, 98, 168, 167, 30, 31, 127, 40, 54, 55, 56, 57, 123, 135, 11, 131, 191, 73, 87, 88, 89, 90, 119, 105, 108,
109, 110, 106, 107, 157, 155, 156, 114, 64, 97, 166, 165, 27, 28, 29, 126, 7, 9, 39, 50, 51, 52, 53, 12, 122, 134, 130, 190,
72, 83, 84, 85, 86, 118, 196, 194, 183, 215, 161, 162, 151, 150, 149, 163, 112, 62, 142, 95, 19, 20, 21, 22, 23, 24, 18,
124, 181, 182, 175, 178, 179, 180, 177, 176, 169, 172, 173, 174, 171, 170, 214, 213, 148, 37, 42, 43, 44, 45, 218, 219,
120, 132, 6, 8, 208, 10, 201, 199, 200, 198, 144, 128, 188, 136, 138, 70, 75, 76, 77, 78, 116, 140, 146, 184, 216, 154,
153, 152, 113, 63, 143, 96, 164, 25, 26, 34, 36, 35, 125, 38, 46, 47, 48, 49, 121, 133, 187, 145, 129, 189, 137, 139, 71,
79, 80, 81, 82, 117, 141, 147, 209, 210, 212, 202, 203, 16, 103, 104, 197, 195, 185, 193, 69, 102, 33, 14, 111, 186, 15,
211 };
```

```
   String[] SCANNER_ERROR =
   {
      "Caractere n?o esperado",
      "",
      "Erro identificando literal",
      "",
      "",
      "",
      "Erro identificando inteiro ou real",
      "",
      "",
      "",
      "",
      "",
      "Erro identificando \"\n\"",
      "",
      "",
      "Erro identificando real",
      "",
      "",
      "",
      ""
   };

}


/**
 * Arquivo SemanticError.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

public class SemanticError extends AnalysisError
{
            private static final long serialVersionUID = 1L;

            public SemanticError(String msg, int position)
             {
      super(msg, position);
   }

   public SemanticError(String msg)
   {
      super(msg);
   }
}


/**
 * Arquivo Semantico.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes.BaseAcaoSemantica;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator.Class;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator.MethodSignature;

import java.util.HashMap;
import java.util.Map;
import java.util.Stack;
```

```
public class Semantico implements Constants {

	private boolean unconditionalBranch;

	private int tempVars;
	private int tempArray;
	private int tempPointers;
	private int tempSeqs;

	private Stack<Integer> arrayLength;
	private Map<MethodSignature,Stack<Integer>> labels;

	private Stack<String> varInicializadas;
	private Stack<Integer> pilha;
	private Stack<String> pilhaDeArrays;
	private Stack<String> pilhaDePonteiros;
	private Stack<Integer> pilhaDeInteiros;
	private Stack<String> pilhaDeLiterais;
	private Stack<Float> pilhaDeReais;
	private Stack<String> pilhaDeInstrucoes;
	private boolean usingTypeForReturn;
	private String currentMethod;

	private Class classe;

	private Map<String,BaseAcaoSemantica> classes;

	public Semantico() {
	}

	public void clear() {

		unconditionalBranch = false;

		tempVars = 0;
		tempArray = 0;
		tempPointers = 0;
		tempSeqs = 0;

		classes = new HashMap<String, BaseAcaoSemantica>();
		labels = new HashMap<MethodSignature, Stack<Integer>>();
		classe = new Class();

		arrayLength = new Stack<Integer>();
		pilhaDeInteiros = new Stack<Integer>();
		pilhaDeLiterais = new Stack<String>();
		pilhaDePonteiros = new Stack<String>();
		pilhaDeArrays = new Stack<String>();
		varInicializadas = new Stack<String>();
		pilhaDeReais = new Stack<Float>();
		pilhaDeInstrucoes = new Stack<String>();
		pilha = new Stack<Integer>();
	}

	private Stack<String> getVarInicializadas() {

		return varInicializadas;
	}
```

```
private void cleanVarinicializadas() {

	getVarInicializadas().clear();
}

public boolean temVarNaPilha( String var ) {

	return getVarInicializadas().contains( var );
}

public void pushVarParaPilha( String var ) {

	getVarInicializadas().push( var );
}

public String popVarDaPilha() {

	return getVarInicializadas().pop();
}

public void removeUnconditionalBranch() {

	unconditionalBranch = false;
}

public void addUnconditionalBranch() {

	unconditionalBranch = true;
}

public boolean haveUnconditionalBranch() {

	return unconditionalBranch;
}

private Map<MethodSignature,Stack<Integer>> getLabels() {

	return labels;
}

public Stack<Integer> getLabels( MethodSignature method) {

	if ( !getLabels().containsKey( method ) ) {

		getLabels().put( method, new Stack<Integer>() );
	}

	return getLabels().get( method );
}

private Stack<Integer> getArraysLength() {

	return arrayLength;
}

public void pushArrayLenghNaPilha( int value ) {

	getArraysLength().push( value );
}
```

```
public int popArrayLenghDaPilha() {

	//return getArraysLength().pop();
	return getArraysLength().get( getArraysLength().size()-1 );
}

public void pushLabelNaPilha( MethodSignature method, int value ) {

	getLabels( method ).push( value );
}

public int popLabelDaPilha( MethodSignature method ) {

	return getLabels( method ).pop();
}

public boolean pilhaTemLabel( MethodSignature method, int label ) {

	return getLabels( method ).contains( label );
}

private Stack<Integer> getPilha() {

	return pilha;
}

public void pushTipoParaPilha( int type ) {

	getPilha().push( type );
}

public void clearTiposDaPilha() {

	getPilha().clear();
}

public int rpopTipoDaPilha() {

	if ( getPilha().size() > 0 ) {

		return getPilha().remove( 0 );
	}

	return 0;
}

public int popTipoDaPilha() {

	return getPilha().pop();
}

public int popCurrentTempSeqs() {

	int temp = tempSeqs;
	tempSeqs--;
	return temp;
}

public void useTempSeqs() {
```

```
		tempSeqs++;
}

public int getCurrentTempSeqs() {

		int temp = tempSeqs;
		useTempSeqs();
		return temp;
}

public void clearTempSeqs() {

		tempSeqs = 0;
}

public int popCurrentTempPointers() {

		int temp = tempPointers;
		tempPointers--;
		return temp;
}

public void useTempPointers() {

		tempPointers++;
}

public int getCurrentTempPointers() {

		int temp = tempPointers;
		useTempPointers();
		return temp;
}

public void clearTempPointers() {

		tempPointers = 0;
}

public int popCurrentTempArray() {

		int temp = tempArray;
		tempArray--;
		return temp;
}

public void useTempArray() {

		tempVars++;
}

public int getCurrentTempArray() {

		int temp = tempArray;
		useTempArray();
		return temp;
}

public void clearTempArray() {
```

```
        tempArray = 0;
}

public int popCurrentTempVar() {

        int temp = tempVars;
        tempVars--;
        return temp;
}

public void useTempVars() {

        tempVars++;
}

public int getCurrentTempVar() {

        int temp = tempVars;
        useTempVars();
        return temp;
}

public void clearTempVars() {

        tempVars = 0;
}

public void pushInteiroNaPilha( int value ) {

        pilhaDeInteiros.push( value );
}

public int rpopInteiroDaPilha() {

        return pilhaDeInteiros.remove( 0 );
}

public int popInteiroDaPilha() {

        return pilhaDeInteiros.pop();
}

public void pushLiteralNaPilha( String value ) {

        pilhaDeLiterais.push( value );
}

public String popLiteralDaPilha() {

        return pilhaDeLiterais.pop();
}

public boolean pilhaTemArrays( String array ) {

        return pilhaDeArrays.contains( array );
}

private void clearPilhaDeArrays() {

        pilhaDePonteiros.clear();
```

```
}

public void pushArrayNaPilha( String value ) {

	pilhaDeArrays.push( value );
}

public String rpopArrayDaPilha() {

	return pilhaDeArrays.remove( 0 );
}

public String popArrayDaPilha() {

	return pilhaDeArrays.pop();
}

public boolean pilhaTemPonteiro( String ponteiro ) {

	return pilhaDePonteiros.contains( ponteiro );
}

private void clearPilhaDePonteiros() {

	pilhaDePonteiros.clear();
}

public void pushPonteiroNaPilha( String value ) {

	pilhaDePonteiros.push( value );
}

public String rpopPonteiroDaPilha() {

	return pilhaDePonteiros.remove( 0 );
}

public String popPonteiroDaPilha() {

	return pilhaDePonteiros.pop();
}

public void pushInstrucaoNaPilha( String value ) {

	pilhaDeInstrucoes.push( value );
}

public String rpopInstrucaoDaPilha() {

	return pilhaDeInstrucoes.remove( 0 );
}

public String popInstrucaoDaPilha() {

	return pilhaDeInstrucoes.pop();
}

public void pushRealNaPilha( float value ) {

	pilhaDeReais.push( value );
```

```
}

public float popRealDaPilha() {

	return pilhaDeReais.pop();
}

public Class getClasse() {

	return classe;
}

public void setUsingTypeForReturn( boolean usingTypeForReturn ) {

	this.usingTypeForReturn = usingTypeForReturn;
}

public boolean isUsingTypeForReturn() {

	return usingTypeForReturn;
}

public MethodSignature getCurrentMethod() {

	if ( currentMethod != null ) {

		if ( getMethods().containsKey( currentMethod ) ) {

			return getMethods().get( currentMethod );
		}
	}

	return null;
}
private void updateCurrentMethod( String methodName ) {

	this.currentMethod = methodName;
}

public void printCurrentMethod() {

	System.out.println( getCurrentMethod() );
}

public Map<String,MethodSignature> getMethods() {

	return getClasse().getMethods();
}

public void addMethod( String methodName ) {

	clearPilhaDeArrays();
	clearPilhaDePonteiros();
	clearTiposDaPilha();
	clearTempPointers();
	clearTempSeqs();
	cleanVarinicializadas();

	if ( !getMethods().containsKey( methodName ) ) {
```

```
                getMethods().put( methodName,
                        new MethodSignature( methodName ) );
        }

        updateCurrentMethod( methodName );
}

public void addMethod( MethodSignature method ) {

        clearTiposDaPilha();

        if ( !getMethods().containsKey( method.getName() ) ) {

                getMethods().put( method.getName(), method );
        }

        updateCurrentMethod( method.getName() );
}

private Map<String,BaseAcaoSemantica> getClasses() {

        return classes;
}

private BaseAcaoSemantica getClasse( String className ) {

        BaseAcaoSemantica acao;

        if ( !getClasses().containsKey( className ) ) {

                try {

                        acao = (BaseAcaoSemantica)java.lang.Class.
                                forName( className ).newInstance();

                        acao.conhecaoSemantico( this );
                        getClasses().put( className,
                                (BaseAcaoSemantica) acao );

                } catch ( Exception e ) {

                        return null;
                }
        }

        return getClasses().get( className );
}

public void executeAction( int action,
                Token token ) throws SemanticError {

        String className = "inf.ufsc.br.tcc.cantu.parser." +
                "jvm2llvm.acoes.AcaoSemantica"+ action;

        BaseAcaoSemantica acao =
                 getClasse( className );

        if ( acao != null ) {

                try {
```

```
                                        acao.executaAcao( token );

                                } catch ( Exception e ) {

                                        e.printStackTrace();
                                }
                        }
                }
}

/**
 * Arquivo Sintatico.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

import java.util.Stack;

public class Sintatico implements Constants
{
   private Stack<Integer> stack = new Stack<Integer>();
   private Token currentToken;
   private Token previousToken;
   private Lexico scanner;
   private Semantico semanticAnalyser;

   private static final boolean isTerminal(int x)
   {
      return x < FIRST_NON_TERMINAL;
   }

   private static final boolean isNonTerminal(int x)
   {
      return x >= FIRST_NON_TERMINAL && x < FIRST_SEMANTIC_ACTION;
   }

   @SuppressWarnings("unused")
         private static final boolean isSemanticAction(int x)
   {
      return x >= FIRST_SEMANTIC_ACTION;
   }

   private boolean step() throws LexicalError, SyntaticError, SemanticError
   {
      if (currentToken == null)
      {
         int pos = 0;
         if (previousToken != null)
            pos = previousToken.getPosition()+previousToken.getLexeme().length();

         currentToken = new Token(DOLLAR, "$", pos);
      }

      int x = ((Integer)stack.pop()).intValue();
      int a = currentToken.getId();

      if (x == EPSILON)
      {
         return false;
      }
```

```
    else if (isTerminal(x))
    {
        if (x == a)
        {
            if (stack.empty())
                return true;
            else
            {
                previousToken = currentToken;
                currentToken = scanner.nextToken();
                return false;
            }
        }
        else
        {
            throw new SyntaticError(PARSER_ERROR[x], currentToken.getPosition());
        }
    }
    else if (isNonTerminal(x))
    {
        if (pushProduction(x, a))
            return false;
        else
            throw new SyntaticError(PARSER_ERROR[x], currentToken.getPosition());
    }
    else // isSemanticAction(x)
    {
        semanticAnalyser.executeAction(x-FIRST_SEMANTIC_ACTION, previousToken);
        return false;
    }
}

private boolean pushProduction(int topStack, int tokenInput)
{
    int p = PARSER_TABLE[topStack-FIRST_NON_TERMINAL][tokenInput-1];
    if (p >= 0)
    {
        int[] production = PRODUCTIONS[p];
        //empilha a produ??o em ordem reversa
        for (int i=production.length-1; i>=0; i--)
        {
            stack.push(new Integer(production[i]));
        }
        return true;
    }
    else
        return false;
}

public void parse(Lexico scanner, Semantico semanticAnalyser) throws LexicalError, SyntaticError, SemanticError
{
    this.scanner = scanner;
    this.semanticAnalyser = semanticAnalyser;

    stack.clear();
    stack.push(new Integer(DOLLAR));
    stack.push(new Integer(START_SYMBOL));

    currentToken = scanner.nextToken();
```

```
        while ( ! step() )
            ;
    }
}


/**
 * Arquivo SyntaticError.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

public class SyntaticError extends AnalysisError
{

            private static final long serialVersionUID = 1L;

            public SyntaticError(String msg, int position)
             {
        super(msg, position);
    }

    public SyntaticError(String msg)
    {
        super(msg);
    }
}


/**
 * Arquivo Token.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm;

public class Token
{
    private int id;
    private String lexeme;
    private int position;

    public Token(int id, String lexeme, int position)
    {
        this.id = id;
        this.lexeme = lexeme;
        this.position = position;
    }

    public final int getId()
    {
        return id;
    }

    public final String getLexeme()
    {
        return lexeme;
    }

    public final int getPosition()
    {
        return position;
    }
```

```
   public String toString()
   {
      return id+" ( "+lexeme+" ) @ "+position;
   };
}


/**
 * Arquivo AcaoSemantica.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Semantico;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public interface AcaoSemantica {

	public void conhecaoSemantico( Semantico semantico );
	public void executaAcao( Token token );
}

/**
 * Arquivo AcaoSemantica1.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica1 extends BaseAcaoSemantica {

	public AcaoSemantica1() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
	}
}


/**
 * Arquivo AcaoSemantica10.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica10 extends BaseAcaoSemantica {

	public AcaoSemantica10() {

		super();
	}

	/**
    * J? sabemos o m?todo
    * e temos uma par?metro atual.
	 */
```

```
	@Override
	public void executaAcao( Token token ) {

		if ( getSemantico().isUsingTypeForReturn() ) {

			getSemantico().getCurrentMethod().setReturnType(
				token.getLexeme() );

			getSemantico().setUsingTypeForReturn( false );

		} else {

			getSemantico().getCurrentMethod().
				getCurrentParameter().setType( token.getLexeme() );
		}
	}
}

/**
 * Arquivo AcaoSemantica101.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica101 extends BaseAcaoSemantica {

	public AcaoSemantica101() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica102.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica102 extends BaseAcaoSemantica {

	public AcaoSemantica102() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}
```

```
/**
 * Arquivo AcaoSemantica103.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica103 extends BaseAcaoSemantica {

	public AcaoSemantica103() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		getSemantico().pushInteiroNaPilha( 0 );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_INT );
	}
}

/**
 * Arquivo AcaoSemantica104.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica104 extends BaseAcaoSemantica {

	public AcaoSemantica104() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		getSemantico().pushInteiroNaPilha( 1 );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_INT );
	}
}

/**
 * Arquivo AcaoSemantica105.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;
```

```
public class AcaoSemantica105 extends BaseAcaoSemantica {

	public AcaoSemantica105() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		getSemantico().pushInteiroNaPilha( 2 );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_INT );
	}
}

/**
 * Arquivo AcaoSemantica106.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica106 extends BaseAcaoSemantica {

	public AcaoSemantica106() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		getSemantico().pushInteiroNaPilha( 3 );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_INT );
	}
}

/**
 * Arquivo AcaoSemantica107.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica107 extends BaseAcaoSemantica {

	public AcaoSemantica107() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
```

```
                getSemantico().pushInteiroNaPilha( 4 );

                getSemantico().pushTipoParaPilha(
                        Constantes.STACK_TYPE_INT );
        }
}

/**
 * Arquivo AcaoSemantica108.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica108 extends BaseAcaoSemantica {

        public AcaoSemantica108() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {

                getSemantico().pushInteiroNaPilha( 5 );

                getSemantico().pushTipoParaPilha(
                        Constantes.STACK_TYPE_INT );
        }
}

/**
 * Arquivo AcaoSemantica109.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica109 extends BaseAcaoSemantica {

        public AcaoSemantica109() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {


        }
}


/**
 * Arquivo AcaoSemantica11.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
```

```
public class AcaoSemantica11 extends BaseAcaoSemantica {

	public AcaoSemantica11() {

		super();
	}

	/**
    * J? sabemos o m?todo
    * e temos uma par?metro atual.
	 */
	@Override
	public void executaAcao( Token token ) {

		if ( getSemantico().isUsingTypeForReturn() ) {

			getSemantico().getCurrentMethod().setReturnType(
				token.getLexeme() );

			getSemantico().setUsingTypeForReturn( false );

		} else {

			getSemantico().getCurrentMethod().
				getCurrentParameter().setType( token.getLexeme() );
		}
	}
}

/**
 * Arquivo AcaoSemantica110.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica110 extends BaseAcaoSemantica {

	public AcaoSemantica110() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica111.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica111 extends BaseAcaoSemantica {
```

```
	public AcaoSemantica111() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica112.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica112 extends BaseAcaoSemantica {

	public AcaoSemantica112() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica113.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica113 extends BaseAcaoSemantica {

	public AcaoSemantica113() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica114.java
```

```
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica114 extends BaseAcaoSemantica {

	public AcaoSemantica114() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica115.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica115 extends BaseAcaoSemantica {

	public AcaoSemantica115() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica116.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica116 extends BaseAcaoSemantica {

	public AcaoSemantica116() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
	}
}
```

```
/**
 * Arquivo AcaoSemantica117.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica117 extends BaseAcaoSemantica {

	public AcaoSemantica117() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica118.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica118 extends BaseAcaoSemantica {

	public AcaoSemantica118() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica119.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica119 extends BaseAcaoSemantica {

	public AcaoSemantica119() {

		super();
	}

	@Override
```

```
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica12.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica12 extends BaseAcaoSemantica {

	public AcaoSemantica12() {

		super();
	}

	/**
    * J? sabemos o m?todo
    * e temos uma par?metro atual.
	 */
	@Override
	public void executaAcao( Token token ) {

		getSemantico().getCurrentMethod().
			getCurrentParameter().setName( token.getLexeme() );
	}
}


/**
 * Arquivo AcaoSemantica120.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica120 extends BaseAcaoSemantica {

	public AcaoSemantica120() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica121.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;
```

```
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica121 extends BaseAcaoSemantica {

	public AcaoSemantica121() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}

/**
 * Arquivo AcaoSemantica122.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica122 extends BaseAcaoSemantica {

	public AcaoSemantica122() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica123.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica123 extends BaseAcaoSemantica {

	public AcaoSemantica123() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}
```

```
/**
 * Arquivo AcaoSemantica124.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica124 extends BaseAcaoSemantica {

	public AcaoSemantica124() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica125.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica125 extends BaseAcaoSemantica {

	public AcaoSemantica125() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica126.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica126 extends BaseAcaoSemantica {

	public AcaoSemantica126() {

		super();
	}
```

```
	@Override
	public void executaAcao( Token token ) {

		/*
		getSemantico().
			pushInstrucaoNaPilha( "%tempint0" );
		*/

		getSemantico().pushInstrucaoNaPilha(
			"%ptr0" );

		getSemantico().pushPonteiroNaPilha(
			"%ptr0" );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_VAR );
	}
}

/**
 * Arquivo AcaoSemantica127.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica127 extends BaseAcaoSemantica {

	public AcaoSemantica127() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		/*
		getSemantico().pushInstrucaoNaPilha( "%tempint1" );
		*/

		getSemantico().pushInstrucaoNaPilha(
			"%ptr1" );

		getSemantico().pushPonteiroNaPilha(
			"%ptr1" );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_VAR );
	}
}

/**
 * Arquivo AcaoSemantica128.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;
```

```
public class AcaoSemantica128 extends BaseAcaoSemantica {

	public AcaoSemantica128() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		/*
		getSemantico().pushInstrucaoNaPilha( "%tempint2" );
		*/

		getSemantico().pushInstrucaoNaPilha(
			"%ptr2" );

		getSemantico().pushPonteiroNaPilha(
			"%ptr2" );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_VAR );
	}
}

/**
 * Arquivo AcaoSemantica129.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica129 extends BaseAcaoSemantica {

	public AcaoSemantica129() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		/*
		getSemantico().pushInstrucaoNaPilha( "%tempint3" );
		*/

		getSemantico().pushInstrucaoNaPilha(
			"%ptr3" );

		getSemantico().pushPonteiroNaPilha(
			"%ptr3" );

		getSemantico().pushTipoParaPilha(
				Constantes.STACK_TYPE_VAR );
	}
}

/**
 * Arquivo AcaoSemantica13.java
```

```
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica13 extends BaseAcaoSemantica {

	public AcaoSemantica13() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
	}
}


/**
 * Arquivo AcaoSemantica130.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica130 extends BaseAcaoSemantica {

	public AcaoSemantica130() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica131.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica131 extends BaseAcaoSemantica {

	public AcaoSemantica131() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}
```

```
/**
 * Arquivo AcaoSemantica132.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica132 extends BaseAcaoSemantica {

	public AcaoSemantica132() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica133.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica133 extends BaseAcaoSemantica {

	public AcaoSemantica133() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica134.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica134 extends BaseAcaoSemantica {

	public AcaoSemantica134() {

		super();
	}
```

```
	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica135.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica135 extends BaseAcaoSemantica {

	public AcaoSemantica135() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica136.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica136 extends BaseAcaoSemantica {

	public AcaoSemantica136() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica137.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica137 extends BaseAcaoSemantica {
```

```
	public AcaoSemantica137() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica138.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica138 extends BaseAcaoSemantica {

	public AcaoSemantica138() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica139.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica139 extends BaseAcaoSemantica {

	public AcaoSemantica139() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica14.java
```

```
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica14 extends BaseAcaoSemantica {

	public AcaoSemantica14() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

	}
}

/**
 * Arquivo AcaoSemantica140.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica140 extends BaseAcaoSemantica {

	public AcaoSemantica140() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica141.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica141 extends BaseAcaoSemantica {

	public AcaoSemantica141() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}
```

```
/**
 * Arquivo AcaoSemantica142.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica142 extends BaseAcaoSemantica {

	public AcaoSemantica142() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		getSemantico().pushInstrucaoNaPilha(
			"%array0" );
		getSemantico().pushArrayNaPilha( "%array0" );
	}
}

/**
 * Arquivo AcaoSemantica143.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica143 extends BaseAcaoSemantica {

	public AcaoSemantica143() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica144.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica144 extends BaseAcaoSemantica {

	public AcaoSemantica144() {

		super();
	}
```

```
	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica145.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica145 extends BaseAcaoSemantica {

	public AcaoSemantica145() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica146.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica146 extends BaseAcaoSemantica {

	public AcaoSemantica146() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		String op = "%temp"+ getSemantico().
			getCurrentTempVar();

		String index = "";

		switch ( getSemantico().popTipoDaPilha() ) {

			case Constantes.STACK_TYPE_INT:
				index += getSemantico().popInteiroDaPilha();
				break;
```

```
                        case Constantes.STACK_TYPE_VAR:

                                String operando = getSemantico().
                                        popInstrucaoDaPilha();

                                if ( getSemantico().pilhaTemPonteiro(
                                        operando ) ) {

                                        String tempvar = operando+"temp"+getSemantico().
                                                getCurrentTempVar();

                                        getSemantico().getCurrentMethod().
                                                addCommad( tempvar+" = load i32* "+operando );
                                        operando = tempvar;
                                }

                                index += operando;
                                break;

                        default:
                                break;
                }

                String operand1 =
                        getSemantico().popInstrucaoDaPilha();

                String ptr = "%arrayptr"+ getSemantico().
                        getCurrentTempPointers();

                getSemantico().getCurrentMethod().
                        addCommad( ptr +" = getelementptr i32* "+
                                operand1 +", i32 "+ index );

                getSemantico().getCurrentMethod().
                        addCommad( op +" = load i32* "+ptr );

                getSemantico().pushInstrucaoNaPilha( op );
                getSemantico().pushTipoParaPilha(
                        Constantes.STACK_TYPE_VAR );
        }
}

/**
 * Arquivo AcaoSemantica147.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica147 extends BaseAcaoSemantica {

        public AcaoSemantica147() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {
```

```
	}
}


/**
 * Arquivo AcaoSemantica148.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica148 extends BaseAcaoSemantica {

	public AcaoSemantica148() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica149.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica149 extends BaseAcaoSemantica {

	public AcaoSemantica149() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica15.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica15 extends BaseAcaoSemantica {

	public AcaoSemantica15() {
```

```
		super();
	}

	@Override
	public void executaAcao( Token token ) {

		getSemantico().pushInteiroNaPilha(
			Integer.parseInt( token.getLexeme() ) );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_INT );
	}
}

/**
 * Arquivo AcaoSemantica150.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica150 extends BaseAcaoSemantica {

	public AcaoSemantica150() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica151.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica151 extends BaseAcaoSemantica {

	public AcaoSemantica151() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica152.java
```

```
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica152 extends BaseAcaoSemantica {

	public AcaoSemantica152() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica153.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica153 extends BaseAcaoSemantica {

	public AcaoSemantica153() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica154.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica154 extends BaseAcaoSemantica {

	public AcaoSemantica154() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
```

```
	}
}


/**
 * Arquivo AcaoSemantica155.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica155 extends BaseAcaoSemantica {

	public AcaoSemantica155() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica156.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica156 extends BaseAcaoSemantica {

	public AcaoSemantica156() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica157.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica157 extends BaseAcaoSemantica {

	public AcaoSemantica157() {

		super();
```

```
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica158.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica158 extends BaseAcaoSemantica {

	public AcaoSemantica158() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica159.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica159 extends BaseAcaoSemantica {

	public AcaoSemantica159() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		if ( !getSemantico().temVarNaPilha( "%intvar0" ) ) {

			getSemantico().getCurrentMethod().addCommad(
				"%intvar0 = alloca i32" );

			getSemantico().getCurrentMethod().addCommad(
				"%ptro = getelementptr i32* %intvar0" );

			getSemantico().pushVarParaPilha( "%intvar0" );
		}
```

```
switch ( getSemantico().popTipoDaPilha() ) {

	case Constantes.STACK_TYPE_INT:
		getSemantico().getCurrentMethod().
			addCommad( "store i32 "+
				getSemantico().popInteiroDaPilha() +
					", i32* %ptr0");
		break;

	case Constantes.STACK_TYPE_VAR:

		String operando = getSemantico().
			popInstrucaoDaPilha();

		if ( getSemantico().pilhaTemPonteiro(
			operando ) ) {

			String tempvar = operando+"temp"+getSemantico().
				getCurrentTempVar();

			getSemantico().getCurrentMethod().
				addCommad( tempvar+" = load i32* "+operando );
			operando = tempvar;
		}

		getSemantico().getCurrentMethod().
			addCommad( "store i32 "+ operando +
				", i32* %ptr0");
		break;

	default:
		break;
}

if ( !getSemantico().temVarNaPilha( "%intvar0" ) ) {

	getSemantico().getCurrentMethod().addCommad(
		"%tempint0 = load i32* %ptr0" );
}

getSemantico().pushInstrucaoNaPilha(
	"%ptr0" );

getSemantico().pushPonteiroNaPilha(
	"%ptr0" );

/*
getSemantico().getCurrentMethod().addCommad(
	"%intvar0 = add i32 0," );

switch ( getSemantico().popTipoDaPilha() ) {

	case Constantes.STACK_TYPE_INT:
		getSemantico().getCurrentMethod().getCurrentCommand().
			setRightSide( String.valueOf(
				getSemantico().popInteiroDaPilha() ) );
		break;

	case Constantes.STACK_TYPE_VAR:
```

```
                                        getSemantico().getCurrentMethod().getCurrentCommand().
                                                setRightSide( getSemantico().
                                                        popInstrucaoDaPilha() );
                                        break;
                        }

                        getSemantico().getCurrentMethod().getCurrentCommand().
                                setOperation( "" );

                        getSemantico().pushInstrucaoNaPilha( "%intvar0");
                        */
                }
}

/**
 * Arquivo AcaoSemantica16.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica16 extends BaseAcaoSemantica {

        public AcaoSemantica16() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {


        }
}


/**
 * Arquivo AcaoSemantica160.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica160 extends BaseAcaoSemantica {

        public AcaoSemantica160() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {

                if ( !getSemantico().temVarNaPilha( "%intvar1" ) ) {

                        getSemantico().getCurrentMethod().addCommad(
                                "%intvar1 = alloca i32" );

                        getSemantico().getCurrentMethod().addCommad(
```

```
                "%ptr1 = getelementptr i32* %intvar1" );

        getSemantico().pushVarParaPilha( "%intvar1" );
}

switch ( getSemantico().popTipoDaPilha() ) {

        case Constantes.STACK_TYPE_INT:
                getSemantico().getCurrentMethod().
                        addCommad( "store i32 "+
                                getSemantico().popInteiroDaPilha() +
                                        ", i32* %ptr1");
                break;

        case Constantes.STACK_TYPE_VAR:

                String operando = getSemantico().
                        popInstrucaoDaPilha();

                if ( getSemantico().pilhaTemPonteiro(
                        operando ) ) {

                        String tempvar = operando+"temp"+getSemantico().
                                getCurrentTempVar();

                        getSemantico().getCurrentMethod().
                                addCommad( tempvar+" = load i32* "+operando );
                        operando = tempvar;
                }

                getSemantico().getCurrentMethod().
                        addCommad( "store i32 "+ operando +
                                ", i32* %ptr1");
                break;

        default:
                break;
}

if ( !getSemantico().temVarNaPilha( "%intvar1" ) ) {

        getSemantico().getCurrentMethod().addCommad(
                "%tempint1 = load i32* %ptr1" );
}

getSemantico().pushInstrucaoNaPilha(
        "%ptr1" );

getSemantico().pushPonteiroNaPilha(
        "%ptr1" );

/*
getSemantico().getCurrentMethod().addCommad(
        "%intvar1 = add i32 0," );

switch ( getSemantico().popTipoDaPilha() ) {

        case Constantes.STACK_TYPE_INT:
                getSemantico().getCurrentMethod().getCurrentCommand().
                        setRightSide( String.valueOf(
```

```
                                        getSemantico().popInteiroDaPilha() ) );
                                break;

                        case Constantes.STACK_TYPE_VAR:
                                getSemantico().getCurrentMethod().getCurrentCommand().
                                        setRightSide( getSemantico().
                                                popInstrucaoDaPilha() );
                                break;
                }

                getSemantico().getCurrentMethod().getCurrentCommand().
                        setOperation( "" );

                getSemantico().pushInstrucaoNaPilha( "%intvar1");
                */
        }
}

/**
 * Arquivo AcaoSemantica161.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica161 extends BaseAcaoSemantica {

        public AcaoSemantica161() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {

                if ( !getSemantico().temVarNaPilha( "%intvar2" ) ) {

                        getSemantico().getCurrentMethod().addCommad(
                                "%intvar2 = alloca i32" );

                        getSemantico().getCurrentMethod().addCommad(
                                "%ptr2 = getelementptr i32* %intvar2" );

                        getSemantico().pushVarParaPilha( "%intvar2" );
                }

                switch ( getSemantico().popTipoDaPilha() ) {

                        case Constantes.STACK_TYPE_INT:
                                getSemantico().getCurrentMethod().
                                        addCommad( "store i32 "+
                                                getSemantico().popInteiroDaPilha() +
                                                        ", i32* %ptr2");
                                break;

                        case Constantes.STACK_TYPE_VAR:

                                String operando = getSemantico().
                                        popInstrucaoDaPilha();
```

```
                                if ( getSemantico().pilhaTemPonteiro(
                                        operando ) ) {

                                        String tempvar = operando+"temp"+getSemantico().
                                                getCurrentTempVar();

                                        getSemantico().getCurrentMethod().
                                                addCommad( tempvar+" = load i32* "+operando );
                                        operando = tempvar;
                                }

                                getSemantico().getCurrentMethod().
                                        addCommad( "store i32 "+ operando +
                                                ", i32* %ptr2");
                                break;

                        default:
                                break;
                }

                if ( !getSemantico().temVarNaPilha( "%intvar2" ) ) {

                        getSemantico().getCurrentMethod().addCommad(
                                "%tempint2 = load i32* %ptr2" );
                }

                getSemantico().pushInstrucaoNaPilha(
                        "%ptr2" );

                getSemantico().pushPonteiroNaPilha(
                        "%ptr2" );
                /*
                getSemantico().getCurrentMethod().addCommad(
                        "%intvar2 = add i32 0," );

                switch ( getSemantico().popTipoDaPilha() ) {

                        case Constantes.STACK_TYPE_INT:
                                getSemantico().getCurrentMethod().getCurrentCommand().
                                        setRightSide( String.valueOf(
                                                getSemantico().popInteiroDaPilha() ) );
                                break;

                        case Constantes.STACK_TYPE_VAR:
                                getSemantico().getCurrentMethod().getCurrentCommand().
                                        setRightSide( getSemantico().
                                                popInstrucaoDaPilha() );
                                break;
                }

                getSemantico().getCurrentMethod().getCurrentCommand().
                        setOperation( "" );

                getSemantico().pushInstrucaoNaPilha( "%intvar2");
                */
        }
}

/**
```

```
 * Arquivo AcaoSemantica162.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica162 extends BaseAcaoSemantica {

	public AcaoSemantica162() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		if ( !getSemantico().temVarNaPilha( "%intvar3" ) ) {

			getSemantico().getCurrentMethod().addCommad(
				"%intvar3 = alloca i32" );

			getSemantico().getCurrentMethod().addCommad(
				"%ptr3 = getelementptr i32* %intvar3" );

			getSemantico().pushVarParaPilha( "%intvar3" );
		}

		switch ( getSemantico().popTipoDaPilha() ) {

			case Constantes.STACK_TYPE_INT:
				getSemantico().getCurrentMethod().
					addCommad( "store i32 "+
						getSemantico().popInteiroDaPilha() +
							", i32* %ptr3");
				break;

			case Constantes.STACK_TYPE_VAR:

				String operando = getSemantico().
					popInstrucaoDaPilha();

				if ( getSemantico().pilhaTemPonteiro(
					operando ) ) {

					String tempvar = operando+"temp"+getSemantico().
						getCurrentTempVar();

					getSemantico().getCurrentMethod().
						addCommad( tempvar+" = load i32* "+operando );
					operando = tempvar;
				}

				getSemantico().getCurrentMethod().
					addCommad( "store i32 "+ operando +
						", i32* %ptr3");
				break;

			default:
				break;
```

```
                }

                if ( !getSemantico().temVarNaPilha( "%intvar3" ) ) {

                        getSemantico().getCurrentMethod().addCommad(
                                "%tempint3 = load i32* %ptr3" );
                }

                getSemantico().pushInstrucaoNaPilha(
                        "%ptr3" );

                getSemantico().pushPonteiroNaPilha(
                        "%ptr3" );
                /*
                getSemantico().getCurrentMethod().addCommad(
                        "%intvar3 = add i32 0," );

                switch ( getSemantico().popTipoDaPilha() ) {

                        case Constantes.STACK_TYPE_INT:
                                getSemantico().getCurrentMethod().getCurrentCommand().
                                        setRightSide( String.valueOf(
                                                getSemantico().popInteiroDaPilha() ) );
                                break;

                        case Constantes.STACK_TYPE_VAR:
                                getSemantico().getCurrentMethod().getCurrentCommand().
                                        setRightSide( getSemantico().
                                                popInstrucaoDaPilha() );
                                break;
                }

                getSemantico().getCurrentMethod().getCurrentCommand().
                        setOperation( "" );

                getSemantico().pushInstrucaoNaPilha( "%intvar3");
                */
        }
}

/**
 * Arquivo AcaoSemantica163.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica163 extends BaseAcaoSemantica {

        public AcaoSemantica163() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {


        }
}
```

```
/**
 * Arquivo AcaoSemantica164.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica164 extends BaseAcaoSemantica {

	public AcaoSemantica164() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica165.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica165 extends BaseAcaoSemantica {

	public AcaoSemantica165() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica166.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica166 extends BaseAcaoSemantica {

	public AcaoSemantica166() {

		super();
	}
```

```
	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica167.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica167 extends BaseAcaoSemantica {

	public AcaoSemantica167() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica168.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica168 extends BaseAcaoSemantica {

	public AcaoSemantica168() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica169.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica169 extends BaseAcaoSemantica {
```

```
	public AcaoSemantica169() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica17.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica17 extends BaseAcaoSemantica {

	public AcaoSemantica17() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica170.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica170 extends BaseAcaoSemantica {

	public AcaoSemantica170() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica171.java
```

```
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica171 extends BaseAcaoSemantica {

	public AcaoSemantica171() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica172.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica172 extends BaseAcaoSemantica {

	public AcaoSemantica172() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica173.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica173 extends BaseAcaoSemantica {

	public AcaoSemantica173() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
```

```
	}
}


/**
 * Arquivo AcaoSemantica174.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica174 extends BaseAcaoSemantica {

	public AcaoSemantica174() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica175.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica175 extends BaseAcaoSemantica {

	public AcaoSemantica175() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica176.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica176 extends BaseAcaoSemantica {

	public AcaoSemantica176() {

		super();
```

```
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica177.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica177 extends BaseAcaoSemantica {

	public AcaoSemantica177() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica178.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica178 extends BaseAcaoSemantica {

	public AcaoSemantica178() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica179.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
```

```
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica179 extends BaseAcaoSemantica {

	public AcaoSemantica179() {

		super();
	}

	private String getOpDaPilha () {

		switch ( getSemantico().popTipoDaPilha() ) {

		case Constantes.STACK_TYPE_INT:
			return ""+getSemantico().popInteiroDaPilha();

		case Constantes.STACK_TYPE_VAR:

			String operando = getSemantico().
				popInstrucaoDaPilha();

			if ( getSemantico().pilhaTemPonteiro(
				operando ) ) {

				String tempvar = operando+"temp"+getSemantico().
					getCurrentTempVar();

				getSemantico().getCurrentMethod().
					addCommad( tempvar+" = load i32* "+operando );
				operando = tempvar;
			}

			//index += operando;
			return operando;
		}

		return "";
	}

	@Override
	public void executaAcao( Token token ) {

		String value = getOpDaPilha();
		String index = getOpDaPilha();

		String operand1 =
			getSemantico().popInstrucaoDaPilha();

		String ptr = "%arrayptr"+ getSemantico().
			getCurrentTempPointers();

		getSemantico().getCurrentMethod().
			addCommad( ptr +"= getelementptr i32* "+
				operand1 +", i32 "+ index );

		getSemantico().getCurrentMethod().
			addCommad( "store i32 "+ value +", i32* "+
				ptr );
	}
}
```

```
/**
 * Arquivo AcaoSemantica18.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator.MethodParameter;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator.MethodSignature;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;
import inf.ufsc.br.tcc.cantu.parser.utils.Util;

public class AcaoSemantica18 extends BaseAcaoSemantica {

	public AcaoSemantica18() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		String opcode = null;
		StringBuilder builder = new StringBuilder();

		MethodSignature method =
			getSemantico().getMethods().
				get( token.getLexeme() );

		boolean temRetorno =
			( ( method.getReturnType() != null )
				&& ( !method.getReturnType().equals( "" ) )
					&& ( !method.getReturnType().equals( "void" ) ) );

		String methodName = method.getName();

		String returnType = Util.jvmType2LlvmType(
			method.getReturnType() );

		if( temRetorno ) {

			opcode =
				"%retorno"+ getSemantico().getCurrentTempVar();

			builder.append( opcode +" = " );
		}

		builder.append( "tail call " );
		builder.append( returnType );
		builder.append( " @" );
		builder.append( methodName );

		String leftSide = builder.toString();
		getSemantico().getCurrentMethod().
			addCommad( leftSide );

		builder.delete( 0, builder.length() );

		int i = 0;
```

```
                builder.append( "(" );
                for ( MethodParameter param : method.
                                getParameters().values() ) {

                        builder.append( Util.jvmType2LlvmType( param.getType() ) );
                        builder.append( " " );

                        if ( param.getType().equals( "int" ) ) {

                                int type = getSemantico().rpopTipoDaPilha();

                                switch ( type ) {

                                        case Constantes.STACK_TYPE_INT:
                                                builder.append( getSemantico().
                                                        rpopInteiroDaPilha() );
                                                break;

                                        case Constantes.STACK_TYPE_VAR:
                                                builder.append( getSemantico().
                                                        popInstrucaoDaPilha() );
                                                break;
                                }
                        }

                        if ( ++i < method.getParameters().size() ) {

                                builder.append( ", " );
                        }
                }
                builder.append( ")" );

                String rightSide = builder.toString();
                getSemantico().getCurrentMethod().
                        getCurrentCommand().setRightSide(
                                rightSide );

                if( temRetorno ) {

                        getSemantico().
                                pushInstrucaoNaPilha( opcode );

                        getSemantico().pushTipoParaPilha(
                                Constantes.STACK_TYPE_VAR );
                }
        }
}


/**
 * Arquivo AcaoSemantica180.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica180 extends BaseAcaoSemantica {

        public AcaoSemantica180() {
```

```
		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica181.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica181 extends BaseAcaoSemantica {

	public AcaoSemantica181() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica182.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica182 extends BaseAcaoSemantica {

	public AcaoSemantica182() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica183.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;
```

```
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica183 extends BaseAcaoSemantica {

	public AcaoSemantica183() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica184.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica184 extends BaseAcaoSemantica {

	public AcaoSemantica184() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica185.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica185 extends BaseAcaoSemantica {

	public AcaoSemantica185() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}
```

```
/**
 * Arquivo AcaoSemantica186.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica186 extends BaseAcaoSemantica {

	public AcaoSemantica186() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica187.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica187 extends BaseAcaoSemantica {

	public AcaoSemantica187() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica188.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica188 extends BaseAcaoSemantica {

	public AcaoSemantica188() {

		super();
	}

	@Override
```

```
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica189.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica189 extends BaseAcaoSemantica {

	public AcaoSemantica189() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica190.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica190 extends BaseAcaoSemantica {

	public AcaoSemantica190() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica191.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica191 extends BaseAcaoSemantica {
```

```
	public AcaoSemantica191() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica192.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica192 extends BaseAcaoSemantica {

	public AcaoSemantica192() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica193.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica193 extends BaseAcaoSemantica {

	public AcaoSemantica193() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica194.java
 */
```

```
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica194 extends BaseAcaoSemantica {

	public AcaoSemantica194() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica195.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica195 extends BaseAcaoSemantica {

	public AcaoSemantica195() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica196.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica196 extends BaseAcaoSemantica {

	public AcaoSemantica196() {

		super();
	}

	private String getOperand() {

		switch ( getSemantico().popTipoDaPilha() ) {
```

```
                        case Constantes.STACK_TYPE_INT:
                                return ""+getSemantico().popInteiroDaPilha();

                        case Constantes.STACK_TYPE_VAR:

                                String operand = getSemantico().popInstrucaoDaPilha();
                                if ( getSemantico().pilhaTemPonteiro(
                                                operand ) ) {

                                        String tempvar = operand+"temp"+getSemantico().
                                                getCurrentTempVar();

                                        getSemantico().getCurrentMethod().
                                                addCommad( tempvar+" = load i32* "+operand );
                                        operand = tempvar;
                                }

                                return operand;
                        }

                        return "";
                }

                @Override
                public void executaAcao( Token token ) {

                        String operand1;
                        String operand2;
                        String temp = "%temp"+
                                getSemantico().getCurrentTempVar();

                        operand1 = getOperand();
                        operand2 = getOperand();

                        getSemantico().getCurrentMethod().
                                addCommad( temp +" = add i32" );

                        getSemantico().getCurrentMethod().
                                getCurrentCommand().setRightSide( operand2 +","+ operand1 );

                        getSemantico().getCurrentMethod().
                                getCurrentCommand().setOperation( " " );

                        getSemantico().pushInstrucaoNaPilha( temp );

                        getSemantico().pushTipoParaPilha(
                                Constantes.STACK_TYPE_VAR );
                }
}


/**
 * Arquivo AcaoSemantica197.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica197 extends BaseAcaoSemantica {
```

```
	public AcaoSemantica197() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica198.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica198 extends BaseAcaoSemantica {

	public AcaoSemantica198() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica199.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica199 extends BaseAcaoSemantica {

	public AcaoSemantica199() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica2.java
 */
```

```
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica2 extends BaseAcaoSemantica {

	public AcaoSemantica2() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
	}
}


/**
 * Arquivo AcaoSemantica200.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica200 extends BaseAcaoSemantica {

	public AcaoSemantica200() {

		super();
	}

	private String getOperand() {

		switch ( getSemantico().popTipoDaPilha() ) {

			case Constantes.STACK_TYPE_INT:
				return ""+getSemantico().popInteiroDaPilha();

			case Constantes.STACK_TYPE_VAR:

				String operand = getSemantico().popInstrucaoDaPilha();
				if ( getSemantico().pilhaTemPonteiro(
						operand ) ) {

					String tempvar = operand+"temp"+getSemantico().
						getCurrentTempVar();

					getSemantico().getCurrentMethod().
						addCommad( tempvar+" = load i32* "+operand );
					operand = tempvar;
				}

				return operand;
		}

		return "";
	}

	@Override
```

```
	public void executaAcao( Token token ) {

		String operand1;
		String operand2;
		String temp = "%temp"+
			getSemantico().getCurrentTempVar();

		operand1 = getOperand();
		operand2 = getOperand();

		getSemantico().getCurrentMethod().
			addCommad( temp +" = sub i32" );

		getSemantico().getCurrentMethod().
			getCurrentCommand().setRightSide( operand2 +","+ operand1 );

		getSemantico().getCurrentMethod().
			getCurrentCommand().setOperation( " " );

		getSemantico().pushInstrucaoNaPilha( temp );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_VAR );
	}
}


/**
 * Arquivo AcaoSemantica201.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica201 extends BaseAcaoSemantica {

	public AcaoSemantica201() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica202.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica202 extends BaseAcaoSemantica {

	public AcaoSemantica202() {
```

```
		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica203.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica203 extends BaseAcaoSemantica {

	public AcaoSemantica203() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica204.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica204 extends BaseAcaoSemantica {

	public AcaoSemantica204() {

		super();
	}

	private String getOperand() {

		switch ( getSemantico().popTipoDaPilha() ) {

			case Constantes.STACK_TYPE_INT:
				return ""+getSemantico().popInteiroDaPilha();

			case Constantes.STACK_TYPE_VAR:

				String operand = getSemantico().popInstrucaoDaPilha();
				if ( getSemantico().pilhaTemPonteiro(
						operand ) ) {
```

```
                                        String tempvar = operand+"temp"+getSemantico().
                                                getCurrentTempVar();

                                        getSemantico().getCurrentMethod().
                                                addCommad( tempvar+" = load i32* "+operand );
                                        operand = tempvar;
                                }

                                return operand;
                        }

                        return "";
                }

                @Override
                public void executaAcao( Token token ) {

                        String operand1;
                        String operand2;
                        String temp = "%temp"+
                                getSemantico().getCurrentTempVar();

                        operand1 = getOperand();
                        operand2 = getOperand();

                        getSemantico().getCurrentMethod().
                                addCommad( temp +" = mul i32" );

                        getSemantico().getCurrentMethod().
                                getCurrentCommand().setRightSide( operand2 +","+ operand1 );

                        getSemantico().getCurrentMethod().
                                getCurrentCommand().setOperation( " " );

                        getSemantico().pushInstrucaoNaPilha( temp );

                        getSemantico().pushTipoParaPilha(
                                Constantes.STACK_TYPE_VAR );
                }
}

/**
 * Arquivo AcaoSemantica205.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica205 extends BaseAcaoSemantica {

        public AcaoSemantica205() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {
```

```
	}
}


/**
 * Arquivo AcaoSemantica206.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica206 extends BaseAcaoSemantica {

	public AcaoSemantica206() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica207.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica207 extends BaseAcaoSemantica {

	public AcaoSemantica207() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica208.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica208 extends BaseAcaoSemantica {

	public AcaoSemantica208() {
```

```
		super();
	}

	private String getOperand() {

		switch ( getSemantico().popTipoDaPilha() ) {

			case Constantes.STACK_TYPE_INT:
				return ""+getSemantico().popInteiroDaPilha();

			case Constantes.STACK_TYPE_VAR:

				String operand = getSemantico().popInstrucaoDaPilha();
				if ( getSemantico().pilhaTemPonteiro(
						operand ) ) {

					String tempvar = operand+"temp"+getSemantico().
						getCurrentTempVar();

					getSemantico().getCurrentMethod().
						addCommad( tempvar+" = load i32* "+operand );
					operand = tempvar;
				}

				return operand;
		}

		return "";
	}

	@Override
	public void executaAcao( Token token ) {

		String operand1;
		String operand2;
		String temp = "%temp"+
			getSemantico().getCurrentTempVar();

		operand1 = getOperand();
		operand2 = getOperand();

		getSemantico().getCurrentMethod().
			addCommad( temp +" = sdiv i32" );

		getSemantico().getCurrentMethod().
			getCurrentCommand().setRightSide( operand2 +","+ operand1 );

		getSemantico().getCurrentMethod().
			getCurrentCommand().setOperation( " " );

		getSemantico().pushInstrucaoNaPilha( temp );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_VAR );
	}
}

/**
 * Arquivo AcaoSemantica209.java
 */
```

```
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica209 extends BaseAcaoSemantica {

	public AcaoSemantica209() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica21.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica21 extends BaseAcaoSemantica {

	public AcaoSemantica21() {

		super();
	}

	/**
	 * J? temos um m?todo,
	 * vamos criar um par?metro para
	 * depois setar o nome e o tipo.
	 */
	@Override
	public void executaAcao( Token token ) {

		getSemantico().getCurrentMethod().
			addEmptyParameter();
	}
}

/**
 * Arquivo AcaoSemantica210.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica210 extends BaseAcaoSemantica {

	public AcaoSemantica210() {

		super();
	}
```

```
	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica211.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica211 extends BaseAcaoSemantica {

	public AcaoSemantica211() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica212.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica212 extends BaseAcaoSemantica {

	public AcaoSemantica212() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica213.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica213 extends BaseAcaoSemantica {
```

```
	public AcaoSemantica213() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica214.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica214 extends BaseAcaoSemantica {

	public AcaoSemantica214() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica215.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica215 extends BaseAcaoSemantica {

	public AcaoSemantica215() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica216.java
```

```
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica216 extends BaseAcaoSemantica {

	public AcaoSemantica216() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica217.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica217 extends BaseAcaoSemantica {

	public AcaoSemantica217() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica218.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica218 extends BaseAcaoSemantica {

	public AcaoSemantica218() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
```

```
	}
}


/**
 * Arquivo AcaoSemantica219.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica219 extends BaseAcaoSemantica {

	public AcaoSemantica219() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica22.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica22 extends BaseAcaoSemantica {

	public AcaoSemantica22() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		getSemantico().setUsingTypeForReturn( true );
	}
}

/**
 * Arquivo AcaoSemantica220.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica220 extends BaseAcaoSemantica {

	public AcaoSemantica220() {

		super();
	}
```

```
	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica221.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica221 extends BaseAcaoSemantica {

	public AcaoSemantica221() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica222.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica222 extends BaseAcaoSemantica {

	public AcaoSemantica222() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica223.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
```

```
public class AcaoSemantica223 extends BaseAcaoSemantica {

	public AcaoSemantica223() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica224.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica224 extends BaseAcaoSemantica {

	public AcaoSemantica224() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica225.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica225 extends BaseAcaoSemantica {

	public AcaoSemantica225() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
```

```
 * Arquivo AcaoSemantica226.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica226 extends BaseAcaoSemantica {

	public AcaoSemantica226() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica227.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica227 extends BaseAcaoSemantica {

	public AcaoSemantica227() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica228.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica228 extends BaseAcaoSemantica {

	public AcaoSemantica228() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
```

```
	}
}


/**
 * Arquivo AcaoSemantica229.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica229 extends BaseAcaoSemantica {

	public AcaoSemantica229() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica23.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator.MethodParameter;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica23 extends BaseAcaoSemantica {

	public AcaoSemantica23() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		for ( MethodParameter param : getSemantico().getCurrentMethod().
			getParameters().values() ) {

			if ( param.getType().equals( Constantes.JVM_TYPE_INT ) ) {

				getSemantico().getCurrentMethod().
					addCommad( "%intvar"+param.getPosition() +
						" = alloca i32" );

				getSemantico().getCurrentMethod().
					addCommad( "%ptr"+param.getPosition() +
						" = getelementptr i32* %intvar"+ param.getPosition() );

				getSemantico().getCurrentMethod().
```

```
                                        addCommad( "store i32 "+ param.getSpecificName() +
                                                ", i32* %ptr"+param.getPosition() );

                                getSemantico().getCurrentMethod().addCommad(
                                        "%tempint"+ param.getPosition() +" = load i32* %ptr"+
                                                param.getPosition());

                                getSemantico().pushInstrucaoNaPilha(
                                        "%tempint"+ param.getPosition() );
                        }
                }
        }
}

/**
 * Arquivo AcaoSemantica230.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica230 extends BaseAcaoSemantica {

        public AcaoSemantica230() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {


        }
}


/**
 * Arquivo AcaoSemantica231.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica231 extends BaseAcaoSemantica {

        public AcaoSemantica231() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {


        }
}


/**
 * Arquivo AcaoSemantica232.java
```

```
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica232 extends BaseAcaoSemantica {

	public AcaoSemantica232() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica233.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica233 extends BaseAcaoSemantica {

	public AcaoSemantica233() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica234.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica234 extends BaseAcaoSemantica {

	public AcaoSemantica234() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
```

```
	}
}


/**
 * Arquivo AcaoSemantica235.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica235 extends BaseAcaoSemantica {

	public AcaoSemantica235() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica236.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica236 extends BaseAcaoSemantica {

	public AcaoSemantica236() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica237.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica237 extends BaseAcaoSemantica {

	public AcaoSemantica237() {

		super();
```

```
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica238.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica238 extends BaseAcaoSemantica {

	public AcaoSemantica238() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica239.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica239 extends BaseAcaoSemantica {

	public AcaoSemantica239() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica24.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
```

```
public class AcaoSemantica24 extends BaseAcaoSemantica {

	public AcaoSemantica24() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		int label = Integer.parseInt(
			token.getLexeme() );

		getSemantico().getCurrentMethod().
			addCommad( label +":" );

		getSemantico().getCurrentMethod().
			getCurrentCommand().setShow( false );
	}
}

/**
 * Arquivo AcaoSemantica240.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica240 extends BaseAcaoSemantica {

	public AcaoSemantica240() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica241.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica241 extends BaseAcaoSemantica {

	public AcaoSemantica241() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
```

```
	}
}


/**
 * Arquivo AcaoSemantica242.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica242 extends BaseAcaoSemantica {

	public AcaoSemantica242() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica243.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica243 extends BaseAcaoSemantica {

	public AcaoSemantica243() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica244.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica244 extends BaseAcaoSemantica {

	public AcaoSemantica244() {
```

```
		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica245.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica245 extends BaseAcaoSemantica {

	public AcaoSemantica245() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica246.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica246 extends BaseAcaoSemantica {

	public AcaoSemantica246() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica247.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;
```

```
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica247 extends BaseAcaoSemantica {

	public AcaoSemantica247() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica248.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica248 extends BaseAcaoSemantica {

	public AcaoSemantica248() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica249.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica249 extends BaseAcaoSemantica {

	public AcaoSemantica249() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}
```

```
/**
 * Arquivo AcaoSemantica25.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica25 extends BaseAcaoSemantica {

	public AcaoSemantica25() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

	}
}

/**
 * Arquivo AcaoSemantica250.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica250 extends BaseAcaoSemantica {

	public AcaoSemantica250() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica251.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica251 extends BaseAcaoSemantica {

	public AcaoSemantica251() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
```

```
	}
}


/**
 * Arquivo AcaoSemantica252.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica252 extends BaseAcaoSemantica {

	public AcaoSemantica252() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica253.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica253 extends BaseAcaoSemantica {

	public AcaoSemantica253() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica254.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica254 extends BaseAcaoSemantica {

	public AcaoSemantica254() {
```

```
		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica255.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica255 extends BaseAcaoSemantica {

	public AcaoSemantica255() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica256.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica256 extends BaseAcaoSemantica {

	public AcaoSemantica256() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica257.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;
```

```
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica257 extends BaseAcaoSemantica {

	public AcaoSemantica257() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica258.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica258 extends BaseAcaoSemantica {

	public AcaoSemantica258() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica259.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica259 extends BaseAcaoSemantica {

	public AcaoSemantica259() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}
```

```
/**
 * Arquivo AcaoSemantica26.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica26 extends BaseAcaoSemantica {

	public AcaoSemantica26() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		int label = Integer.parseInt(
			token.getLexeme() );

		getSemantico().
			pushLabelNaPilha( getSemantico().
				getCurrentMethod(), label );

		if ( getSemantico().haveUnconditionalBranch() ) {

			getSemantico().removeUnconditionalBranch();

			getSemantico().getCurrentMethod().
				addCommad( "br label %Label"+ label );

		} else {

			String leftSide = "br i1 "+
			getSemantico().popInstrucaoDaPilha() +", ";

			getSemantico().getCurrentMethod().
				addCommad( leftSide );

			int numseq = getSemantico().getCurrentTempSeqs();
			String seq = "Seq"+numseq;

			getSemantico().getCurrentMethod().
				getCurrentCommand().setRightSide(
					"label %Label"+ label +", label %"+seq );

			getSemantico().getCurrentMethod().
				getCurrentCommand().setOperation( "" );

			getSemantico().getCurrentMethod().
				addCommad( seq+":" );
			getSemantico().getCurrentMethod().
				getCurrentCommand().setLabel( true );
		}
	}
}

/**
```

```
 * Arquivo AcaoSemantica260.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica260 extends BaseAcaoSemantica {

	public AcaoSemantica260() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica261.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica261 extends BaseAcaoSemantica {

	public AcaoSemantica261() {

		super();
	}

	private String getOperand() {

		switch ( getSemantico().popTipoDaPilha() ) {

			case Constantes.STACK_TYPE_INT:
				return ""+getSemantico().popInteiroDaPilha();

			case Constantes.STACK_TYPE_VAR:
				return getSemantico().popInstrucaoDaPilha();
		}

		return "";
	}

	@Override
	public void executaAcao( Token token ) {

		String o1 = getOperand();

		if ( getSemantico().pilhaTemPonteiro( o1 ) ) {

			String tempvar = o1+"temp1"+getSemantico().
				getCurrentTempVar();
```

```
                        getSemantico().getCurrentMethod().
                                addCommad( tempvar+" = load i32* "+o1 );
                        o1 = tempvar;
                }

                String o2 = getOperand();

                if ( getSemantico().pilhaTemPonteiro( o2 ) ) {

                        String tempvar = o2+"temp2"+getSemantico().
                                getCurrentTempVar();

                        getSemantico().getCurrentMethod().
                                addCommad( tempvar+" = load i32* "+o2 );
                        o2 = tempvar;
                }

                String rightSide = o2 +", "+ o1;

                String var = "%cond"+
                        getSemantico().getCurrentTempVar();

                String leftSide = var +" = icmp slt i32 ";

                getSemantico().getCurrentMethod().
                        addCommad( leftSide );

                getSemantico().getCurrentMethod().
                        getCurrentCommand().setRightSide( rightSide );

                getSemantico().getCurrentMethod().
                        getCurrentCommand().setOperation( "" );

                getSemantico().pushInstrucaoNaPilha( var );
        }
}

/**
 * Arquivo AcaoSemantica262.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica262 extends BaseAcaoSemantica {

        public AcaoSemantica262() {

                super();
        }


        private String getOperand() {

                switch ( getSemantico().popTipoDaPilha() ) {

                        case Constantes.STACK_TYPE_INT:
                                return ""+getSemantico().popInteiroDaPilha();
```

```
                        case Constantes.STACK_TYPE_VAR:
                                return getSemantico().popInstrucaoDaPilha();
                }

                return "";
        }

        @Override
        public void executaAcao( Token token ) {

                String o1 = getOperand();

                if ( getSemantico().pilhaTemPonteiro( o1 ) ) {

                        String tempvar = o1+"temp1"+getSemantico().
                                getCurrentTempVar();

                        getSemantico().getCurrentMethod().
                                addCommad( tempvar+" = load i32* "+o1 );
                        o1 = tempvar;
                }

                String o2 = getOperand();

                if ( getSemantico().pilhaTemPonteiro( o2 ) ) {

                        String tempvar = o2+"temp2"+getSemantico().
                                getCurrentTempVar();

                        getSemantico().getCurrentMethod().
                                addCommad( tempvar+" = load i32* "+o2 );
                        o2 = tempvar;
                }

                String rightSide = o2 +", "+ o1;

                String var = "%cond"+
                        getSemantico().getCurrentTempVar();

                String leftSide = var +" = icmp sge i32 ";

                getSemantico().getCurrentMethod().
                        addCommad( leftSide );

                getSemantico().getCurrentMethod().
                        getCurrentCommand().setRightSide( rightSide );

                getSemantico().getCurrentMethod().
                        getCurrentCommand().setOperation( "" );

                getSemantico().pushInstrucaoNaPilha( var );
        }
}

/**
 * Arquivo AcaoSemantica263.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
```

```
public class AcaoSemantica263 extends BaseAcaoSemantica {

	public AcaoSemantica263() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica264.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica264 extends BaseAcaoSemantica {

	public AcaoSemantica264() {

		super();
	}

	private String getOperand() {

		switch ( getSemantico().popTipoDaPilha() ) {

			case Constantes.STACK_TYPE_INT:
				return ""+getSemantico().popInteiroDaPilha();

			case Constantes.STACK_TYPE_VAR:
				return getSemantico().popInstrucaoDaPilha();
		}

		return "";
	}

	@Override
	public void executaAcao( Token token ) {

		String o1 = getOperand();

		if ( getSemantico().pilhaTemPonteiro( o1 ) ) {

			String tempvar = o1+"temp1"+getSemantico().
				getCurrentTempVar();

			getSemantico().getCurrentMethod().
				addCommad( tempvar+" = load i32* "+o1 );
			o1 = tempvar;
		}
```

```
		String o2 = getOperand();

		if ( getSemantico().pilhaTemPonteiro( o2 ) ) {

			String tempvar = o2+"temp2"+getSemantico().
				getCurrentTempVar();

			getSemantico().getCurrentMethod().
				addCommad( tempvar+" = load i32* "+o2 );
			o2 = tempvar;
		}

		String rightSide = o2 +", "+ o1;

		String var = "%cond"+
			getSemantico().getCurrentTempVar();

		String leftSide = var +" = icmp sle i32 ";

		getSemantico().getCurrentMethod().
			addCommad( leftSide );

		getSemantico().getCurrentMethod().
			getCurrentCommand().setRightSide( rightSide );

		getSemantico().getCurrentMethod().
			getCurrentCommand().setOperation( "" );

		getSemantico().pushInstrucaoNaPilha( var );
	}
}


/**
 * Arquivo AcaoSemantica265.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica265 extends BaseAcaoSemantica {

	public AcaoSemantica265() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica266.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;
```

```
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica266 extends BaseAcaoSemantica {

	public AcaoSemantica266() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica267.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica267 extends BaseAcaoSemantica {

	public AcaoSemantica267() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		getSemantico().addUnconditionalBranch();
	}
}

/**
 * Arquivo AcaoSemantica268.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica268 extends BaseAcaoSemantica {

	public AcaoSemantica268() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}
```

```
/**
 * Arquivo AcaoSemantica269.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica269 extends BaseAcaoSemantica {

	public AcaoSemantica269() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica27.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica27 extends BaseAcaoSemantica {

	public AcaoSemantica27() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		getSemantico().pushInstrucaoNaPilha(
			"%ptr"+ token.getLexeme() );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_VAR );
	}
}

/**
 * Arquivo AcaoSemantica270.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica270 extends BaseAcaoSemantica {

	public AcaoSemantica270() {

		super();
```

```
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica271.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica271 extends BaseAcaoSemantica {

	public AcaoSemantica271() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica272.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica272 extends BaseAcaoSemantica {

	public AcaoSemantica272() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		String rightSide = getSemantico().
			popInstrucaoDaPilha();

		if ( getSemantico().pilhaTemPonteiro(
				rightSide ) ) {

			String tempvar = rightSide+"temp"+getSemantico().
				getCurrentTempVar();

			getSemantico().getCurrentMethod().
				addCommad( tempvar+" = load i32* "+rightSide );
```

```
                        rightSide = tempvar;
                }

                getSemantico().getCurrentMethod().
                        addCommad( "ret i32" );

                getSemantico().getCurrentMethod().
                        getCurrentCommand().setRightSide( rightSide );

                getSemantico().getCurrentMethod().
                        getCurrentCommand().setOperation( " " );

                getSemantico().clearTempVars();
        }
}

/**
 * Arquivo AcaoSemantica273.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica273 extends BaseAcaoSemantica {

        public AcaoSemantica273() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {


        }
}


/**
 * Arquivo AcaoSemantica274.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica274 extends BaseAcaoSemantica {

        public AcaoSemantica274() {

                super();
        }

        @Override
        public void executaAcao( Token token ) {


        }
}
```

```
/**
 * Arquivo AcaoSemantica275.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica275 extends BaseAcaoSemantica {

	public AcaoSemantica275() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica276.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica276 extends BaseAcaoSemantica {

	public AcaoSemantica276() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica277.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica277 extends BaseAcaoSemantica {

	public AcaoSemantica277() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
```

```
	}
}


/**
 * Arquivo AcaoSemantica278.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica278 extends BaseAcaoSemantica {

	public AcaoSemantica278() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica279.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica279 extends BaseAcaoSemantica {

	public AcaoSemantica279() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica28.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica28 extends BaseAcaoSemantica {

	public AcaoSemantica28() {
```

```
		super();
	}

	@Override
	public void executaAcao( Token token ) {

		String incremento = token.getLexeme();

		String var = getSemantico().
			popInstrucaoDaPilha();

		String tempvar = var+"temp"+ getSemantico().
			getCurrentTempVar();
		String tempvar1 = var+"temp"+getSemantico().
			getCurrentTempVar();

		getSemantico().getCurrentMethod().
			addCommad( tempvar +" = load i32* "+ var );

		getSemantico().getCurrentMethod().
			addCommad( tempvar1 +" = add i32 "+ tempvar +
				", "+ incremento );

		getSemantico().getCurrentMethod().
			addCommad( "store i32 "+ tempvar1 +", i32* "+
				var );
	}
}

/**
 * Arquivo AcaoSemantica280.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica280 extends BaseAcaoSemantica {

	public AcaoSemantica280() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica281.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica281 extends BaseAcaoSemantica {
```

```
	public AcaoSemantica281() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica282.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica282 extends BaseAcaoSemantica {

	public AcaoSemantica282() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica283.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica283 extends BaseAcaoSemantica {

	public AcaoSemantica283() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica284.java
```

```
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica284 extends BaseAcaoSemantica {

	public AcaoSemantica284() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

	}
}

/**
 * Arquivo AcaoSemantica285.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica285 extends BaseAcaoSemantica {

	public AcaoSemantica285() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica286.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica286 extends BaseAcaoSemantica {

	public AcaoSemantica286() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}
```

```
/**
 * Arquivo AcaoSemantica287.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica287 extends BaseAcaoSemantica {

	public AcaoSemantica287() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		int size =
			getSemantico().popInteiroDaPilha();

		String operand1 = "alloca i32, i32 "+ size;

		getSemantico().pushArrayLenghNaPilha( size );

		String temp = "%array"+
			getSemantico().getCurrentTempArray();

		getSemantico().getCurrentMethod().
			addCommad( temp +" = " );

		getSemantico().getCurrentMethod().
			getCurrentCommand().setRightSide( operand1 +"");

		getSemantico().getCurrentMethod().
			getCurrentCommand().setOperation( " " );

		getSemantico().pushInstrucaoNaPilha( temp );
		getSemantico().pushArrayNaPilha( temp );
	}
}

/**
 * Arquivo AcaoSemantica288.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica288 extends BaseAcaoSemantica {

	public AcaoSemantica288() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {
```

```
	}
}


/**
 * Arquivo AcaoSemantica289.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;

public class AcaoSemantica289 extends BaseAcaoSemantica {

	public AcaoSemantica289() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		getSemantico().popInstrucaoDaPilha();

		getSemantico().pushInteiroNaPilha(
			getSemantico().popArrayLenghDaPilha() );

		getSemantico().pushTipoParaPilha(
			Constantes.STACK_TYPE_INT );
	}
}

/**
 * Arquivo AcaoSemantica290.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica290 extends BaseAcaoSemantica {

	public AcaoSemantica290() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica291.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;
```

```
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica291 extends BaseAcaoSemantica {

	public AcaoSemantica291() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica292.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica292 extends BaseAcaoSemantica {

	public AcaoSemantica292() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica293.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica293 extends BaseAcaoSemantica {

	public AcaoSemantica293() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}
```

```
/**
 * Arquivo AcaoSemantica294.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica294 extends BaseAcaoSemantica {

	public AcaoSemantica294() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica295.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica295 extends BaseAcaoSemantica {

	public AcaoSemantica295() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica296.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica296 extends BaseAcaoSemantica {

	public AcaoSemantica296() {

		super();
	}

	@Override
```

```
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica297.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica297 extends BaseAcaoSemantica {

	public AcaoSemantica297() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica298.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica298 extends BaseAcaoSemantica {

	public AcaoSemantica298() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica299.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica299 extends BaseAcaoSemantica {
```

```
	public AcaoSemantica299() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica3.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica3 extends BaseAcaoSemantica {

	public AcaoSemantica3() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica300.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica300 extends BaseAcaoSemantica {

	public AcaoSemantica300() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica301.java
 */
```

```
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica301 extends BaseAcaoSemantica {

	public AcaoSemantica301() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica302.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica302 extends BaseAcaoSemantica {

	public AcaoSemantica302() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {


	}
}


/**
 * Arquivo AcaoSemantica303.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica303 extends BaseAcaoSemantica {

	public AcaoSemantica303() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		System.out.println("Token: "+token);
	}
```

```
}


/**
 * Arquivo AcaoSemantica4.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator.Command;

public class AcaoSemantica4 extends BaseAcaoSemantica {

	public AcaoSemantica4() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		for ( Command command : getSemantico().
			getCurrentMethod().getCommands().values() ) {

			if ( !command.isShow() ) {

				int index = command.getLeftSide().
					indexOf( ':' );

				if ( index >= 0 ) {

					int label = Integer.parseInt(
						command.getLeftSide().substring(
							0, index ) );

					if ( getSemantico().pilhaTemLabel(
						getSemantico().getCurrentMethod(),label ) ) {

						String left =
							command.getLeftSide();

						command.setLeftSide( "Label"+ left );
						command.setShow( true );
						command.setLabel( true );
					}
				}
			}
		}
	}
}


/**
 * Arquivo AcaoSemantica5.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator.MethodSignature;
```

```
public class AcaoSemantica5 extends BaseAcaoSemantica {

	public AcaoSemantica5() {

		super();
	}

	@Override
	public void executaAcao( Token token ) {

		getSemantico().addMethod(
			new MethodSignature( token.getLexeme() ) );
	}
}


/**
 * Arquivo AcaoSemantica6.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica6 extends BaseAcaoSemantica {

	public AcaoSemantica6() {

		super();
	}

	/**
    * J? sabemos o m?todo
    * e temos uma par?metro atual.
	 */
	@Override
	public void executaAcao( Token token ) {

		if ( getSemantico().isUsingTypeForReturn() ) {

			getSemantico().getCurrentMethod().setReturnType(
				token.getLexeme() );

			getSemantico().setUsingTypeForReturn( false );

		} else {

			getSemantico().getCurrentMethod().
				getCurrentParameter().setType( token.getLexeme() );
		}
	}
}

/**
 * Arquivo AcaoSemantica7.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica7 extends BaseAcaoSemantica {
```

```
	public AcaoSemantica7() {

		super();
	}

	/**
    * J? sabemos o m?todo
    * e temos uma par?metro atual.
	 */
	@Override
	public void executaAcao( Token token ) {

		if ( getSemantico().isUsingTypeForReturn() ) {

			getSemantico().getCurrentMethod().setReturnType(
				token.getLexeme() );

			getSemantico().setUsingTypeForReturn( false );

		} else {

			if ( getSemantico().getCurrentMethod() != null ) {

				if ( getSemantico().getCurrentMethod().
					getCurrentParameter() != null ) {

					getSemantico().getCurrentMethod().
						getCurrentParameter().setType(
							token.getLexeme() );
				}
			}
		}
	}
}

/**
 * Arquivo AcaoSemantica8.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica8 extends BaseAcaoSemantica {

	public AcaoSemantica8() {

		super();
	}

	/**
    * J? sabemos o m?todo
    * e temos uma par?metro atual.
	 */
	@Override
	public void executaAcao( Token token ) {

		if ( getSemantico().isUsingTypeForReturn() ) {

			getSemantico().getCurrentMethod().setReturnType(
```

```
                                token.getLexeme() );

                        getSemantico().setUsingTypeForReturn( false );

                } else {

                        getSemantico().getCurrentMethod().
                                getCurrentParameter().setType( token.getLexeme() );
                }
        }
}

/**
 * Arquivo AcaoSemantica9.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public class AcaoSemantica9 extends BaseAcaoSemantica {

        public AcaoSemantica9() {

                super();
        }

        /**
    * J? sabemos o m?todo
    * e temos uma par?metro atual.
         */
        @Override
        public void executaAcao( Token token ) {

                if ( getSemantico().isUsingTypeForReturn() ) {

                        getSemantico().getCurrentMethod().setReturnType(
                                token.getLexeme() );

                        getSemantico().setUsingTypeForReturn( false );

                } else {

                        getSemantico().getCurrentMethod().
                                getCurrentParameter().setType( token.getLexeme() );
                }
        }
}


/**
 * Arquivo BaseAcaoSemantica.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.acoes;

import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Semantico;
import inf.ufsc.br.tcc.cantu.parser.jvm2llvm.Token;

public abstract class BaseAcaoSemantica implements AcaoSemantica {

        private Semantico semantico;
```

```
	public BaseAcaoSemantica() {
	}

	@Override
	public void conhecaoSemantico( Semantico semantico ) {

		this.semantico = semantico;
	}

	public Semantico getSemantico() {

		return semantico;
	}

	@Override
	public abstract void executaAcao( Token token );

}

/**
 * Arquivo Class.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator;

import java.util.HashMap;
import java.util.Map;

public class Class {

	private Map<String,MethodSignature> methods;

	public Class() {

		methods = new HashMap<String, MethodSignature>();
	}

	public Map<String,MethodSignature> getMethods() {

		return methods;
	}

	public String toString() {

		StringBuilder builder = new StringBuilder();

		MethodSignature main = getMethods().get( "main" );
		getMethods().remove( "main" );

		for ( MethodSignature method : getMethods().values() )  {

			builder.append( method );
			builder.append( "\n" );
		}

		builder.append( main );

		return builder.toString();
	}
}
```

```
/**
 * Arquivo Command.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator;

public class Command {

	private int id;
	private String leftSide;
	private String rightSide;
	private String operation;
	private boolean show;
	private boolean label;

	public Command( int id, String leftSide, String rightSide, String operation ) {

		this.id = id;
		this.leftSide = leftSide;
		this.rightSide = rightSide;
		this.operation = operation;
		this.show = true;
		this.label = false;
	}

	public boolean isLabel() {

		return label;
	}

	public void setLabel( boolean label ) {

		this.label = label;
	}

	public boolean isShow() {

		return show;
	}

	public void setShow( boolean show ) {

		this.show = show;
	}

	public int getId() {

		return id;
	}

	public void setId(int id) {

		this.id = id;
	}

	public String getLeftSide() {

		return leftSide;
	}
```

```
	public void setLeftSide(String leftSide) {

		this.leftSide = leftSide;
	}

	public String getRightSide() {

		return rightSide;
	}

	public void setRightSide(String rightSide) {

		this.rightSide = rightSide;
	}

	public String getOperation() {

		return operation;
	}

	public void setOperation(String operation) {

		this.operation = operation;
	}

	public String toString() {

		StringBuilder builder = new StringBuilder();

		builder.append( getLeftSide() );

		if ( ( getOperation() != null )
			&& ( !getOperation().equals( "" ) ) ) {

			builder.append( " " );
			builder.append( getOperation() );
			builder.append( " " );
		}

		builder.append( getRightSide() );

		return builder.toString();
	}
}

/**
 * Arquivo MethodParameter.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator;

import inf.ufsc.br.tcc.cantu.parser.utils.Util;

public class MethodParameter {

	private int position;
	private String type;
	private String name;

	public MethodParameter( String type, String name ) {
```

```
		this.type = type;
		this.name = name;
	}

	public void setPosition( int position ) {

		this.position = position;
	}

	public int getPosition() {

		return position;
	}

	public String getType() {

		return type;
	}
	public void setType(String type) {

		this.type = type;
	}
	public String getName() {

		return name;
	}
	public void setName(String name) {

		this.name = name;
	}

	public String getSpecificName() {

		StringBuilder builder = new StringBuilder();
		builder.append( " %" );
		builder.append( getType() );
		builder.append( "param" );
		builder.append( String.valueOf( getPosition() ) );

		return builder.toString();
	}

	public String toString() {

		StringBuilder builder = new StringBuilder();

		builder.append( Util.jvmType2LlvmType( getType() ) );
		builder.append( getSpecificName() );

		return builder.toString();
	}
}

/**
 * Arquivo MethodSignature.java
 */
package inf.ufsc.br.tcc.cantu.parser.jvm2llvm.codegenerator;

import inf.ufsc.br.tcc.cantu.parser.utils.Util;
import inf.ufsc.br.tcc.cantu.parser.utils.Constantes;
```

```
import java.util.Map;
import java.util.TreeMap;

public class MethodSignature {

	private String currentParameter;
	private int currentCommand;
	private String name;

	private Map<Integer,Command> commands;
	private Map<String,MethodParameter> parameters;
	private String returnType;

	public MethodSignature( String name ) {

		this.name = name;
		parameters = new TreeMap<String, MethodParameter>();
		commands = new TreeMap<Integer, Command>();
	}

	public Command getCurrentCommand() {

		if ( currentCommand >= 0 ) {

			if ( getCommands().containsKey( currentCommand ) ) {

				return getCommands().get( currentCommand );
			}
		}

		return null;
	}

	private void updateCurrentCommand( int commandName ) {

		this.currentCommand = commandName;
	}

	public MethodParameter getCurrentParameter() {

		if ( currentParameter != null ) {

			if ( getParameters().containsKey( currentParameter ) ) {

				return getParameters().get( currentParameter );
			}
		}

		return null;
	}

	private void updateCurrentParameter( String parameterName ) {

		this.currentParameter = parameterName;
	}

	public String getName() {

		return name;
```

```
}

public void setName(String name) {

	this.name = name;
}

public void addEmptyCommad() {

	addCommad(  "" );
}

public void addCommad( String leftSide ) {

	int id = getCommands().size();

	if ( !getCommands().containsKey( id ) ) {

		getCommands().put( id,
			new Command( id, leftSide, "", "" ) );
	}

	updateCurrentCommand( id );
}

public void addCommad( int id, String leftSide,
	String rightSide, String operation ) {

	if ( !getCommands().containsKey( id ) ) {

		getCommands().put( id,
			new Command( id, leftSide, rightSide, operation ) );
	}

	updateCurrentCommand( id );
}

public Map<Integer, Command> getCommands() {

	return commands;
}

public void addEmptyParameter() {

	addParameter( String.valueOf(
		getParameters().size() ), "" );
}

public void addParameter( String name, String type ) {

	if ( !getParameters().containsKey( name ) ) {

		getParameters().put( name,
			new MethodParameter( type, name ) );
	}

	updateCurrentParameter( name );

	getCurrentParameter().setPosition(
		getParameters().size()-1 );
```

```
}

public Map<String, MethodParameter> getParameters() {

	return parameters;
}

public void setParameters(Map<String, MethodParameter> parameters) {

	this.parameters = parameters;
}

public String getReturnType() {

	return returnType;
}

public void setReturnType(String returnType) {

	this.returnType = returnType;
}

public String toString() {

	StringBuilder builder = new StringBuilder();

	builder.append( "define " );
	builder.append( Util.jvmType2LlvmType( getReturnType() ) );
	builder.append( " @" );
	builder.append( getName() );
	builder.append( "(" );

	int i = 0;
	for ( MethodParameter param : getParameters().values() ) {

		builder.append( param );
		if( ++i < getParameters().size() ) {

			builder.append( "," );
		}
	}

	builder.append( ")" );
	builder.append( "\n" );
	builder.append( "{" );
	builder.append( "\n" );
	builder.append( "entry:" );
	builder.append( "\n" );

	Command previous = null;

	for ( Command command : getCommands().values() ) {

		if ( command.isLabel() ) {

			if ( previous.getLeftSide().
				indexOf( "br label" ) < 0  ) {

				Command temp = new Command(0,
					"br label %"+ command.getLeftSide().
```

```
                                                        replace( ":", "" ),
                                                                "", "" );

                                        builder.append( temp );
                                        builder.append( "\n" );
                                }
                        }

                        if ( command.isShow() ) {

                                if ( !command.isLabel() ) {

                                        builder.append( "    " );
                                }

                                builder.append( command );
                                builder.append( "\n" );
                        }

                        previous = command;
                }

                if ( getReturnType().equals(
                        Constantes.LLVM_TYPE_VOID ) ) {

                        builder.append( "    " );
                        builder.append( "ret void" );
                        builder.append( "\n" );
                }

                builder.append( "}" );
                builder.append( "\n" );

                return builder.toString();
        }
}

/**
 * Arquivo Constantes.java
 */
package inf.ufsc.br.tcc.cantu.parser.utils;

public final class Constantes {

        public static final String JVM_TYPE_INT = "int";
        public static final String LLVM_TYPE_INT = "i32";

        public static final String JVM_TYPE_VOID = "void";
        public static final String LLVM_TYPE_VOID = "void";

        public static final int STACK_TYPE_INT = 1;
        public static final int STACK_TYPE_VAR = 2;

        public static final int NOP = 100;
        public static final int ACONST_NULL = 101;
        public static final int ICONST_M1 = 102;
        public static final int ICONST_0 = 103;
        public static final int ICONST_1 = 104;
        public static final int ICONST_2 = 105;
        public static final int ICONST_3 = 106;
```

```
public static final int ICONST_4 = 107;
public static final int ICONST_5 = 108;
public static final int LCONST_0 = 109;
public static final int LCONST_1 = 110;
public static final int FCONST_0 = 111;
public static final int FCONST_1 = 112;
public static final int FCONST_2 = 113;
public static final int DCONST_0 = 114;
public static final int DCONST_1 = 115;
public static final int BIPUSH = 116;
public static final int SIPUSH = 117;
public static final int LDC = 118;
public static final int LDC_W = 119;
public static final int LDC2_W = 120;
public static final int ILOAD = 121;
public static final int LLOAD = 122;
public static final int FLOAD = 123;
public static final int DLOAD = 124;
public static final int ALOAD = 125;
public static final int ILOAD_0 = 126;
public static final int ILOAD_1 = 127;
public static final int ILOAD_2 = 128;
public static final int ILOAD_3 = 129;
public static final int LLOAD_0 = 130;
public static final int LLOAD_1 = 131;
public static final int LLOAD_2 = 132;
public static final int LLOAD_3 = 133;
public static final int FLOAD_0 = 134;
public static final int FLOAD_1 = 135;
public static final int FLOAD_2 = 136;
public static final int FLOAD_3 = 137;
public static final int DLOAD_0 = 138;
public static final int DLOAD_1 = 139;
public static final int DLOAD_2 = 140;
public static final int DLOAD_3 = 141;
public static final int ALOAD_0 = 142;
public static final int ALOAD_1 = 143;
public static final int ALOAD_2 = 144;
public static final int ALOAD_3 = 145;
public static final int IALOAD = 146;
public static final int LALOAD = 147;
public static final int FALOAD = 148;
public static final int DALOAD = 149;
public static final int AALOAD = 150;
public static final int BALOAD = 151;
public static final int CALOAD = 152;
public static final int SALOAD = 153;
public static final int ISTORE = 154;
public static final int LSTORE = 155;
public static final int FSTORE = 156;
public static final int DSTORE = 157;
public static final int ASTORE = 158;
public static final int ISTORE_0 = 159;
public static final int ISTORE_1 = 160;
public static final int ISTORE_2 = 161;
public static final int ISTORE_3 = 162;
public static final int LSTORE_0 = 163;
public static final int LSTORE_1 = 164;
public static final int LSTORE_2 = 165;
public static final int LSTORE_3 = 166;
```

```
public static final int FSTORE_0 = 167;
public static final int FSTORE_1 = 168;
public static final int FSTORE_2 = 169;
public static final int FSTORE_3 = 170;
public static final int DSTORE_0 = 171;
public static final int DSTORE_1 = 172;
public static final int DSTORE_2 = 173;
public static final int DSTORE_3 = 174;
public static final int ASTORE_0 = 175;
public static final int ASTORE_1 = 176;
public static final int ASTORE_2 = 177;
public static final int ASTORE_3 = 178;
public static final int IASTORE = 179;
public static final int LASTORE = 180;
public static final int FASTORE = 181;
public static final int DASTORE = 182;
public static final int AASTORE = 183;
public static final int BASTORE = 184;
public static final int CASTORE = 185;
public static final int SASTORE = 186;
public static final int POP = 187;
public static final int POP2 = 188;
public static final int DUP = 189;
public static final int DUP_X1 = 190;
public static final int DUP_X2 = 191;
public static final int DUP2 = 192;
public static final int DUP2_X1 = 193;
public static final int DUP2_X2 = 194;
public static final int SWAP = 195;
public static final int IADD = 196;
public static final int LADD = 197;
public static final int FADD = 198;
public static final int DADD = 199;
public static final int ISUB = 200;
public static final int LSUB = 201;
public static final int FSUB = 202;
public static final int DSUB = 203;
public static final int IMUL = 204;
public static final int LMUL = 205;
public static final int FMUL = 206;
public static final int DMUL = 207;
public static final int IDIV = 208;
public static final int LDIV = 209;
public static final int FDIV = 210;
public static final int DDIV = 211;
public static final int IREM = 212;
public static final int LREM = 213;
public static final int FREM = 214;
public static final int DREM = 215;
public static final int INEG = 216;
public static final int LNEG = 217;
public static final int FNEG = 218;
public static final int DNEG = 219;
public static final int ISHL = 220;
public static final int LSHL = 221;
public static final int ISHR = 222;
public static final int LSHR = 223;
public static final int IUSHR = 224;
public static final int LUSHR = 225;
public static final int IAND = 226;
```

```
public static final int LAND = 227;
public static final int IOR = 228;
public static final int LOR = 229;
public static final int IXOR = 230;
public static final int LXOR = 231;
public static final int IINC = 232;
public static final int I2L = 233;
public static final int I2F = 234;
public static final int I2D = 235;
public static final int L2I = 236;
public static final int L2F = 237;
public static final int L2D = 238;
public static final int F2I = 239;
public static final int F2L = 240;
public static final int F2D = 241;
public static final int D2I = 242;
public static final int D2L = 243;
public static final int D2F = 244;
public static final int I2B = 245;
public static final int I2C = 246;
public static final int I2S = 247;
public static final int LCMP = 248;
public static final int FCMPL = 249;
public static final int FCMPG = 250;
public static final int DCMPL = 251;
public static final int DCMPG = 252;
public static final int IFEQ = 253;
public static final int IFNE = 254;
public static final int IFLT = 255;
public static final int IFGE = 256;
public static final int IFGT = 257;
public static final int IFLE = 258;
public static final int IF_ICMPEQ = 259;
public static final int IF_ICMPNE = 260;
public static final int IF_ICMPLT = 261;
public static final int IF_ICMPGE = 262;
public static final int IF_ICMPGT = 263;
public static final int IF_ICMPLE = 264;
public static final int IF_ACMPEQ = 265;
public static final int IF_ACMPNE = 266;
public static final int GOTO = 267;
public static final int JSR = 268;
public static final int RET = 269;
public static final int TABLESWITCH = 270;
public static final int LOOKUPSWITCH = 271;
public static final int IRETURN = 272;
public static final int LRETURN = 273;
public static final int FRETURN = 274;
public static final int DRETURN = 275;
public static final int ARETURN = 276;
public static final int RETURN = 277;
public static final int GETSTATIC = 278;
public static final int PUTSTATIC = 279;
public static final int GETFIELD = 280;
public static final int PUTFIELD = 281;
public static final int INVOKEVIRTUAL = 282;
public static final int INVOKESPECIAL = 283;
public static final int INVOKESTATIC = 284;
public static final int INVOKEINTERFACE = 285;
public static final int NEW = 286;
```

```
	public static final int NEWARRAY = 287;
	public static final int ANEWARRAY = 288;
	public static final int ARRAYLENGTH = 289;
	public static final int ATHROW = 290;
	public static final int CHECKCAST = 291;
	public static final int INSTANCEOF = 292;
	public static final int MONITORENTER = 293;
	public static final int MONITOREXIT = 294;
	public static final int WIDE = 295;
	public static final int MULTIANEWARRAY = 296;
	public static final int IFNULL = 297;
	public static final int IFNONNULL = 298;
	public static final int GOTO_W = 299;
	public static final int JSR_W = 300;
	public static final int BREAKPOINT = 301;
	public static final int IMPDEP1 = 302;
	public static final int IMPDEP2 = 303;
}

/**
 * Arquivo OrderComparator.java
 */
package inf.ufsc.br.tcc.cantu.parser.utils;

import java.util.Comparator;


public class OrderComparator implements Comparator<Integer> {

	@Override
	public int compare( Integer o1, Integer o2 ) {

		return o1.compareTo( o2 );
	}
}

/**
 * Arquivo Util.java
 */
package inf.ufsc.br.tcc.cantu.parser.utils;

import java.util.HashMap;
import java.util.Map;


public final class Util {

	private static final Map<String,String> jvmTypeXLlvmType =
		new HashMap<String,String>();

	static {

		jvmTypeXLlvmType.put( Constantes.JVM_TYPE_INT,
			Constantes.LLVM_TYPE_INT );

		jvmTypeXLlvmType.put( Constantes.JVM_TYPE_VOID,
			Constantes.LLVM_TYPE_VOID );
	}

	public static final String jvmType2LlvmType( String jvmType ) {
```

```
		if ( jvmTypeXLlvmType.containsKey( jvmType ) ) {

			return jvmTypeXLlvmType.get( jvmType );
		}

		return jvmType;
	}

	private static final Map<String,String> javaTypeXJvmType =
		new HashMap<String, String>();

	static {

		javaTypeXJvmType.put( "I", "int" );
		javaTypeXJvmType.put( "D", "double" );
		javaTypeXJvmType.put( "Ljava/lang/String;", "string" );
		javaTypeXJvmType.put( "B", "bool" );
		javaTypeXJvmType.put( "L", "int" );
		javaTypeXJvmType.put( "V", "void" );
	}

	public static final String javaType2JvmType( String key ) {

		if ( javaTypeXJvmType.containsKey( key ) ) {

			return javaTypeXJvmType.get( key );
		}

		return key;
	}

	public static final String javaArgsToJvmArgs( String args ) {

		String[] varargs =
			new String[]{
				"Ljava/lang/String;","I","D",
					"B","L","V"};

		String returnargs = args.substring(
			args.indexOf( "(")+1, args.indexOf( ")") );

		for ( String arg : varargs ) {

			returnargs = returnargs.replaceAll(
				arg, javaType2JvmType( arg )+"," );
		}

		returnargs = returnargs+"@";

		returnargs = returnargs.replaceAll( ",@", "" );
		returnargs = returnargs.replaceAll( "@", "" );
		returnargs = "("+ returnargs +")";

		return returnargs;
	}
}
```