

Universidade Federal de Santa Catarina

Aplicativo para plataforma Android: "AgileClass", Editor de Diagramas de Classes para Dispositivos Sensíveis ao Toque

Rodolfo Pamplona Tenfen

Florianópolis – SC
2017/1

Universidade Federal de Santa Catarina
Departamento de Informática e Estatística
Curso de Ciências de Computação

Aplicativo para plataforma Android: "AgileClass", Editor de Diagramas de Classes para Dispositivos Sensíveis ao Toque

Rodolfo Pamplona Tenfen

Trabalho de conclusão de curso apresentado
como parte dos requisitos para obtenção do
grau de Bacharel em Ciências da
Computação

Orientador: Raul Sidnei Wazlawick

Florianópolis - SC
2017/1

Rodolfo Pamplona Tenfen

Aplicativo para plataforma Android: "AgileClass", Editor de Diagramas de Classes para Dispositivos Sensíveis ao Toque

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação

Raul Sidnei Wazlawick
Orientador

Banca Examinadora

Patricia Vilain
Membro da Banca

Maurício Floriano Galimberti
Membro da Banca

Agradecimentos

Meus mais sinceros agradecimentos aos meus amigos Alyssa, Iasminy, Janine, Raiza e Stephan, pelo apoio moral e por acalmar meu desespero quando necessário.

Aos meus pais e familiares, por me apoiarem no desenvolvimento da minha vida e me permitirem chegar onde cheguei.

Ao professor Raul, meu orientador, pelo apoio desde a escolha do tema até a entrega da monografia e por sua paciência com meus erros e dificuldades durante o percurso.

Aos professores Maurício e Patricia, por aceitarem fazer parte da banca avaliadora.

Ao meu namorado, que trouxe nova luz a minha vida e ajudou a manter minha sanidade.

E a Deus, por ter me permitido vir ao mundo e viver todos os dias de minha vida. Obrigado pelas dádivas e desafios, que me trouxeram experiência para enfrentar a tarefa de desenvolver este projeto.

Resumo

A metodologia de desenvolvimento de *software* atual tem evoluído de modo a abranger formas ágeis de projeto e criação de sistemas computacionais. Contudo, as ferramentas disponíveis hoje que dão suporte ao desenvolvimento de *software* são simples demais, restritas e de baixa usabilidade, quando baratas, e complexas, com recursos muito além do necessário e confusas, quando pagas. Sendo que a grande maioria delas se restringe aos sistemas operacionais de computadores de mesa, existindo, então, sob este paradigma de utilização, ou seja, *mouse* e teclado.

Neste trabalho foi desenvolvido um sistema aplicativo para a plataforma *Android* que quebre os paradigmas clássicos. Buscou-se como resultado um editor de diagramas de classes com usabilidade apropriada a dispositivos sensíveis ao toque. Foi desenvolvido um sistema com interação com o usuário com suporte a comandos por gestos, além de entradas mais simples e comuns por botões e teclado.

O sistema também foi desenvolvido tendo em vista requisitos de usabilidade, compatibilidade e performance. Esses requisitos visam adequar o aplicativo às necessidades de desenvolvedores ágeis, bem como garantir o funcionamento do sistema em diferentes plataformas, como dispositivos com diferentes tamanhos de tela, e que possam ter restrições de *hardware*, como depender de baterias e ter especificações menos potentes. Também se deu importância a questões específicas do ambiente *Android*, como seu *look and feel* e sua característica de ter no mercado uma ampla variedade de versões do sistema operacional.

Palavras Chave:

Android, metodologia ágil, aplicativo, diagrama de classes, heurísticas de usabilidade.

Lista de Ilustrações

Figura 1: Ciclo de vida de uma Atividade	15
Figura 2: Tela inicial do Diagrid	26
Figura 3: Diagrid com três classes e relacionamentos entre elas	27
Figura 4: Tela inicial do Draw Express	28
Figura 5: Alguns dos gestos que o Draw Express reconhece	29
Figura 6: À esquerda: área de edição de texto do Draw Express. À direita: Exemplo de retângulo sem texto e retângulo resultante do texto à esquerda	30
Figura 7: Tela inicial do HandyUML	31
Figura 8: Botões do HandyUML	32
Figura 9: Telas para adição de atributos e de métodos no HandyUML	33
Figura 10: Exemplo de descrição textual de diagrama de classes e a imagem gerada a partir do texto	34
Figura 11: Diagrama de Classes do Modelo do AgileClass	38
Figura 12: Tela inicial do AgileClass	39
Figura 13: Áreas principais da tela inicial	40
Figura 14: Tela inicial na horizontal.....	41
Figura 15: Exemplos de telas de edição de classe, relacionamento, operação e atributo, respectivamente	42
Figura 16: AgileClass com algumas classes de exemplo com a Classe 3 selecionada ..	47
Figura 17: Tela inicial na horizontal com exemplos	47
Figura 18: Código que verifica a versão do sistema Android sendo executada em tem de execução	49
Figura 19: Ilustração de como o Android mapeia tamanhos e densidades reais em tamanhos e densidades generalizadas	49
Figura 20: Exemplo de estrutura de diretórios mostrando uma pasta para layouts para orientação vertical (sem sufixo) e outra para orientação horizontal.....	50
Figura 21: Exemplo de estrutura de diretórios mostrando pastas que combinam diferentes orientações e tamanhos de tela.....	50
Figura 22: Exemplo de estrutura de diretórios mostrando uma pasta recursos desenháveis (como imagens) para diferentes densidades de píxeis	51

Lista de Abreviaturas e Siglas

<i>API</i>	<i>Application Programming Interface</i>
<i>CAD</i>	<i>Computer Aided Design</i>
<i>CASE</i>	<i>Computer-Aided Software Engineering</i>
<i>IBGE</i>	Instituto Brasileiro de Geografia e Estatística
<i>IDE</i>	<i>Integrated Development Environment</i>
<i>MVC</i>	<i>Model-View-Controller</i>
<i>PNG</i>	<i>Portable Network Graphics</i>
<i>TCC</i>	Trabalho de Conclusão de Curso
<i>UML</i>	<i>Unified Modeling Language</i>
<i>XML</i>	<i>Extensible Markup Language</i>

Sumário

1	Introdução	10
1.1	Objetivos	11
1.1.1	Objetivo Geral	11
1.1.2	Objetivos Específicos	11
1.2	Estrutura do Documento	12
2	Fundamentação	14
2.1	Android	14
2.1.1	Atividade	14
2.1.2	View	15
2.1.3	Canvas	16
2.1.4	Paint	16
2.2	Compatibilidade	16
2.2.1	Bibliotecas de Suporte	17
2.3	Diagramas de Classe	17
2.4	Heurísticas de Usabilidade	18
2.4.1	Jacob Nielsen	19
2.4.2	Ben Shneiderman	21
2.4.3	J. M. Christian Bastien e Dominique L. Scapin	22
2.5	Desenvolvimento Ágil de Software	23
3	Estudo do Estado da Arte	25
3.1	Diagrid	25
3.2	Draw Express	28
3.3	HandyUML	30
3.4	iPlantUML	33
4	Desenvolvimento	36

4.1	Requisitos Funcionais	36
4.2	Requisitos Não-Funcionais	37
4.3	Modelo do Sistema	37
4.4	Interação com o Usuário	39
4.4.1	Telas	39
4.4.1.1	Tela Inicial.....	39
4.4.1.2	Telas de Edição	41
4.4.1.3	Gestos	42
4.4.1.4	Alternativas	44
4.4.2	Componentes Visuais	44
4.4.2.1	Atividade Principal.....	44
4.4.2.2	Área de Edição	45
4.4.2.3	Classe	46
4.4.2.4	Relacionamento	48
4.5	Compatibilidade.....	48
4.6	Interface Escalável.....	49
5	Conclusão	52
5.1	Trabalhos Futuros	52
6	Referências	54

1 Introdução

No desenvolvimento de projetos, faz-se uso de ferramentas e tecnologias que permitam e facilitem alcançar objetivos. A popularização do computador o firmou como ferramenta e hoje ele é utilizado em diversas áreas do conhecimento. As funcionalidades de um computador são providas por *softwares*, muitos dos quais são desenvolvidos especificamente para certas áreas, como ferramentas de desenho assistido por computador, ou *CAD* em inglês, no intuito de auxiliar o desenvolvimento e execução de projetos. Logo, a Ciência da Computação e a Engenharia de *Software* são áreas do conhecimento de grande importância para outras áreas, pois são responsáveis pelo desenvolvimento de ferramentas modernas que permitem tornar realidade grandes projetos de engenharia.

Contudo, o próprio desenvolvimento de um *software* se trata de um projeto, o qual possui espaço para auxílio e automatização. Hoje, o processo de elaboração e criação de *software* é assistido por ferramentas como *IDEs*, editores de diagramas, ferramentas *CASE* e sistemas de *design* gráfico de interface com usuário.

O uso de ferramentas *CASE* promete aumentar a qualidade do *software* gerado e a produtividade da geração de código, mas, de acordo com Iivari (1996) [4], o uso de ferramentas *CASE* é muito menor do que o esperado, tendo em vista os seus benefícios. Por outro lado, Ambler (2009a) [9] menciona potenciais custos de se usar essas ferramentas. Dentre eles tem-se custos com treinamento, interface com usuário pobre e alta complexidade das ferramentas.

Esses custos tornam ferramentas *CASE* comuns inadequadas para o desenvolvimento ágil de *software*. Como o nome indica, a ideia desse paradigma de desenvolvimento é acelerar o projeto, encurtando o tempo entre entregas para o cliente, diminuindo, principalmente, o volume de documentação e enfatizando a geração de código. Tendo em vista esses princípios, Ambler recomenda o uso da ferramenta mais simples possível, em detrimento de ferramentas complexas que fazem muito mais do que o estritamente necessário. Ele, porém, propõe a utilização de objetos físicos como cartões, papel e quadros brancos. Essas ferramentas, porém, são demasiadamente simples, gerando modelos grosseiros e incompletos e sem possibilidade de serem automatizados ou de serem usados para a automatização de processos futuros.

Logo, métodos ágeis não são totalmente desprovidos de documentação e poderiam se favorecer do uso de ferramentas *CASE* mais sofisticadas. A proposta deste trabalho, então, é explorar as premissas da metodologia ágil, de modo a desenvolver um editor de diagrama de classes com alto nível de usabilidade e com um conjunto necessário e suficiente de funcionalidades para auxiliar o desenvolvimento de *software* de forma ágil, reduzindo os custos do uso da própria ferramenta.

Antes da popularização de *tablets* e *smartphones*, sistemas computacionais pessoais sofisticados capazes de suportar ferramentas de desenvolvimento resumiam-se a computadores de mesa e *laptops*, ambos comumente controlados por *mouse* e teclado. O paradigma de interação com computadores, contudo, mudou e se diversificou, aparelhos móveis e a utilização de telas sensíveis ao toque substituindo totalmente ou trabalhando em conjunto com *mouse* e teclado se tornaram populares e amplamente aceitos, de acordo com o IBGE [29], os acessos à Internet feitos por celulares apenas já ultrapassam os feitos por microcomputadores no Brasil, o que mostra que estes aparelhos têm se tornado mais e mais populares. Esse novo paradigma de interação com computadores e a sua popularização trazem novas possibilidades de desenvolvimento de ferramentas, com formas de interação diferentes e requisitos de usabilidade diferentes.

A escolha da plataforma *Android* se dá pela sua popularidade [28] em dispositivos móveis com telas sensíveis ao toque, como *smartphones* e *tablets*. Seu ambiente de desenvolvimento provê uma *API* completa para a implementação do aplicativo, seguindo os requisitos de usabilidade que se quer explorar.

1.1 Objetivos

1.1.1 Objetivo Geral

Desenvolvimento de aplicativo para criação e edição de diagramas de classes para o sistema operacional *Android* que tenha nível de usabilidade tendo em vista requisitos de ferramentas *CASE* para desenvolvimento ágil.

1.1.2 Objetivos Específicos

- Estudar a plataforma *Android*, seus componentes e bibliotecas utilizadas no desenvolvimento do aplicativo.

- Estudar heurísticas de usabilidade e identificar sua importância no desenvolvimento de *software*.
- Estudar a metodologia ágil e identificar seus requisitos no que se trata de ferramentas utilizadas para desenvolvimento ágil de *software*.
- Desenvolver o modelo do sistema geral, tratando a criação e edição de diagramas de classes. Desenvolver a visão do sistema, usando uma interface simples e agradável. E desenvolver a interação com o usuário, permitindo entradas via tela sensível ao toque, com reconhecimento de gestos.
- Permitir a escalabilidade da interface para telas pequenas, de 3,5 a 5 polegadas, como encontradas em *smartphones* e outros dispositivos móveis, a telas médias, 7 a 12 polegadas, como encontradas em *tablets*.

1.2 Estrutura do Documento

No capítulo 2 é apresentada a fundamentação teórica deste trabalho. São tratados os assuntos pertinentes para o desenvolvimento, tanto do ponto de vista do desenvolvimento técnico do projeto, quanto do desenvolvimento lógico e cognitivo do projeto. Primeiramente fala-se sobre as questões técnicas da plataforma *Android*, seus componentes e como ela permite alcançar compatibilidade entre diversas versões do sistema operacional e diferentes *hardwares*. Depois são explicados os fundamentos dos conceitos usados no desenvolvimento do projeto, que são diagramas de classe, heurísticas de usabilidade e metodologia ágil.

No capítulo 3 é feita a revisão do estado da arte, isto é, são analisadas soluções disponíveis no mercado para a proposta deste projeto. São discutidos quatro aplicativos que podem ser encontrados na loja de aplicativos do *Android*. Sobre cada um são analisados seus pontos fortes e seus pontos fracos e é medida sua aptidão ao seu uso em um cenário de desenvolvimento ágil.

No capítulo 4 está descrito o processo de desenvolvimento do trabalho e os principais resultados obtidos. São explicados os requisitos funcionais e os requisitos não-funcionais do aplicativo, bem como o modelo gerado e a forma como a interação com o usuário ocorre, levando em conta os requisitos de usabilidade impostos. Por fim, são tratadas a forma como se alcançou compatibilidade entre diversas versões do *Android* e diversos *hardwares* e a forma como se alcançou a escalabilidade da interface.

Por fim, no capítulo 5, conclui-se o projeto, apresentando as considerações finais sobre os resultados obtidos. Também são tratados potenciais trabalhos futuros, isto é, formas como o trabalho pode ser enriquecido e aperfeiçoado.

2 Fundamentação

2.1 Android

Android é um sistema operacional de código aberto para dispositivos móveis baseado no núcleo do sistema operacional Linux [18] e desenvolvido pela Google [15]. O sistema foi desenvolvido para ser usado, principalmente, através de uma tela sensível ao toque, mas oferece suporte a diversos meios de entrada [30] e possui suporte para alguns comandos por voz, como o *Google Voice Search*, aplicativo que permite realizar pesquisas por voz [6] e é comparável ao *Siri* da Apple [33].

Hoje o sistema *Android* é o mais utilizado em *smartphones* do mundo [19]. Sua abrangência o torna uma plataforma de desenvolvimento ideal, visto que a maioria dos usuários de sistemas móveis terão acesso a aplicações desenvolvidas para este sistema operacional. Além disso, a plataforma oferece todas as ferramentas necessárias para desenvolvimento de *software*. Dentre os principais elementos da *API* do sistema *Android* que servem como base para a criação de aplicativos pode-se citar: Atividades, a classe *View*, a classe *Canvas* e a classe *Paint*.

2.1.1 Atividade

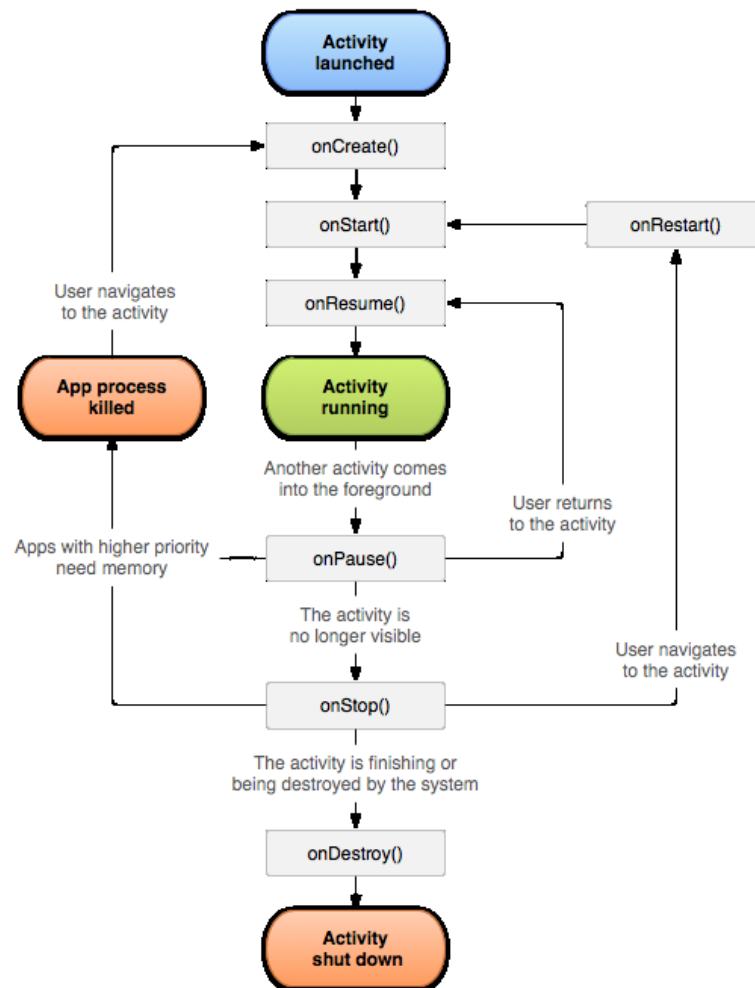
Atividades são componentes da *API* do sistema *Android*, utilizadas por aplicativos, pelas quais os usuários podem interagir com a aplicação. Às atividades são atribuídas uma janela, que pode preencher a tela ou flutuar sobre outras janelas [12].

Um aplicativo pode conter várias atividades, mas normalmente uma delas é a principal. Essa atividade serve como porta de entrada para o usuário, serve como tela inicial e nela ou a partir dela é possível executar as principais funcionalidades do aplicativo.

As atividades são responsáveis por gerenciar seu ciclo de vida (figura 1), isto é, reagir de forma apropriada a eventos do sistema, como iniciar, minimizar, restaurar, fechar, etc. Para tanto, a assinatura da classe oferece métodos que podem ser sobrescritos para gerenciar o programa, como *onCreate*, *onPause*, *onResume* e outros. A função da gerência do ciclo de vida de uma atividade é controlar os recursos que ela utiliza. Ao minimizar uma atividade, é importante que ela libere o máximo de recursos possíveis, a fim de garantir eficiência energética, pouco uso de memória e outros

requisitos de desempenho, mas seja capaz de retornar ao seu estado anterior caso seja restaurada.

Figura 1: Ciclo de vida de uma Atividade



2.1.2 View

View é uma superclasse estendida por todos os componentes visuais do *Android*, desde *TextView*, que funciona como um rótulo, até *Button*, para botão [41]. Ela contém os métodos necessários para organizar o componente que a estende na tela, isto é, definir seu *layout*, possui alguns métodos para implementar interação pela interface de toque, isto é, como o componente deve reagir a um toque, e é capaz de se desenhar na tela.

No *AgileClass*, os principais componentes visuais foram implementados estendendo a classe *View*. Isso permite reuso de código já presente na *API* padrão do

Android, bem como uma integração mais natural do código do desenvolvedor com as funcionalidades do sistema operacional.

2.1.3 Canvas

O sistema *Android* possui suporte à aceleração por *hardware* e sua *API* fornece diversas formas de desenvolver aplicativos multimídia, eficazes e interativos [17]. O *framework Android* permite o desenvolvimento de gráficos em 2D e 3D e permite desenvolver aplicativos graficamente intensivos e aplicativos com gráficos mais estáticos.

Um componente que se destaca para o desenvolvimento menos graficamente intenso é o *Canvas*. Ele é associado à classe *View* e permite desenhar diretamente na tela, informando valores em pixel para tamanho e posição dos componentes na tela. Ou seja, é com o auxílio de um objeto *Canvas* que a classe *View* se desenha na tela.

A classe *Canvas* possui métodos que permitem desenhar formas comuns na tela. Alguns exemplos de formas suportadas são retângulos, círculos e linhas. Também é possível desenhar texto e, para formas menos comuns, *paths*, que são listas de pontos ligados por linhas.

2.1.4 Paint

Completando o componente *Canvas*, vem a classe *Paint* (38). Essa classe encapsula informações de estilo, cor e tema do “pincel” que pinta o *Canvas*. Com um objeto dessa classe é possível alterar o tipo de linha usada (tracejada ou contínua), espessura e cor, permite determinar se polígonos serão preenchidos ou não e mais. O objeto é informado ao *Canvas* na chamada à rotina de desenho, isso permite modificar rapidamente e dinamicamente qual pincel usar.

2.2 Compatibilidade

O sistema *Android* foi projetado para ser executado em dispositivos diversos, com diferenças que vão desde versão do sistema operacional a diferença de tamanho de tela e plataforma (televisão, relógio, carro, tablete, telefone e mais) [22]. É possível limitar os dispositivos compatíveis com um aplicativo, baseado na versão da plataforma, configurações da tela e/ou características do dispositivo, mas o ideal é desenvolver

aplicações que sejam amplamente compatíveis, buscando utilizar os recursos mais recentes e ao mesmo tempo oferecendo suporte a dispositivos mais antigos.

Para alcançar suporte a múltiplos tipos de dispositivos, a *API* do *Android* permite consultar em tempo de execução a versão dela utilizada e quais as características do *hardware* do dispositivo. É possível, por exemplo, verificar se um *smartphone* conta com uma bússola. Isso permite que o desenvolvedor escreva código com funcionalidades adaptáveis.

O ambiente de desenvolvimento também oferece suporte à definição de diferentes *layouts* para diferentes tamanhos de tela. São reconhecidas quatro categorias generalizadas de densidade de pixel (*ldpi*, *mdpi*, *hdpi* e *xhdpi*) e também quatro de tamanho de tela (*small*, *normal*, *large* e *xlarge*).

O *layout* estático do aplicativo costuma ser definido em arquivos *XML*. É possível criar diferentes arquivos *XML* para diferentes tamanhos de tela e também para diferentes orientações da tela (horizontal ou vertical). O *layout* apropriado é escolhido automaticamente, permitindo ao desenvolvedor criar aplicações de forma independente de plataforma, mas ajustáveis a necessidades específicas de cada sistema.

2.2.1 Bibliotecas de Suporte

O pacote *Android Support Library* oferece bibliotecas que viabilizam o desenvolvimento de aplicações que sejam compatíveis com múltiplas versões do sistema operacional [39]. As duas bibliotecas principais são a *v4 support* e a *v7 appcompat*. Elas contêm rotinas e classes que formam uma forma de *middleware* entre a implementação dependente de versão da *API* do sistema e o código do desenvolvedor.

2.3 Diagramas de Classe

Sommerville (2010) [2] escreve em seu livro *Software Engineering* que diagramas de classe são usados quando se está desenvolvendo um sistema orientado a objetos, a fim de mostrar as classes do sistema e as relações entre elas [2]. Ele propõe que a forma mais simples para se representar uma classe é com um retângulo com o nome da classe dentro. O relacionamento, ou associação, entre as classes pode ser mostrado como uma linha que ligue os retângulos das classes relevantes. Também é possível mostrar quantos objetos de uma classe participam de uma relação, ou associação, também conhecido como multiplicidade, através de números e símbolos próximos à linha e às classes

relevantes. Para denotar a multiplicidade de uma relação, usa-se números (zero ou maior) ou o símbolo “*”, para representar que não há limite superior para o número de objetos envolvidos.

Quando o diagrama exibe detalhes de uma classe, a representação dessa classe é feita com um retângulo com três seções horizontais. A seção superior contém o nome da classe. No meio fica a seção que contém os atributos de uma classe. Por fim, na área inferior do retângulo, consta a seção que contém as operações de uma classe (conhecidas como assinatura dos métodos na linguagem *Java* e outras linguagens orientadas a objeto).

Quanto à relação entre classes, Somerville cita, além de associação, herança e agregação, mas é importante também tratar de composição e implementação. A relação mais simples é a de associação, denotada por uma linha contínua, representa o uso de uma classe por parte de outra. Uma associação pode ser unidirecional ou bidirecional, pode ter um nome e tem multiplicidades. Herança expressa uma relação de generalização, isto é, significa que uma classe é um subtipo de outra classe. Implementação indica que uma classe realiza ou implementa o comportamento que um outro componente especifica. Na linguagem *Java*, as estruturas que especificam comportamento são conhecidas como interfaces. Por fim, agregação e composição denotam, respectivamente, uma relação de “tem um” e uma relação de “é formado por um”.

2.4 *Heurísticas de Usabilidade*

Usabilidade é o termo dado à medida da facilidade de uso de objetos e ferramentas feitos pelo ser humano. Jakob Nielsen (2012) [36] define usabilidade em computadores como sendo um atributo de qualidade que mede o quão fácil de usar são as interfaces com o usuário. Nielsen argumenta que a importância de se ter usabilidade é o fato de que usuários deixam de usar produtos, como páginas *web*, se elas forem difíceis ou desagradáveis de usar.

Existem diversas formas de se aumentar a usabilidade de um sistema. Alguns métodos analisam a resposta de usuários diretamente, enquanto outros utilizam princípios e estratégias para buscar boa usabilidade desde o início do *design* e desenvolvimento do projeto sem interação com usuários finais. Uma dessas formas é a chamada Avaliação Heurística, onde se busca encontrar problemas de usabilidade tendo

em vista princípios de usabilidade escolhidos de acordo com o sistema avaliado. Esses princípios, ou heurísticas, devem ser seguidos durante o projeto para se aumentar as chances de obtenção de um produto final de qualidade. Dentre as heurísticas mais conhecidas, pode-se citar as “10 Heurísticas de Usabilidade para *Design* de Interfaces de Usuário” de Jakob Nielsen, as “Oito Regras de Ouro para *Design* de Interfaces” de Ben Shneiderman e os “Critérios Ergonômicos para a Avaliação de Interfaces Humano-Computador” de Bastien e Scapin. Esses conjuntos de heurísticas possuem diversos pontos em comum e poucas especificidades únicas. Dentre os três conjuntos citados, as heurísticas de Nielsen são as mais conhecidas e mais comumente usadas.

2.4.1 Jacob Nielsen

Nielsen (1995) [37] propõe dez heurísticas para o projeto de interação humano-computador. Esses princípios são regras amplas e não diretrizes de usabilidade específicas. Elas são úteis para nortear os esforços dos desenvolvedores e diminuir a dependência em testes com usuários. As heurísticas propostas são:

1. Visibilidade do estado do sistema:

O sistema deve sempre manter os usuários informados do que está acontecendo através de *feedback* apropriado e em tempo adequado. Isto é, o usuário deve ser capaz de saber qual o estado do sistema. Caso este esteja tomando alguma ação que tome tempo considerável, deve-se exibir uma barra de progresso.

2. Compatibilidade entre sistema e mundo real:

O sistema deve falar a língua do usuário com palavras, frases e conceitos familiares a ele, em detrimento de termos específicos do sistema. Deve-se seguir convenções do mundo real, exibindo a informação de uma forma natural e lógica.

3. Controle e liberdade para o usuário:

Nielsen afirma que usuários comumente escolhem funcionalidades do sistema por engano e precisam de “saídas emergenciais” claramente marcadas para sair do estado indesejado sem ter que passar por diálogos extensos. Deve-se suportar ações de desfazer e refazer.

4. Consistência e padrões:

Usuários não devem ter de se perguntar se diferentes palavras, situações e ações significam a mesma coisa. Deve-se seguir as convenções da plataforma usada.

5. Prevenção de erros:

Para Nielsen, melhor do que boas mensagens de erro é um *design* cuidadoso que previna que erros ocorram. Seja eliminando condições que costumam causar erros ou verificando essas condições e buscando confirmação do usuário antes que ele finalize sua ação.

6. Reconhecimento em vez de lembrança:

Deve-se minimizar o uso da memória do usuário tornando objetos, ações e opções visíveis. O usuário não deve ter que lembrar de informação de uma parte do diálogo para outra. Instruções de como usar o sistema devem ser visíveis e fáceis de reaver sempre que apropriado.

7. Flexibilidade e eficiência de uso:

Aceleradores, invisíveis ao usuário novato, podem frequentemente aumentar a velocidade a interação de usuários veteranos, de tal modo que o sistema possa atender a tanto usuários inexperientes quanto experientes.

8. Projeto minimalista e estético:

Diálogos não devem conter informações que sejam irrelevantes ou raramente necessárias. Cada nova unidade de informação em um diálogo compete com as unidades de informação relevantes e diminui a visibilidade relativa delas.

9. Ajudar usuários a reconhecer, diagnosticar e se recuperar de erros:

Mensagens de erro devem ser expressadas em linguagem simples (sem códigos), devem indicar o problema precisamente e sugerir soluções de forma construtiva. Identificar erros através de códigos pode ser útil, mas é importante que o usuário não precise procurar o que significa o código. Em vez disso, ele deve ser capaz de identificar rapidamente qual o problema.

10. Ajuda e documentação:

Apesar de ser melhor que o sistema possa ser usado sem documentação, pode ser necessário providenciar ajuda e documentação. Qualquer informação desse tipo deve ser fácil de encontrar, focada na tarefa que o usuário quer realizar, listar passos concretos para se seguir e não ser grandes demais.

2.4.2 Ben Shneiderman

Alternativamente a Nielsen, Shneiderman (2013) [5] e (2016) [20] propôs oito regras para guiar o *design* de interfaces. As regras propostas por Shneiderman se assemelham às heurísticas de Nielsen, mas Shneiderman acrescenta a noção de grupo de ações, que são sequências lógicas de ações tomadas pelo usuário. Shneiderman afirma em suas “Oito Regras de Ouro para Projeto de Interface” que o desenvolvedor deve:

1. Esforçar-se para ter consistência:

Sequências de ações consistentes são necessárias em situações similares. Deve-se usar terminologia igual em diálogos, menus e telas de ajuda. E, por fim, comandos consistentes devem ser empregados por todo o projeto.

2. Permitir que usuários frequentes usem atalhos:

Shneiderman argumenta que a medida que a frequência de uso aumenta, também aumenta a vontade do usuário de diminuir o número de interações e aumentar o ritmo delas. Abreviações, atalhos do teclado e comandos ocultos são úteis para usuários experientes.

3. Oferecer *feedback* informativo:

Para todas as ações deve existir algum tipo de *feedback* por parte do sistema. Para ações frequentes ou ações de menor prioridade, a resposta do sistema pode ser modesta, enquanto que ações infrequentes e de maior prioridade devem ter respostas substanciais por parte do sistema.

4. Projetar diálogos para que tenham final:

Sequências de ações devem ser organizadas em grupos com começo, meio e fim. O *feedback* ao completar um grupo de ações dá aos usuários satisfação, sensação de alívio, sinaliza que é possível esquecer planos de contingência e opções de suas mentes e os indica que é possível seguir para o próximo grupo de ações.

5. Oferecer formas simples de lidar com erros:

Shneiderman diz que, tanto quanto possível, deve-se projetar um sistema tal que o usuário não possa cometer erros sérios. Se um erro acontecer, o sistema deve ser capaz de detectar o erro e oferecer mecanismos simples e compreensivos para lidar com o erro.

6. Permitir reversão fácil de ações:

Essa funcionalidade reduz ansiedade, já que o usuário sabe que erros podem ser desfeitos, argumenta Ben. Isso então encoraja exploração de opções não familiares.

As unidades reversíveis podem ser uma única ação, uma entrada de dados ou um grupo completo de ações.

7. Suportar controle local:

Usuários experientes desejam sentir que estão no comando do sistema e que o sistema responde às ações deles. Deve-se projetar um sistema que torne os usuários os iniciadores de ações em vez de os respondedores.

8. Reduzir o uso da memória de curto prazo:

As limitações de processamento da memória de curto prazo humana requerem que telas sejam mantidas simples, que sejam consolidadas telas com múltiplas páginas, que seja reduzido frequência de movimento de janelas e que tempo seja alocado para treinamento de códigos, mnemônicos e sequências de ações.

2.4.3 J. M. Christian Bastien e Dominique L. Scapin

Bastien e Scapin (1993) [1] propuseram oito “Critérios Ergonômicos para a Avaliação de Interfaces Humano-Computador”. Esses critérios também apresentam grande similaridade às heurísticas de Nielsen, mas eles chamam atenção para a importância de que códigos, quando usados, sejam significativos. Os critérios ergonômicos propostos são:

1. Controle:

Trata-se das formas disponíveis para aconselhar, orientar, informar, instruir e guiar usuários através de suas interações com um computador.

2. Carga de trabalho:

Diz respeito a todos os elementos da interface que tem papel na redução da carga perceptual e cognitiva dos usuários e no aumento da eficiência de diálogos.

3. Controle explícito:

Trata-se tanto do processamento feito pelo sistema de entradas explícitas do usuário, quanto no controle que os usuários têm sobre o processamento de suas entradas.

4. Adaptabilidade:

A adaptabilidade de um sistema diz respeito a sua capacidade de se comportar contextualmente e de se comportar de acordo com as necessidades e preferências do usuário.

5. Gestão de erros:

Trata-se de formas disponíveis de prevenir ou reduzir erros e de se recuperar deles quando ocorrem. Erros são definidos por Bastien e Scapin como sendo entradas inválidas de dados, formato incorreto de entradas de dados, sintaxe incorreta e outros.

6. Consistência:

Diz respeito a forma como escolhas de *design* (códigos, nome, formatos, procedimentos e outros) devem ser mantidos em contextos similares e devem ser diferentes em contextos distintos.

7. Códigos significativos:

Qualifica o relacionamento entre um termo e/ou um código e sua referência. Códigos e nomes têm significado para o usuário quando existe uma relação semântica forte entre tais códigos e os itens ou ações aos quais eles se referem.

8. Compatibilidade:

Diz respeito à compatibilidade entre as características dos usuários (memória, percepção, cultura, habilidades, idade, expectativas e outros) e tanto as características das tarefas quanto a organização da saída, entrada e diálogo para uma determinada aplicação. Também se trata da coerência entre ambientes e entre aplicações.

2.5 Desenvolvimento Ágil de Software

O Movimento Ágil é uma filosofia de desenvolvimento que busca alternativas a formas tradicionais de gerência de projetos. De acordo com Ambler (2001) [45], modelagem ágil é uma metodologia para *design* e documentação de sistemas baseados em *software*. É tanto um conjunto de boas práticas quanto uma coleção de valores e princípios para modelagem de *software*, que pode ser aplicado no desenvolvimento de programas de forma efetiva e com baixo custo.

De acordo com *Agile Alliance* (2017) [46], ser ágil significa criar e reagir a mudanças de modo a alcançar sucesso em meio a incertezas e ambientes turbulentos. Os valores e princípios a serem alcançados são aqueles presentes no Manifesto Ágil. Nele, dezessete desenvolvedores escrevem que se deve valorizar:

- Indivíduos e interações mais que processos e ferramentas
- *Software* em funcionamento mais que documentação abrangente
- Colaboração com o cliente mais que negociação de contratos

- Responder a mudanças mais que seguir um plano

Isto é, apesar de haver valor nos itens à direita das afirmações, deve-se valorizar os itens à esquerda mais. Essa filosofia busca acelerar o desenvolvimento de *software*, através da diferenciação deste de processos clássicos de engenharia e produção industrial. O primeiro princípio deixa explícita a necessidade de ferramentas que facilitem o desenvolvimento e a vida profissional do desenvolvedor, em detrimento de ferramentas que tomem tempo e esforço dos participantes de um projeto.

Logo, pode-se sumarizar a relação entre desenvolvimento ágil e as ferramentas de desenvolvimento usadas em tais ambientes como sendo restritas por requisitos de usabilidade. Isto é, a fim de realmente valorizar indivíduos e interações, é preciso desenvolver ferramentas que levem em consideração suas necessidades como seres humanos, de tal modo que elas sejam não só úteis, mas também agradáveis de usar. Assim, forma-se um paralelo entre usabilidade e o desenvolvimento ágil ou, mais especificamente, entre usabilidade e a criação de ferramentas para auxiliar o desenvolvimento ágil.

3 *Estudo do Estado da Arte*

Poucos aplicativos podem ser encontrados na loja de aplicativos do sistema *Android*, a *Google Play Store* [31], para criação de diagramas *UML*. Dentre eles, alguns oferecem funcionalidades genéricas, que permitem criar e editar também fluxogramas, *mind maps* e outros. Os aplicativos encontrados são ou incompletos ou oferecem mais do que o necessário. Todos são complexos e alguns possuem suporte limitado a gestos básicos comumente presentes em sistemas operados por tela sensível ao toque. Assim, é possível dizer que é pouco provável que qualquer um deles venha a ser adotado em um ambiente ágil de desenvolvimento.

3.1 *Diagrid*

O aplicativo *Diagrid* [23] é um editor gráfico para a criação de diagramas simples. Ele possui suporte para diagramas de classe, fluxogramas e, através de entidades genéricas, como retângulos, elipses e formas humanas, suporta outros diagramas quaisquer diversos.

A interface (figura 2) é simples e conta com uma área de edição e um menu superior com dois botões. Esses botões são voltados para a edição do diagrama. Ele possui ainda um botão para funções do sistema, como salvar e carregar projetos e exportar como arquivo *PNG*.

A interação do usuário com o aplicativo se dá principalmente através de botões. Dos botões no menu superior, um deles permite criar classes e comentários para diagramas de classe, enquanto o outro botão permite criar elementos genéricos, como retângulos e elipses e possui um submenu que oferece ainda mais possibilidades, como elementos típicos de fluxogramas: decisão, estado inicial, estado final e outros.

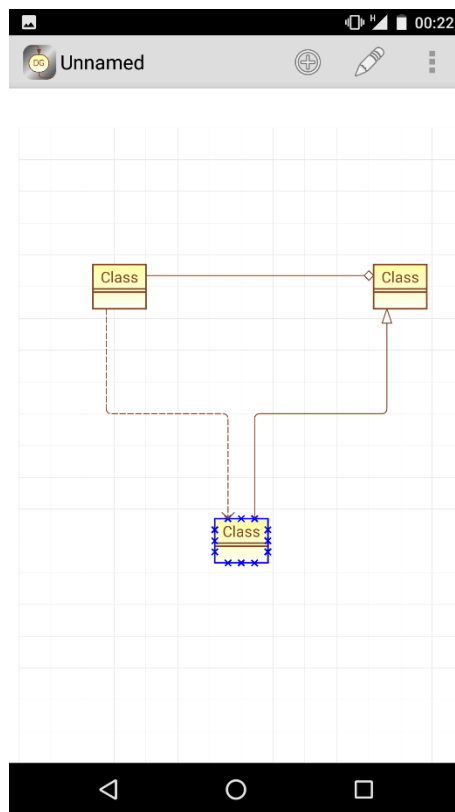
Figura 2: Tela inicial do Diagrid



Em se tratando das funcionalidades de edição de diagramas de classes, ao se clicar em uma classe, o segundo dos dois botões principais muda e permite editar a classe selecionada. Durante a edição da classe é possível mudar o nome dela, torná-la uma interface, uma classe abstrata e adicionar campos e métodos. O outro botão não muda de ícone, mas passa a permitir a criação de relações entre classes. Existem três opções: dependência, agregação e herança. Depois de escolher uma opção de relacionamento, basta clicar em alguma classe e a ligação será feita.

O fato de que um dos botões não muda de ícone, mas passa a executar funções diferentes pode ser visto como um problema de usabilidade. Tendo em vista a heurística de Nielsen sobre visibilidade do estado do sistema, o aplicativo não explicita tão bem sua mudança de estado. Uma alternativa melhor seria que os ícones fossem mais descritivos e refletissem a função do botão.

Figura 3: *Diagrid* com três classes e relacionamentos entre elas



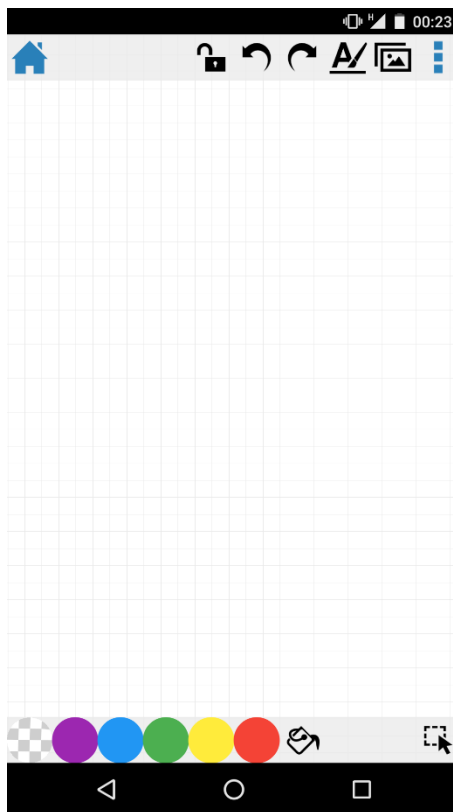
O *Diagrid*, apesar de simples e intuitivo, oferece suporte a poucos gestos, só permite mover o diagrama e mover as classes. Essas características o aproximam de um processo mecanizado de clicar em botões e o afastam de ferramentas físicas, como cartões e quadro branco, onde um diagrama é um desenho, o que vai contra o que Nielsen diz quanto à sua heurística sobre compatibilidade entre sistema e mundo real. Além disso, por ter diversas opções além do que é estritamente necessário para um diagrama de classes, o aplicativo não se adequa tão bem a uma metodologia ágil que preza pela ferramenta mais simples possível.

Por fim, o *Diagrid* também não oferece nativamente suporte a informações como multiplicidade e acesso. Descrever essas informações depende de incluir via texto a informação em questão, por exemplo, escrever “+” antes do nome de um campo ou método. Isso pode ser interpretado como sendo contrário à heurística de Nielsen sobre prevenção de erros, visto que toda essa liberdade ao usuário permite que ele, por exemplo, use símbolos inexistentes, como escrever “%” antes do nome de um campo. Apesar de isso não gerar erros no aplicativo, pode permitir que erros de digitação ocorram e pode dificultar o entendimento entre membros de uma equipe e, assim, dificultar o trabalho em equipe.

3.2 Draw Express

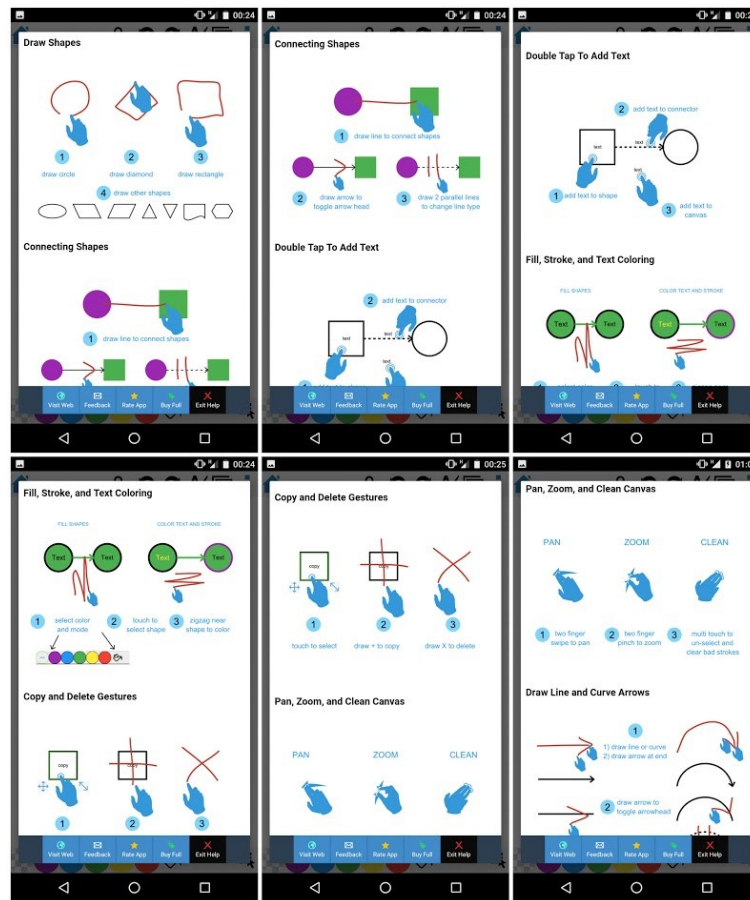
O aplicativo *Draw Express* [24, 25] é um editor de diagramas com suporte aos mais variados tipos de diagrama, como diagramas UML, *mind maps* e diagramas de redes de computadores. Sua tela inicial (figura 4) possui uma ampla área de edição e diversos botões. Os na parte superior realizam funções globais no aplicativo e mostram menus, enquanto que os da parte de baixo mudam a forma de interação com a área de edição.

Figura 4: Tela inicial do Draw Express



O diferencial deste aplicativo é o seu extensivo suporte a gestos (figura 5), que permitem não só navegar pela área de edição, mas criar elementos gráficos baseados no desenho realizado na tela, por exemplo, criar um elemento retangular, circular, uma elipse, um paralelogramo e outros ao desenhar uma dessas figuras. Permite copiar e colar elementos, conectar elementos com os mais diversos tipos de linhas, apagar elementos ao desenhar na tela um “x”, mudar o tipo de linha usada de traço contínuo para traço intermitente desenhando um “+” na tela e muitos, muitos outros gestos.

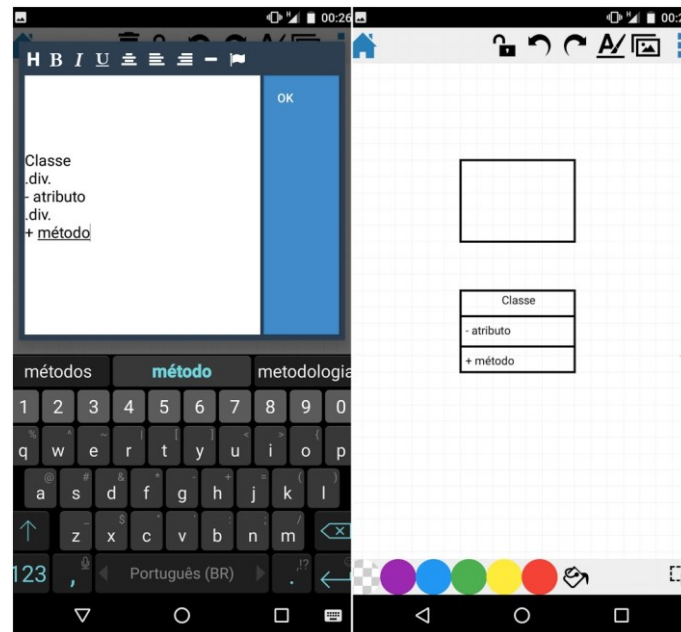
Figura 5: Alguns dos gestos que o Draw Express reconhece



Além de entradas via gestos, o aplicativo também possui botões para inserir formas, para mudar cores com que linhas e letras são desenhadas e que as formas são preenchidas e possui também botões para menus. Pode-se afirmar que a interface não é muito poluída, o que respeita a heurística de Nielsen sobre projeto minimalista e estético. Os gestos mais simples, que permitem desenhar formas, podem cumprir o que Nielsen propõe quando fala em flexibilidade e eficiência de uso, visto que esses gestos aceleram a edição do diagrama, eles funcionam como atalhos. Enquanto que usuários inexperientes podem se sentir mais confortáveis usando botões.

Para editar o texto dos elementos no diagrama, é necessário tocar duas vezes em um elemento. Ao fazer isso, uma janela é aberta, onde é possível escrever. Além de texto, é possível inserir marcações no texto (figura 6) para criar divisões na forma geométrica e assim, por exemplo, criar um retângulo típico de diagramas de classe UML, que possuem três campos: um para o nome da classe, um para atributos e um para métodos.

Figura 6: À esquerda: área de edição de texto do Draw Express. À direita: Exemplo de retângulo sem texto e retângulo resultante do texto à esquerda



Todas essas funcionalidades tornam o aplicativo altamente complexo. Memorizar todos os gestos possíveis é uma tarefa difícil e consultar a sessão de ajuda torna-se uma ação frequente. Isso acaba alongando a curva de aprendizagem em vez de minimizá-la. O usuário acaba tendo a necessidade de memorizar diversos gestos pois eles realizam funções importantes no aplicativo. Alguns deles possuem alternativas com botões, mas, para outros, como criação de setas e mudar o tipo de linha usado (contínua ou intermitente), não foram encontradas alternativas. Isso acaba indo contra a heurística de Nielsen de reconhecimento em vez de lembrança, já que as informações sobre os gestos suportados ficam ocultas durante o uso das principais funções do aplicativo.

Conclui-se então que o aplicativo não se restringe ao necessário e, mais ainda, exige trabalho extra para desenhar uma classe como em diagramas *UML* típicos. Em vez de oferecer nativamente retângulos com áreas específicas para nome, atributos e métodos, ele exige que se insiram marcações explícitas para dividir o retângulo. Então, pode-se afirmar que é pouco provável que o *Draw Express* venha a ser usado em um ambiente de desenvolvimento ágil.

3.3 *HandyUML*

De todos os aplicativos testados, o *HandyUML* [26, 35] é o mais completo em se tratando de desenvolvimento de diagramas de classe *UML* de forma gráfica. Além dos

de classe, ele possui suporte a diagramas genéricos e diagramas de caso de uso. Ao criar um diagrama, é possível escolher qual se vai utilizar e, assim, limitar a ferramenta e sua complexidade. A figura 7 mostra a tela inicial do aplicativo quando se tem aberto um diagrama vazio.

Figura 7: Tela inicial do HandyUML

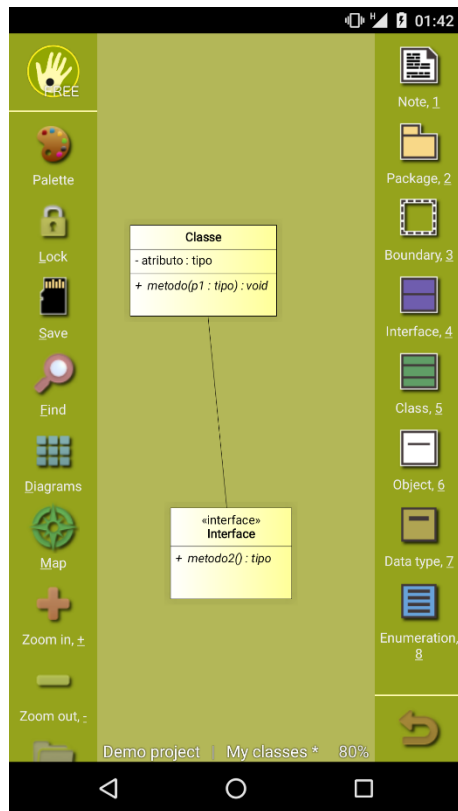


Em se tratando da funcionalidade de editor de diagramas de classes, o aplicativo realiza todas as suas funções através de botões. O único gesto reconhecido é o de mover a área de edição. Até mesmo *zoom* é feito com um botão (figura 8). Para criar uma classe é necessário tocar na área de edição do diagrama, fazendo surgir um menu lateral. Dentre as diversas opções de elementos, uma delas é a de criar uma classe. Outras opções incluem criar interfaces, objetos, pacotes e enumerações.

Ao se criar uma classe, um retângulo vazio já com separadores é criado. Tocando nele surge um menu à direita com botões que permitem editar as propriedades da classe, conectar a classe a outras ou a ela mesma, selecionar, copiar ou remover a classe e outras opções gerais. Tocando no botão para editar propriedades, surgem novos botões que permitem alterar a aparência (como a cor) do elemento gráfico que representa a classe; permitem alterar propriedades gerais, como nome e tipo de classe (geral, abstrata ou final); permitem alterar atributos, incluindo criar, remover e editar

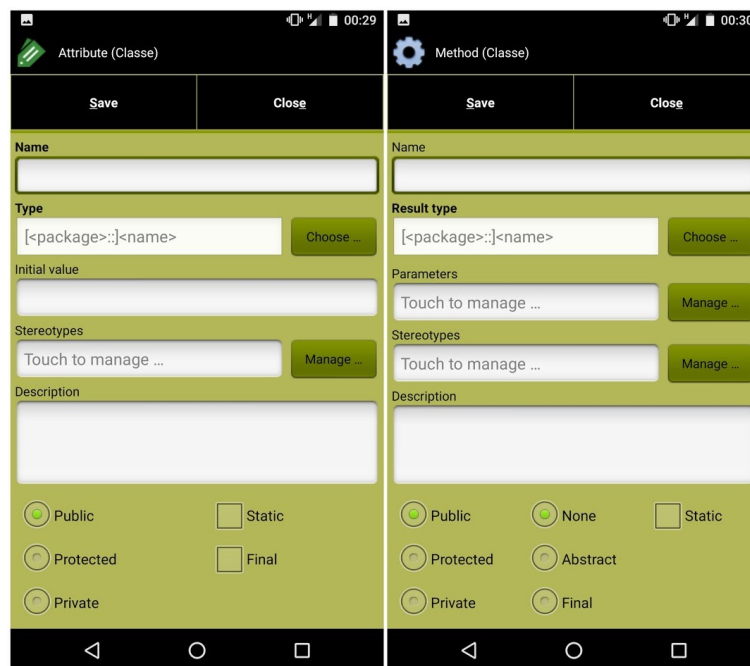
atributos na classe, e permitem alterar métodos, incluindo criar, remover e editar métodos.

Figura 8: Botões do HandyUML



As formas como se adicionam atributos e métodos são similares. Trata-se de uma tela com campos para nome, tipo, parâmetros (no caso do método), descrição e modificadores como público, privado, abstrato, final e outros (figura 9). Essa tela pode ser fechada salvando as alterações ou descartando-as.

Figura 9: Telas para adição de atributos e de métodos no HandyUML

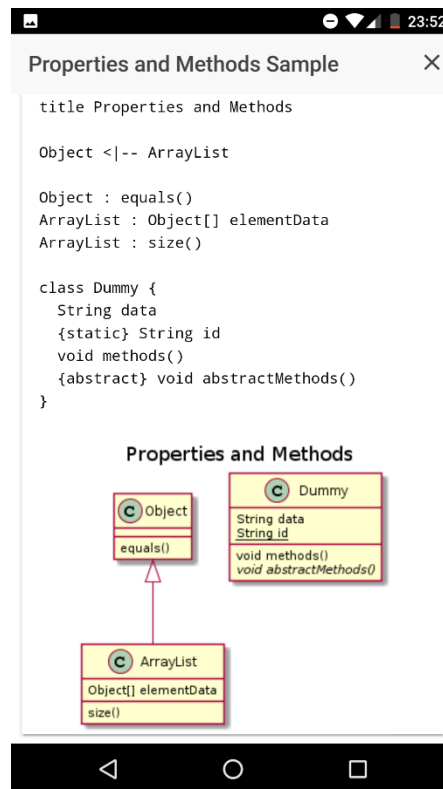


Pela sua complexidade, falta de suporte a gestos e necessidade de diversos toques na tela para realizar qualquer tarefa, o *HandyUML* é possivelmente o aplicativo menos indicado para desenvolvimento ágil. Assim como o *Diagrid*, por reconhecer poucos gestos, usar o *HandyUML* acaba se tornando uma tarefa mecanizada de clicar em botões, o afastando de metáforas do mundo real. Também como no caso do *Diagrid*, isso vai contra a heurística de Nielsen para compatibilidade entre sistema e mundo real. De certa forma, o *HandyUML* acaba parecendo um porte de editores clássicos em computadores pessoais para um ambiente móvel com tela sensível ao toque, em vez de um aplicativo novo que explore as metáforas dessa plataforma diferente.

3.4 *iPlantUML*

O *iPlantUML* [43] é um aplicativo que permite criar diagramas UML diversos, como de classe e de sequência, a partir de texto. Seguindo numa direção diferente das demais ferramentas testadas, o *iPlantUML* não possui um editor gráfico propriamente dito, mas permite criar diagramas a partir de notação textual (figura 10). A partir do texto, o aplicativo gera uma imagem que mostra o diagrama. Por se tratar de uma imagem, não é possível editá-la diretamente como um diagrama. A navegação também é limitada, não é possível mover o diagrama e nem utilizar gestos como o de pinça para aproximar ou distanciar.

Figura 10: Exemplo de descrição textual de diagrama de classes e a imagem gerada a partir do texto



O método adotado pelo *iPlantUML* de realizar a criação e edição via texto o assemelha a terminais em computadores pessoais, onde a interação acontece através de comandos textuais. Se pelo lado positivo esse método pode ser bastante preciso, rápido e próximo da metáfora de programação, a qual também é realizada com comandos textuais, a visualização do sistema fica debilitada durante a escrita do diagrama. Em um diagrama simples, que seria mais fácil de visualizar como texto, a vantagem de precisão e possível agilidade não é tão grande quando comparado a opção de criar o diagrama graficamente. Em um projeto maior, a precisão do método e a potencial dificuldade em montar o diagrama graficamente (devido à grande necessidade de toques na tela, por exemplo) podem não compensar o esforço em escrever e ter de encontrar as definições e usos de cada entidade no texto.

Isto é, a forma como o *iPlantUML* funciona vai contra duas heurísticas de Nielsen. A primeira é a de visibilidade do estado do sistema, uma vez que a única descrição do estado do sistema é o código sendo escrito para gerar o diagrama depois. A segunda é a de reconhecimento em vez de lembrança, visto que o usuário precisa recordar de todos os códigos textuais e que entidades gráficas eles vão gerar depois. Isso

acaba aumentando a dificuldade em utilizar o aplicativo. Ele deixa de ser intuitivo e, consequentemente, deixa de ser uma opção realmente viável para desenvolvimento ágil.

Logo, pode-se concluir que, apesar de sua precisão, o método adotado pelo *iPlantUML* requer memorizar comandos, como em sistemas de terminal em computadores pessoais, possui baixa visibilidade do estado do sistema e quase nula interatividade entre usuário e diagrama pronto. Assim, a criação de um projeto torna-se um processo de programação, de escrever um código que representa um diagrama. Entretanto, o desenho de diagramas pode acontecer em momentos do desenvolvimento de *software* que precede a escrita de código propriamente dito.

4 *Desenvolvimento*

Após o estudo dos aplicativos disponíveis no mercado para *Android*, a fim de averiguar os pontos fortes e os pontos fracos das ferramentas já existentes, e tendo em vista os objetivos do trabalho, são levantados requisitos funcionais e não-funcionais da aplicação. Os requisitos funcionais tratam das funcionalidades típicas de editores de diagramas de classes. Os requisitos não-funcionais tratam da necessidade de se criar uma ferramenta com alta usabilidade e que favoreça o desenvolvimento ágil. Os requisitos foram escolhidos tendo em vista a necessidade de se criar uma ferramenta simples, porém completa.

O desenvolvimento do código do trabalho se dá na linguagem Java, que é a linguagem oficial de desenvolvimento para o sistema *Android* [27].

4.1 *Requisitos Funcionais*

- O sistema deve permitir a criação, alteração e exclusão de classes
- Todas as classes possuem um nome informado pelo usuário ou automaticamente atribuído a elas pelo sistema
- Todas as classes podem ser de uma categoria, como interface, abstrata ou ser uma classe concreta simples
- O sistema deve ser capaz de representar graficamente as classes presentes no diagrama
- O sistema deve permitir mover as entidades gráficas que representam as classes
- O sistema deve permitir a criação, alteração e exclusão de relacionamentos entre classes, os quais são associação, agregação, composição, dependência e herança [3, 6, 42]
- Todos os relacionamentos podem ter um nome definido informado pelo usuário
- Todos os relacionamentos que não sejam de herança têm multiplicidade informada pelo usuário
- A multiplicidade padrão é de 1 para 1
- O sistema deve ser capaz de representar graficamente os relacionamentos entre classes e diferenciar visualmente cada tipo de relacionamento
- O sistema deve ser capaz de mover e redimensionar as entidades gráficas que representam os relacionamentos automaticamente caso as classes às quais elas se conectam sejam movidas
- O sistema deve permitir a criação, alteração, exclusão e reordenamento de operações nas classes
- Todas as operações possuem um nome informado pelo usuário ou automaticamente atribuído a elas pelo sistema
- Todas as operações podem ter visibilidade, parâmetros com tipos e retorno

- O sistema deve permitir a criação, alteração, exclusão e reordenamento de atributos nas classes
- Todos os atributos possuem um nome informado pelo usuário ou automaticamente atribuído a eles pelo sistema
- Todos os atributos podem ter tipo e visibilidade
- O sistema deve ser capaz de representar graficamente atributos e operações das classes e as propriedades de cada um
- O sistema deve permitir nomear e renomear classes, relações, atributos e operações usando teclado
- O sistema deve permitir selecionar e *desselecionar* múltiplas classes ou múltiplos componentes de uma classe
- O sistema deve permitir navegar pela área de edição, movendo as entidades gráficas nela
- O sistema deve permitir a criação, alteração, exclusão e busca de projetos
- Todos os projetos possuem um nome único que os identifica atribuído pelo usuário ou automaticamente pelo sistema
- Um projeto pode conter múltiplos diagramas
- Todos os diagramas possuem um nome único que os identifica atribuído pelo usuário ou automaticamente pelo sistema
- Todos os elementos de um diagrama devem ser unicamente identificados por um identificador
- O sistema deve ser capaz de salvar e carregar projetos no dispositivo

4.2 *Requisitos Não-Funcionais*

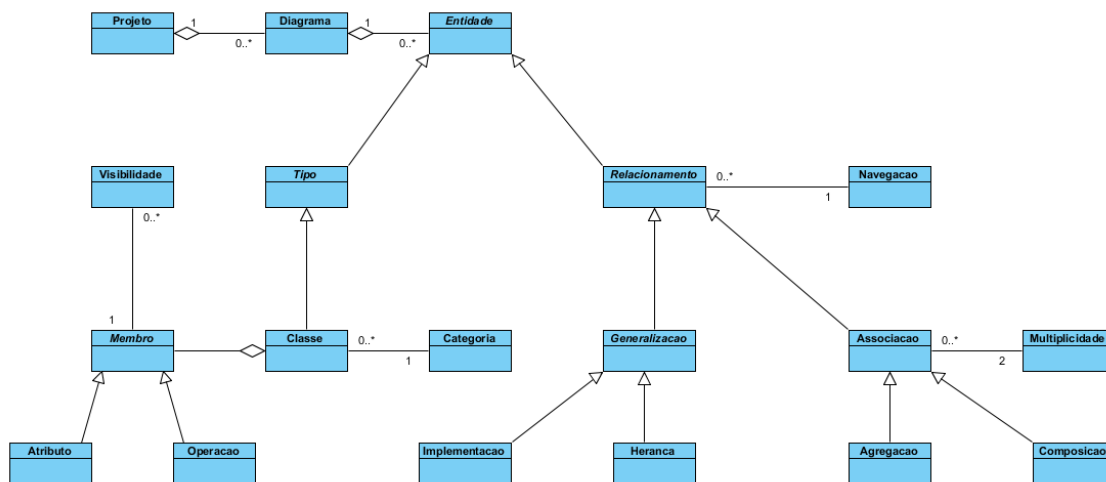
- O sistema deve ser amigável e intuitivo, tal que alguém que sabe o que é um diagrama de classes não precise de treinamento nem de mais do que alguns minutos para saber usar a maioria das funcionalidades do sistema. Isto é, buscase alto nível de usabilidade tendo em vista principalmente as heurísticas de usabilidade de Nielsen, as quais são as mais conhecidas e não perdem em completude para outros conjuntos bem conhecidos de heurísticas
- O sistema deve ter uma interface gráfica escalável, para comportar sistemas com telas menores, como celulares e *tablets*, e sistemas com telas muito maiores, como uma televisão
- O sistema deve usar pouco processamento e memória, visto que será executado em ambientes com recursos computacionais limitados
- O sistema deve ser energeticamente eficiente, já que potencialmente será executado em ambientes com restrições de bateria
- O sistema deve ser usado com telas sensíveis a múltiplos toques

4.3 *Modelo do Sistema*

O núcleo da lógica do aplicativo se encontra no modelo do sistema. O modelo contém classes como *Projeto*, *Diagrama*, *Classe* e *Relacionamento* (figura 11). Essas

classes são responsáveis por guardar e impor limitações aos seus dados, por exemplo a classe *Relacionamento* contém referências para as duas classes do diagrama que se relacionam. As classes no modelo que representam elementos em um diagrama, como as listadas anteriormente, também são encarregadas de fornecer uma representação textual do elemento. Essa representação é usada pelo módulo visual do sistema para representar graficamente cada elemento.

Figura 11: Diagrama de Classes do Modelo do AgileClass



Para se acessar as funcionalidades do modelo é necessário utilizar as classes *Projeto* e *Diagrama*. Como colocado nos requisitos funcionais, um projeto possui um nome único que o identifica e pode possuir zero ou mais diagramas. Um diagrama, por sua vez, possui um nome único que o identifica dentro do projeto e possui zero ou mais classes. Assim, dado um projeto, é possível obter um de seus diagramas através do nome do diagrama. A classe *Diagrama* é a encarregada de oferecer as funcionalidades de adicionar classes e relacionamentos, bem como de adicionar atributos e operações a classes. O usuário do modelo do sistema não possui acesso aos elementos do diagrama diretamente.

Para acessar classes e relacionamentos em um diagrama é necessário informar o identificador do elemento. Esse identificador é atribuído durante a criação da entidade e é único dentro de um diagrama.

Por fim, é importante notar que o modelo do sistema não se responsabiliza em tratar da posição dos elementos dentro do diagrama. Essa funcionalidade é entendida como sendo puramente visual e, pois, é deixada para o módulo visual do sistema. Ou

seja, o modelo vê um diagrama de classes como um grafo, como relações entre classes, mas sem se preocupar em como elas serão visualizadas.

4.4 Interação com o Usuário

4.4.1 Telas

O *AgileClass* é composto por cinco telas principais: a inicial, a de edição de classes, a de edição de atributos em classes, a de edição de operações em classes e a de edição de relacionamentos.

4.4.1.1 Tela Inicial

A primeira visão que o usuário tem do sistema é a tela inicial. É nela que ele realiza as principais e mais básicas funções do aplicativo, como mostram as figuras 12 e 14, que exibem a tela inicial do *AgileClass* na vertical e na horizontal respectivamente. Essa tela é composta por três áreas estáticas, como mostra a figura 13: a barra do aplicativo (marcada como 1) [13], a lista de botões (marcada como 2) e a área de edição (marcada como 3).

Figura 12: Tela inicial do AgileClass

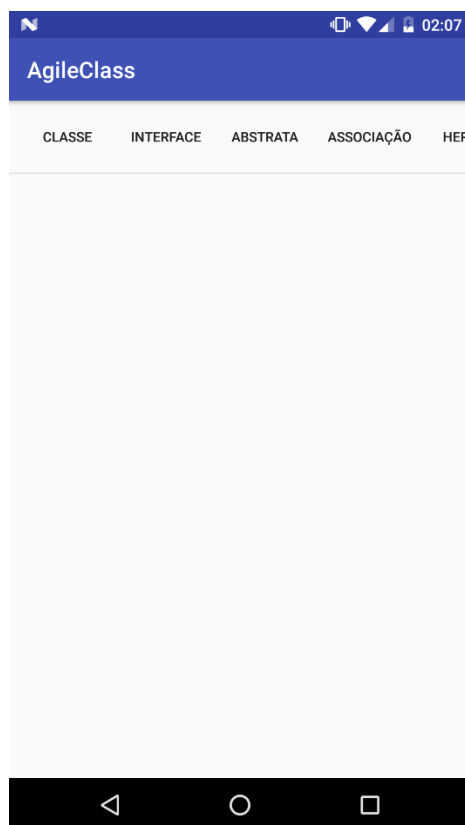
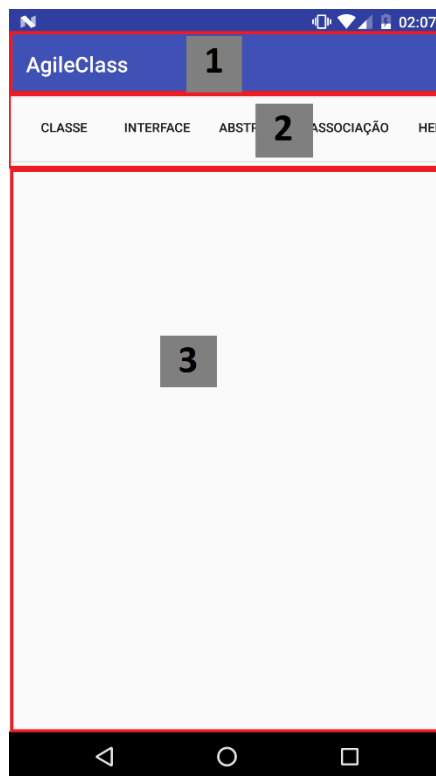


Figura 13: Áreas principais da tela inicial



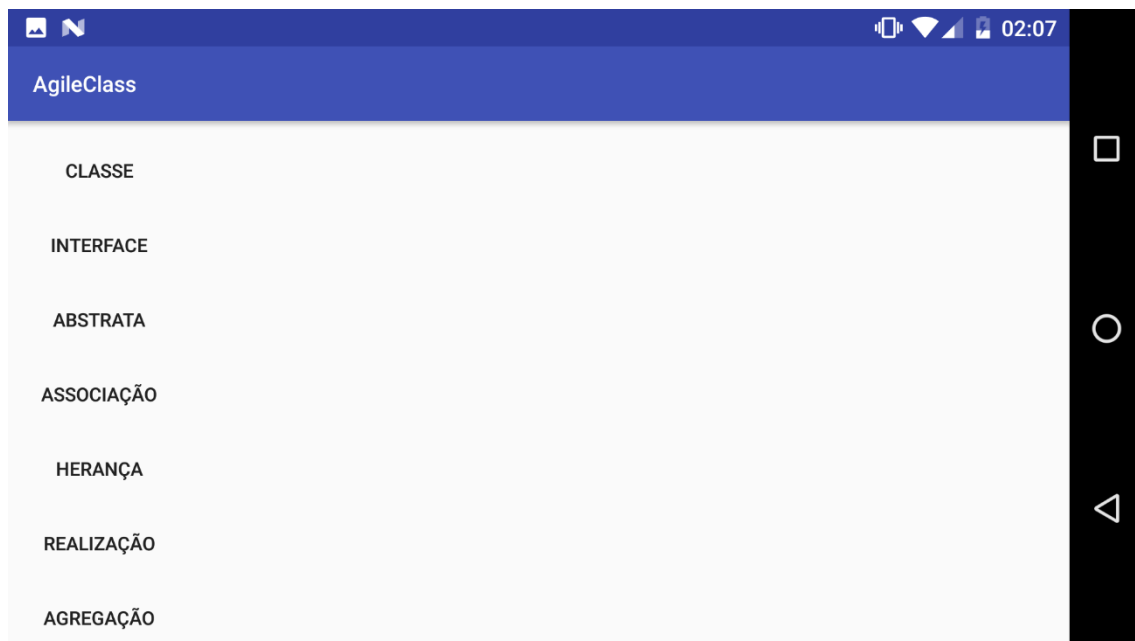
A barra do aplicativo é uma área comumente encontrada em aplicativos para *Android*, é criada automaticamente pelo ambiente de desenvolvimento. Nela encontra-se o nome do aplicativo e pode conter um menu principal, ou menu do aplicativo, onde constam funcionalidades que afetam o sistema por inteiro.

A lista de botões é uma lista horizontal (no modo retrato) ou vertical (no modo paisagem) que contém os botões que alteram o modo de atuação da área de edição. Isto é, cada botão representa uma entidade e tocar em um certo botão e depois na área de edição criará a entidade selecionada. Por exemplo, se o botão “classe” for selecionado, um toque na área de edição criará uma classe.

A escolha por colocar esses botões na lista e não na barra do aplicativo, como é comum em outros aplicativos, se deve pela busca por um sistema escalável quanto ao tamanho da tela. Colocando os botões em uma lista, é possível acrescentar suporte a *scroll* na lista, isto é, caso não seja possível acomodar todos os botões na tela, eles podem ser acessados deslizando a lista para a posição desejada. A barra de aplicativo dispõe de pouco espaço e não seria possível colocar todos os botões para todas as entidades suportadas em uma tela de tamanho normal ou menor. Uma alternativa seria seguir o modelo do *Diagrid*. Nele, como mostra a figura 2, a barra do aplicativo contém

três botões, os dois primeiros mostram um menu que oferece botões para criação das entidades. Isso, porém cria a necessidade de o usuário realizar mais um toque.

Figura 14: Tela inicial na horizontal

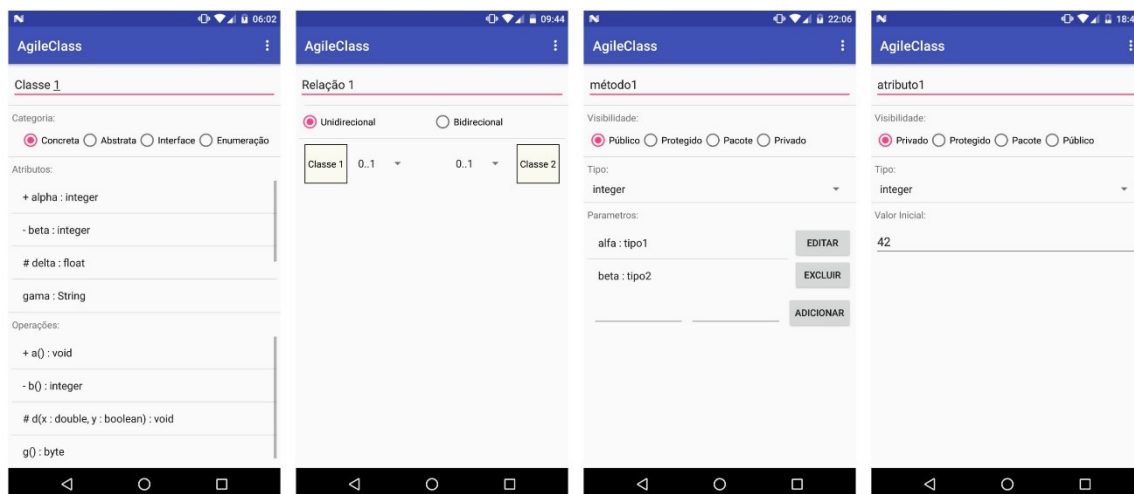


4.4.1.2 Telas de Edição

A principal função das telas de edição, como o nome propõe, é permitir a edição de componentes do diagrama, como mostrado na figura 15. Dentre essas telas, a principal é a de edição de classes. Seus componentes principais são o campo para edição do nome da classe, botões de rádio para modificar o tipo de classe (concreta, interface, abstrata ou enumeração) e duas listas, uma com os atributos e outra com as operações dela. Nas duas listas, seus elementos são botões que, quando clicados, exibem as telas de edição de atributo ou operação.

Para editar atributos e operações, as telas projetadas são similares. Elas contêm campos onde o usuário pode informar o nome do campo, sua visibilidade e informações específicas, como valor inicial para atributos ou parâmetros para operações.

Figura 15: Exemplos de telas de edição de classe, relacionamento, operação e atributo, respectivamente



Por fim, a tela de edição de relacionamentos permite editar as linhas e setas que ligam classes. Os itens mostrados, isto é, as características do relacionamento que podem ser alterados dependem do tipo de relacionamento. Por exemplo, uma relação de herança não possui multiplicidade, logo, sua tela de edição não possui esse campo.

Pode-se dizer que as telas de edição representam conjuntos de ações, como proposto por Shneiderman quando ele afirma que o desenvolvedor deve projetar diálogos para que tenham final. Isto é, cada tela de edição representa uma sequência agrupada de ações que o usuário deve tomar para deixar cada classe e relacionamento como ele achar melhor. Com o uso delas, também se melhora a visibilidade do estado do sistema, visto que a tela do dispositivo é totalmente tomada pela tela de edição vigente, em concordância com as recomendações tanto de Shneiderman, quanto de Nielsen e Bastien e Scapin.

4.4.2 Gestos

A interação com o usuário se dá principalmente através de toque na tela sensível ao toque. O sistema é capaz de reconhecer toques simples, que remetem a cliques com um *mouse*, a gestos com os dedos. A *API Android* considera alguns gestos como sendo os mais básicos e oferece algum suporte para implementação padrão deles [21]. Esses gestos incluem toque, toque duplo, *scroll* e *fling* (arremesso). Quando a aplicação registra um objeto como sendo Observador de Gestos, ele passa a receber eventos, isto é, chamadas de métodos, quando algum desses gestos é detectado pelo sistema

operacional. Interceptar esses eventos permite então que o aplicativo seja programado a reagir de acordo. A *API* também permite registrar um *View* como sendo um *Observador de Toque Duplo*, isso permite a esse *View* realizar ações diferentes caso ele receba um toque simples ou um toque duplo. A identificação de o que configura um toque simples e o que configura um toque duplo ficam a cargo do sistema operacional, mas pode ser sobrescrito pelo desenvolvedor.

No *AgileClass* foram implementados gestos comumente usados em outras aplicações e com significados bem conhecidos. Por exemplo, se o usuário realiza um gesto de *scroll* é possível afirmar que ele deseja que o conteúdo na tela deslize em uma determinada direção. Com isso em mente, quando um gesto desse tipo é realizado na área de edição de diagramas, o diagrama é deslocado na direção desejada. Quando o usuário toca em algum elemento, pode-se afirmar que ele deseja selecionar aquele elemento e interagir com ele especificamente. Dessa forma, ao tocar em uma classe ou relacionamento, essa entidade é selecionada. Quando uma entidade é selecionada, ela pode ser movida.

O mais importante gesto, porém, é o de criação de entidades. O usuário precisa tocar em um dos botões da lista de botões e então tocar na área de edição e isso criará uma entidade na tela. A entidade criada dependerá do botão selecionado. Essa forma de interação segue exemplo de outros aplicativos, como *Facebook*. Nesse aplicativo, é possível editar imagens adicionando figurinhas a elas. Para tanto, o usuário primeiro seleciona uma figurinha com um toque sobre a desejada e depois toca na tela onde ele deseja que a figurinha fique [8]. Logo, esta forma de uso do *AgileClass* se adequa a metáforas já exploradas em outros aplicativos no mercado e aumenta a probabilidade de ela ser bem aceita e rapidamente aprendida.

Como é comum no ambiente *Android*, um toque longo na tela exibe um menu de contexto sobre o elemento que recebeu o toque longo [34]. No *AgileClass*, se uma classe receber um toque longo, um menu surge que possibilita editá-la ou removê-la. A decisão por seguir esse modelo foi tomada por se buscar manter o *look and feel* do sistema *Android*. Essa decisão é compatível com Nielsen quando ele fala sobre consistência e padrões, visto que está se buscando seguir uma convenção da plataforma usada. Assim, pode-se afirmar que um usuário acostumado com o sistema operacional *Android* aprenderá mais rapidamente a usar um aplicativo se o estilo de uso deste se mantiver consistente com o estilo de uso daquele.

Para este projeto, porém, também foram implementados alguns gestos pouco usuais, que realizem funções da interface de forma mais ágil, funcionando como um atalho. Por exemplo, um toque duplo na área de edição permite centralizar a tela no seu ponto central original. Esse gesto alinha-se à heurística de Nielsen sobre flexibilidade e eficiência de uso, pois se trata de um acelerador, isto é, um atalho.

4.4.3 Alternativas

Uma alternativa considerada para a forma de adição de classes e relacionamentos é, seguindo o exemplo do aplicativo *Draw Express*, explicado anteriormente, utilizar desenhos na tela. Isto é, para se adicionar uma classe ao diagrama, o usuário desenharia um retângulo (figura 4), para se adicionar uma relação entre duas classes, o usuário desenharia uma linha ou seta. Essa forma de interação se aproximaria de metáforas físicas, como se a área de edição fosse um quadro-negro ou uma folha de papel. Nesses casos, um diagrama é criado como um desenho.

A escolha por não utilizar esse modelo se deu principalmente pelo choque entre gestos de desenho e gestos comuns reconhecidos por aplicativos, como o de *scroll*. Isto é, ao deslizar o dedo sobre a tela, o aplicativo teria de distinguir entre o desejo do usuário de desenhar e o desejo de navegar pela interface. Uma forma de realizar isso seria criar um “modo de edição”, onde o gesto de *scroll* é interpretado como um desenho, como um pincel. Mas isso colocaria um passo extra no processo criativo, o de selecionar o modo desejado e alternar entre eles.

4.4.4 Componentes Visuais

4.4.4.1 Atividade Principal

Atividades, como explicado na seção 3.1.1, encapsulam os principais componentes gráficos e comportamento do sistema. A atividade principal é responsável pela tela inicial do aplicativo e, no caso do *AgileClass*, comanda a principal funcionalidade do programa: a edição de diagramas de classes.

No *AgileClass*, a atividade principal possui o mesmo nome do aplicativo. Ela é responsável por definir, em sua criação, o layout da janela a partir do arquivo *XML* escrito com auxílio do *Android Studio*, o *IDE* oficial para desenvolvimento para *Android* [16]. Ela também é responsável por criar classes *Java* para os componentes

visuais definidos no arquivo *XML* de *layout*, isto é, criar classes como *Button* para cada botão definido. Isso é feito para que se possa controlar os componentes via código, integrando-os ao restante do sistema.

Outra função importante da atividade é conectar os componentes visuais com os eventos do sistema. A atividade principal é responsável por registrar o *View* da área de edição como um observador de gestos. Se isso não for feito, a área de edição só seria capaz de reagir a interações simples.

Por fim, a atividade principal também é encarregada de controlar o modelo. Ela controla o projeto sendo selecionado e o diagrama atualmente sendo editado. É através da atividade principal que toques em botões se traduzem em ações no modelo, como adicionar e remover classes. Para tal, ao receber um toque na tela, a atividade principal encaminha o evento de toque para o componente gráfico ocupando a área tocada. Se o toque criar uma nova classe ou relacionamento, a atividade principal, que contém referência direta para o objeto diagrama atualmente sendo editado, entrega essa referência para o objeto que representa a área de edição para que este possa criar uma nova classe ou relacionamento no diagrama. O resultado disso é um identificador numérico que permite acessar essa nova entidade no diagrama. De outro modo, caso as ações do usuário exclua ou edite uma classe, a entidade gráfica que representa a classe no diagrama utiliza o identificador numérico que recebeu durante a criação da classe para acessá-la no objeto diagrama.

4.4.4.2 Área de Edição

A área de edição (área 3 da figura 13) é a parte da atividade principal em que são desenhadas as representações gráficas das classes e relações. A classe que encapsula essas funcionalidades estende a classe *View* e é registrada pela atividade principal como uma observadora de gestos, isto é, ela é capaz de se desenhar na tela e de interagir de forma complexa com o usuário.

De modo a modelar as diferentes formas que o processo de desenvolvimento do diagrama assume, a área de edição possui um modo de operação. Esse modo determina se, ao receber um toque, uma classe ou uma relação entre classes deve ser criada, ou ainda se nada deve acontecer. A atividade principal tem um papel importante nesse processo que é o de integrar os botões do menu de edição com a área de edição em si.

Uma das funções mais importantes pela qual que a área de edição é responsável é a de navegação pelo diagrama. Ela é encarregada de se redesenhar quando o usuário arrasta o dedo pela tela, isto é, quando o usuário move o diagrama, a classe precisa identificar essa ação e reagir de acordo.

A classe *Área de Edição* contém estruturas de dados que guardam os *Views* que representam classes e relações. Quando uma classe é adicionada no modelo, um *View* para representá-la graficamente é criado e armazenado em uma estrutura de dados. Essa estrutura é acessada diversas vezes para verificar o tamanho das entidades e requisitar que se desenhem na tela, por exemplo. Quando há necessidade, por exemplo quando o aplicativo é minimizado, essas listas são salvas e posteriormente recarregadas para que o estado atual seja mantido.

Por fim, a Área de Edição é incumbida de organizar as classes e relacionamentos na tela e requisitar que estes se desenhem. Isso é feito de forma relativamente simples. Cada classe é associada a uma posição no espaço e a área de edição limita o *Canvas* de desenho no qual a classe vai se desenhar. Algo similar acontece com os relacionamentos, mas a parte do *Canvas* que eles recebem é determinada pelas classes às quais eles dizem respeito.

4.4.4.3 Classe

Uma classe, do ponto de vista gráfico, é um retângulo com três seções, onde texto é escrito e que representa visualmente uma classe do diagrama. Na primeira seção é escrito o nome da classe, na segunda seção são escritos os atributos e na terceira os métodos da classe, como mostram os exemplos nas figuras 16 e 17.

Figura 16: AgileClass com algumas classes de exemplo com a Classe 3 selecionada

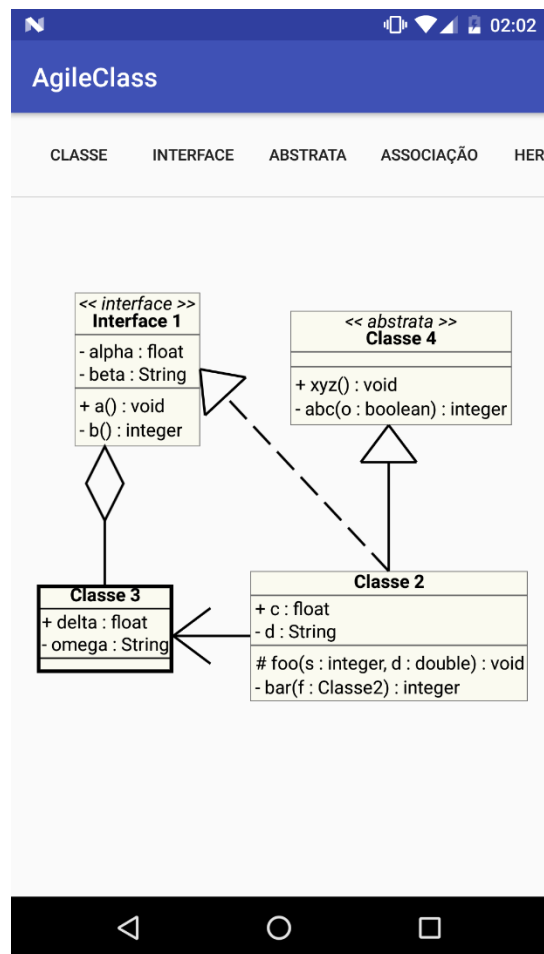
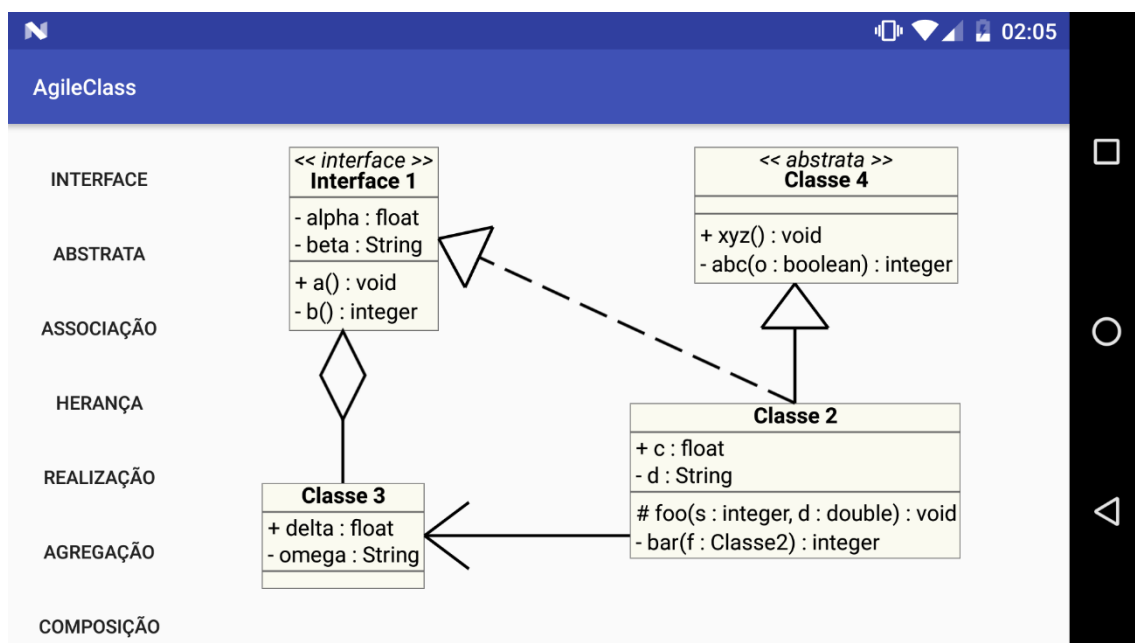


Figura 17: Tela inicial na horizontal com exemplos



Cada classe é responsável por lidar com os toques que recebe do usuário. Por vezes, porém, a classe precisa retornar o controle para a área de edição, visto que a área ocupada pela entidade visual está contida na área ocupada pela área de edição.

Finalmente, é importante ressaltar que a classe gráfica não possui referência direta à classe do diagrama. A entidade gráfica possui o identificador numérico com o qual acessa as informações da entidade no modelo. Isto é, a classe *Diagrama* funciona de forma similar a uma fachada e ela oferece uma interface que permite editar classes e relacionamentos dentro de um diagrama através de chaves. Cada objeto que representa graficamente uma classe do diagrama guarda consigo essa chave que o permite acessar a classe que lhe diz respeito dentro do diagrama. Isso diminui o acoplamento entre modelo e visão com um custo pequeno de o modelo ter de traduzir o identificador em um objeto em si.

4.4.4.4 Relacionamento

Similarmente a classes do diagrama, relacionamentos são representados com entidades gráficas através da classe *Relacionamento* que estende a classe *View*. Ainda como a anterior, o relacionamento é responsável por lidar com os toques que recebem, mas pode precisar da ajuda da área de edição que o encapsula. Por fim, como a classe gráfica, um relacionamento não possui referência para seu representante no modelo, mas possui para as classes gráficas a que ele se liga.

4.5 Compatibilidade

Por fazer uso da linguagem *Java*, uma linguagem que busca ser inerentemente independente de máquina e portátil [7], desenvolvedores para o sistema *Android* não precisam se preocupar com a forma como seu código será executado. Os únicos componentes que precisam de cuidados específicos quanto à “máquina” que executa o aplicativo são os componentes visuais. Esses podem mudar a cada nova versão do ambiente de desenvolvimento *Android*.

A fim de tirar proveito da portabilidade da linguagem *Java* e de mitigar os efeitos das mudanças na *API* do *Android*, o desenvolvimento deste projeto deu prioridade, especialmente no modelo do projeto, a componentes padrões da linguagem *Java*. Esses componentes estão nativamente disponíveis no ambiente de desenvolvimento e é possível afirmar que serão constantes através do tempo, isto é, não

sofrerão mudanças que comprometam sua assinatura, lógica e/ou definição no ciclo de vida próximo da linguagem *Java*.

Por outro lado, o sistema *Android* está em constante mudança, tanto sua *API* quanto interface gráfica [11, 40]. A fim de desenvolver código que seja executado de forma consistente através das múltiplas versões do sistema, é recomendado o uso das bibliotecas de suporte que a *API* oferece. Contudo, mesmo usando componentes presentes na biblioteca de suporte, algumas funcionalidades precisam ser adaptadas pelo desenvolvedor para diferentes versões da *API*. Por exemplo, como mostrado no tutorial oficial do ambiente de desenvolvimento para implementar o gesto de “arremesso” (“*fling*” em inglês). Como mostra a figura 18, a recomendação é escrever código específico para cada versão e verificar em tempo de execução qual a versão do sistema sendo executada e então executar o código apropriado.

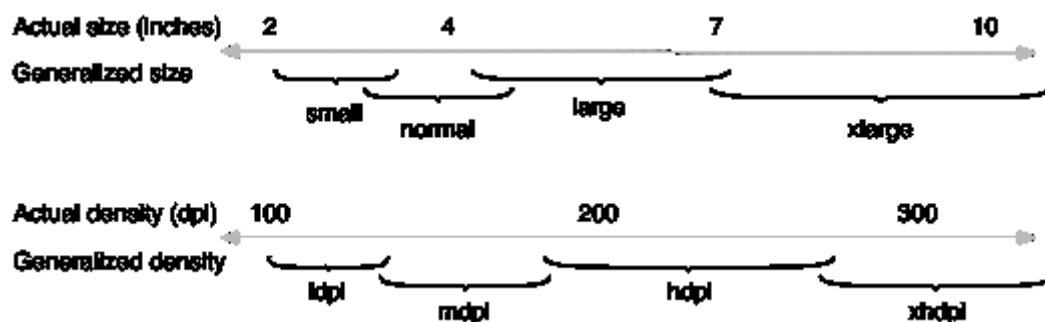
Figura 18: Código que verifica a versão do sistema Android sendo executada em tempo de execução

```
if (Build.VERSION.SDK_INT >= 11) {  
    scrollAnimator.setDuration(scroller.getDuration());  
    scrollAnimator.start();  
}
```

4.6 Interface Escalável

O sistema *Android* é utilizado em uma grande gama de dispositivos e suporta diversos tipos de *hardware*. A fim de oferecer uma interação com o usuário constante, o desenvolvedor deve considerar diferentes tamanhos de tela, dentre outras possíveis especificidades de *hardware*. A fim de sumarizar a miríade de possíveis variações de tamanho, o ambiente de desenvolvimento separa as telas em quatro grupos: pequena, normal, grande e extragrande. Essas classes representam intervalos de polegadas do tamanho da tela, como mostra a figura 19.

Figura 19: Ilustração de como o Android mapeia tamanhos e densidades reais em tamanhos e densidades generalizadas



Os componentes estáticos de um *layout* podem ser descritos em um arquivo *XML*. Esse arquivo é guardado em uma subpasta do diretório “*layout*” do projeto. O ambiente de desenvolvimento *Android* oferece nativamente uma forma de utilizar esses arquivos *XML* adaptando-os para diferentes tamanhos de telas e diferentes densidades de píxeis. Não só para telas diferentes, mas também orientações diferentes. Dessa forma o desenvolvedor pode desenhar sua aplicação tendo em vista diferentes formas como ela vai aparecer na tela e o sistema operacional se encarrega de realizar a transição de uma para outra.

A forma como isso é implementado no ambiente de desenvolvimento é com diretórios “*layout*” com nomes com diferentes sufixos. Normalmente a pasta tem como nome só “*layout*” e contém ali os arquivos *XML* relevantes. Caso o desenvolvedor julgue necessário especificar ele mesmo um *layout* diferente para uma tela de tamanho grande, por exemplo, ele pode criar uma pasta chamada “*layout-large*” e colocar ali dentro o arquivo *XML* relevante. Outros sufixos suportados incluem “*small*”, “*xlarge*” e “*land*”, para telas pequenas e extragrandes e para a orientação paisagem (horizontal), como mostrado nas figuras 20, 21 e 22.

Figura 20: Exemplo de estrutura de diretórios mostrando uma pasta para layouts para orientação vertical (sem sufixo) e outra para orientação horizontal

```
MyProject/  
  res/  
    layout/  
      main.xml  
    layout-land/  
      main.xml
```

Figura 21: Exemplo de estrutura de diretórios mostrando pastas que combinam diferentes orientações e tamanhos de tela

```
MyProject/  
  res/  
    layout/           # default (portrait)  
      main.xml  
    layout-land/      # landscape  
      main.xml  
    layout-large/     # large (portrait)  
      main.xml  
    layout-large-land/ # large landscape  
      main.xml
```

Figura 22: Exemplo de estrutura de diretórios mostrando uma pasta recursos desenháveis (como imagens) para diferentes densidades de pixels

```
MyProject/  
  res/  
    drawable-xhdpi/  
      awesomeimage.png  
    drawable-hdpi/  
      awesomeimage.png  
    drawable-mdpi/  
      awesomeimage.png  
    drawable-ldpi/  
      awesomeimage.png
```

5 Conclusão

Neste trabalho foi desenvolvido um aplicativo para o sistema operacional *Android*, cuja principal funcionalidade é a criação e edição de diagramas de classe como propostos pela *UML*. Esse aplicativo, chamado de *AgileClass*, busca como diferencial ser ágil e simples, isto é, permitir que seus usuários consigam utilizá-lo com rapidez, sem necessidade de treinamento, e sem, contudo, deixar de trazer as principais funcionalidades esperadas de uma ferramenta desse tipo.

O *AgileClass* foi desenvolvido considerando uma interação com seus usuários via tela sensível ao toque. Ele também leva em conta a grande variedade de especificações de *hardware* que suportam a plataforma *Android*, nas quais o aplicativo será potencialmente executado. Isto é, o *AgileClass* busca ser eficiente em processamento e memória e busca ser escalável em sua interface, visto que o sistema *Android* pode ser executado em telas de celulares pequenos, até *tablets* e televisores.

Em comparação com as alternativas estudadas no capítulo 3, o *AgileClass* se destaca por seu suporte a gestos, que é mais expressivo que o *HandyUML* e o *Diagrid*, mas sem ir contra heurísticas de usabilidade, como acaba acontecendo com o *Draw Express*. Também pode-se citar seu foco exclusivo em desenvolvimento de diagramas de classe, que somente foi explorado pelo *HandyUML*, o que deixa o sistema mais focado e contendo funcionalidades necessárias e suficientes.

5.1 Trabalhos Futuros

A fim de aprimorar o *AgileClasse* e complementar sua experiência de uso, algumas funcionalidades extras podem ser adicionadas ao aplicativo. Dentre as possíveis habilidades a se acrescentar, pode-se citar a capacidade de redimensionar e alinhar classes e relacionamentos. Porém, a funcionalidade que mais pode contribuir é um sistema de reconhecimento de voz integrado ao aplicativo. O sistema *Android* possui nativamente uma forma de “teclado” que recebe entradas de voz, mas o uso dessa forma de entrada depende de selecionar como “teclado” o sistema de entrada de voz e então falar. Uma forma mais ágil de permitir o uso de voz seria com um botão próximo às entradas de texto que, quando pressionado, captura som e escreve na entrada textual.

Pode-se também citar a possibilidade de incluir a presença de configurações. Pode ser interessante permitir que o usuário configure seu ambiente de desenvolvimento e também o desenho das entidades (cores, espessura das linhas e outros). Outra funcionalidade não explorada é a de adição de comentários. Essa entidade é prevista em diagramas de classe, mas não foi implementada neste trabalho. Outras formas de prover mais informação sobre os componentes de um diagrama também podem ser úteis, como descrição para classes, atributos e outros. Por fim, diagramas de classes *UML* possuem diversas entidades e notações não exploradas por este trabalho. Por um lado, isto torna a ferramenta mais simples e leve, por outro, ela se torna menos completa e pode perder espaço entre usuários com mais necessidades.

Quanto ao *design* gráfico do aplicativo, como nenhum estudo sobre *design* foi realizado, não é possível tomar decisões embasadas quanto a esse quesito. Por exemplo, o sistema *Android* traz diversos temas para serem usados no aplicativo, mas por falta de estudo apropriado sobre o tema, escolheu-se o tema padrão proposto pelo *Android Studio*. Mais ainda, não foram feitos ícones para a aplicação, nem para os botões nem para o aplicativo em si. Trabalhos futuros podem averiguar os benefícios do uso de ícones e desenvolver ícones apropriados.

Finalmente, propõe-se a realização de estudos completos de usabilidade sobre o aplicativo. Ele foi desenvolvido tendo em vista heurísticas de usabilidade, mas o produto pronto não passou por testes de usabilidade. O estudo pode inclusive abordar o tema escalabilidade da interface, isto é, identificar as melhores opções de como ajustar a interface para tamanhos diferentes de tela. O resultado desses testes pode guiar futuros desenvolvimentos e sanar dúvidas quanto a melhores práticas a serem adotadas.

6 *Referências*

1. BASTIEN, J. M. Christian; SCAPIN, Dominique L.. **Ergonomic Criteria for the Evaluation of Human-Computer Interfaces**. França: Inria, 1993. 77 p. Disponível em: <http://www.cocoaheads.fr/wp-content/uploads/files/Ergonomic_Criteria.pdf>. Acesso em: 07 jul. 2017.
2. SOMMERVILLE, Ian. **Software Engineering**. 9. ed. Boston: Pearson, 2010. 773 p.
3. GOODWIN, David. **Object Oriented Analysis & Design**: Bedford: University Of Bedfordshire, 2012. 43 slides, color. Disponível em: <http://www2.warwick.ac.uk/fac/sci/physics/research/condensedmatt/imr_cdt/students/david_goodwin/teaching/modelling/l3_objectorientation.pdf>. Acesso em: 07 jul. 2017.
4. IIVARI, Juhani. Why are CASE tools not used? **Communications Of The Acm**, [s.l.], v. 39, n. 10, p.94-103, 1 out. 1996. Association for Computing Machinery (ACM). <http://dx.doi.org/10.1145/236156.236183>.
5. SHNEIDERMAN, Ben. **The Eight Golden Rules of Interface Design**. 2016. Disponível em: <<https://www.cs.umd.edu/users/ben/goldenrules.html>>. Acesso em: 07 jul. 2017.
6. **Google**. Disponível em: <<https://www.google.com/search/about/>>. Acesso em: 7 jul. 2017.
7. **The Java Language Environment**. Disponível em: <<http://www.oracle.com/technetwork/java/intro-141325.html>>. Acesso em: 7 jul. 2017.
8. **How do I add stickers to my photos? | Facebook Help Center | Facebook**. Disponível em: <<https://www.facebook.com/help/android->

app/1597631423793468?helpref=platform_switcher&ref=platform_switcher>.

Acesso em: 7 jul. 2017.

9. AMBLER, Scott. **Simple Tools for Software Modeling -OR- It's "Use the Simplest Tool" not "Use Simple Tools"** Simple Tools for Software Modeling - **OR- It's "Use the Simplest Tool" not "Use Simple Tools"**. 2009. Disponível em: <<http://www.agilemodeling.com/essays/simpleTools.htm>>. Acesso em: 07 jul. 2017.
10. AMBLER, Scott. **UML 2 Class Diagrams: An Agile Introduction**. 2009. Disponível em: <<http://www.agilemodeling.com/artifacts/classDiagram.htm>>. Acesso em: 07 jul. 2017.
11. **<uses-sdk>**. Disponível em: <<https://developer.android.com/guide/topics/manifest/uses-sdk-element.html>>. Acesso em: 7 jul. 2017.
12. **Activities**. Disponível em: <<https://developer.android.com/guide/components/activities.html>>. Acesso em: 7 jul. 2017.
13. **Adição da barra do aplicativo**. Disponível em: <<https://developer.android.com/training/appbar/index.html?hl=pt-br>>. Acesso em: 7 jul. 2017.
14. **Agile Methodology Understanding Agile Methodology**. Disponível em: <<http://agilemethodology.org/>>. Acesso em: 7 jul. 2017.
15. **The Android Source Code | Android Open Source Project**. Disponível em: <<https://source.android.com/source/>>. Acesso em: 7 jul. 2017.
16. **Android Studio IDE oficial do Android**. Disponível em: <<https://developer.android.com/studio/index.html?hl=pt-br>>. Acesso em: 7 jul. 2017.

17. **Animation and Graphics Overview.** Disponível em:
<<https://developer.android.com/guide/topics/graphics/overview.html>>. Acesso em: 7 jul. 2017.
18. **Arquitetura da plataforma.** Disponível em:
<<https://developer.android.com/guide/platform/index.html?hl=pt-br>>. Acesso em: 7 jul. 2017.
19. **Browser, OS, Search Engine including Mobile Usage Share.** Disponível em:
<http://gs.statcounter.com/#mobile_os-ww-monthly-201601-201601-bar>. Acesso em: 7 jul. 2017.
20. SHNEIDERMAN, Ben. **Shneiderman's "Eight Golden Rules of Interface Design"**. 2013. Disponível em:
<<http://www.designprinciplesftw.com/collections/shneidermans-eight-golden-rules-of-interface-design>>. Acesso em: 07 jul. 2017.
21. **Detecting Common Gestures.** Disponível em:
<<https://developer.android.com/training/gestures/detector.html>>. Acesso em: 7 jul. 2017.
22. **Device Compatibility.** Disponível em:
<<https://developer.android.com/guide/practices/compatibility.html>>. Acesso em: 7 jul. 2017.
23. **Diagrid - Android Apps on Google Play.** Disponível em:
<<https://play.google.com/store/apps/details?id=de.saax.diagrid>>. Acesso em: 7 jul. 2017.
24. **DrawExpress.** Disponível em: <<http://drawexpress.com/>>. Acesso em: 7 jul. 2017.

25. **DrawExpress Diagram Lite - Android Apps on Google Play.** Disponível em:
<<https://play.google.com/store/apps/details?id=com.drawexpress.lite>>. Acesso em:
7 jul. 2017.
26. **HandyUML FREE - Android Apps on Google Play.** Disponível em:
<<https://play.google.com/store/apps/details?id=com.handyuml.android.free>>.
Acesso em: 7 jul. 2017.
27. SIMS, Gary. **I want to develop Android Apps – What languages should I learn?**
2017. Disponível em: <<http://www.androidauthority.com/want-develop-android-apps-languages-learn-391008/>>. Acesso em: 01 jan. 2017.
28. **IDC: Smartphone OS Market Share.** Disponível em:
<<http://www.idc.com/promo/smartphone-market-share/os>>. Acesso em: 7 jul. 2017.
29. IBGE. **PNAD TIC: em 2014, pela primeira vez, celulares superaram microcomputadores no acesso domiciliar à Internet.** 2016. Disponível em:
<<http://saladeimprensa.ibge.gov.br/noticias?view=noticia&id=1&busca=1&idnoticia=3133>>. Acesso em: 07 jul. 2017.
30. **Input Technical Information | Android Open Source Project.** Disponível em:
<<https://source.android.com/devices/input/>>. Acesso em: 7 jul. 2017.
31. ROSENBERG, Jamie. **Introducing Google Play: All your entertainment, anywhere you go.** 2012. Disponível em:
<<https://googleblog.blogspot.com.br/2012/03/introducing-google-play-all-your.html>>. Acesso em: 07 jul. 2017.
32. **Manifesto for Agile Software Development.** Disponível em:
<<http://agilemanifesto.org/>>. Acesso em: 7 jul. 2017.

33. MARTIN, Mel. **Siri vs. Google Voice Search: may the best robot helper win.** 2012. Disponível em: <<https://www.engadget.com/2012/11/01/siri-vs-google-voice-search-may-the-best-robot-helper-win/>>. Acesso em: 07 jul. 2017.
34. **Menus.** Disponível em: <<https://developer.android.com/guide/topics/ui/menus.html?hl=pt-br#context-menu>>. Acesso em: 7 jul. 2017.
35. MORÁVEK, Adam. **HandyUML - Handy UML editor for Android tablets.** Disponível em: <<http://www.handyuml.com/#about>>. Acesso em: 07 jul. 2017.
36. NIELSEN, Jakob. **Usability 101: Introduction to Usability.** 2012. Disponível em: <<https://www.nngroup.com/articles/usability-101-introduction-to-usability/>>. Acesso em: 07 jul. 2017.
37. NIELSEN, Jakob. **10 Usability Heuristics for User Interface Design.** 1995. Disponível em: <<https://www.nngroup.com/articles/ten-usability-heuristics/>>. Acesso em: 07 jul. 2017.
38. **Paint.** Disponível em: <<https://developer.android.com/reference/android/graphics/Paint.html>>. Acesso em: 7 jul. 2017.
39. **Recursos da Support Library.** Disponível em: <<https://developer.android.com/topic/libraries/support-library/features.html?hl=pt-br>>. Acesso em: 7 jul. 2017.
40. VERGE STAFF. **Android: A visual history.** 2011. Disponível em: <<https://www.theverge.com/2011/12/7/2585779/android-history>>. Acesso em: 07 jul. 2017.

41. **UI Overview**. Disponível em:
<<https://developer.android.com/guide/topics/ui/overview.html>>. Acesso em: 7 jul. 2017.
42. BELL, Donald. **The class diagram**: An introduction to structure diagrams in UML 2. 2004. Disponível em:
<<http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/>>. Acesso em: 07 jul. 2017.
43. **iPlantUML – Apps para Android no Google Play**. Disponível em:
<https://play.google.com/store/apps/details?id=com.odelia.iplantuml&hl=pt_BR>. Acesso em: 7 jul. 2017.
44. AGILE METHODOLOGY (Ed.). **The Agile Movement**. 2008. Disponível em:
<<http://agilemethodology.org/>>. Acesso em: 07 jul. 2017.
45. AMBLER, Scott. **Agile Modeling (AM) Home Page**: Effective Practices for Modeling and Documentation. 2001. Disponível em:
<<http://www.agilemodeling.com/>>. Acesso em: 07 jul. 2017.
46. AGILE ALLIANCE (Org.). **Agile 101**. 2017. Disponível em:
<<https://www.agilealliance.org/agile101/>>. Acesso em: 07 jul. 2017.

7 Apêndice I: Artigo

Aplicativo para plataforma Android: "AgileClass", Editor de Diagramas de Classes para Dispositivos Sensíveis ao Toque

Rodolfo Pamplona Tenfen

rodolfo.tenfen@grad.ufsc.br

Abstract. *Today's software development methodology has been evolving to include agile development. However, the tools available today that give support to software development are either too simple, restrict and of poor usability, or have too many unnecessary features and are confusing. Most them are restricted to desktop operating systems, and exist under that paradigm of use, that is, mouse and keyboard. In this project, an Android application that breaks classical paradigms has been developed. A class diagram editor with appropriate usability for touch screen devices was created. The user interaction was developed with support for input by gestures as well as buttons and keyboard. The system has also been developed considering usability, compatibility and performance requirements, aiming to adequate the application to the needs of agile developers as well as support systems on multiple platforms.*

Resumo. *A metodologia de desenvolvimento de software atual tem evoluído de modo a abranger formas ágeis de desenvolvimento. Contudo, as ferramentas disponíveis hoje que dão suporte ao desenvolvimento de software ou são simples demais, restritas e de baixa usabilidade, ou tem recursos muito além do necessário e são confusas. A grande maioria delas se restringe aos sistemas operacionais de computadores de mesa, existindo sob este paradigma de utilização, ou seja, mouse e teclado. Neste trabalho foi desenvolvido um aplicativo para Android que quebre os paradigmas clássicos. Criou-se um editor de diagramas de classes com usabilidade apropriada a dispositivos sensíveis ao toque. Foi desenvolvida interação com o usuário com suporte a comandos por gestos, além de por botões e teclado. O sistema também foi desenvolvido tendo em vista requisitos de usabilidade, compatibilidade e performance, visando adequar o aplicativo às necessidades de desenvolvedores ágeis, bem como garantir o funcionamento do sistema em diferentes plataformas.*

1. Introdução

No desenvolvimento de projetos, faz-se uso de ferramentas e tecnologias que permitam e facilitem alcançar objetivos. A popularização do computador o firmou como ferramenta e hoje ele é utilizado em diversas áreas do conhecimento. As funcionalidades de um computador são providas por softwares, muitos dos quais são desenvolvidos especificamente para certas áreas, como ferramentas de desenho assistido por computador, ou CAD em inglês, no intuito de auxiliar o desenvolvimento e execução de projetos. Logo, a Ciência da Computação e a Engenharia de Software são áreas do conhecimento de grande importância para outras áreas, pois são responsáveis

pelo desenvolvimento de ferramentas modernas que permitem tornar realidade grandes projetos de engenharia.

Contudo, o próprio desenvolvimento de um software se trata de um projeto, o qual possui espaço para auxílio e automatização. Hoje, o processo de elaboração e criação de software é assistido por ferramentas como ambientes de desenvolvimento integrado, ou IDE em inglês, editores de diagramas, ferramentas de computer-aided software engineering, ou CASE, e sistemas de design gráfico de interface com usuário.

O uso de ferramentas CASE promete aumentar a qualidade do software gerado e a produtividade da geração de código, mas, de acordo com Livari (1996), o uso de ferramentas CASE é muito menor do que o esperado, tendo em vista os seus benefícios. Por outro lado, Ambler (2009) menciona potenciais custos de se usar essas ferramentas. Dentre eles, tem-se custos com treinamento, interface com usuário pobre e alta complexidade das ferramentas.

Esses custos tornam ferramentas CASE comuns inadequadas para o desenvolvimento ágil de software. Como o nome indica, a ideia desse paradigma de desenvolvimento é acelerar o projeto, encurtando o tempo entre entregas para o cliente, diminuindo, principalmente, o volume de documentação e enfatizando a geração de código. Tendo em vista esses princípios, Ambler recomenda o uso da ferramenta mais simples possível, em detrimento de ferramentas complexas que fazem muito mais do que o estritamente necessário. Ele, porém, propõe a utilização de objetos físicos como cartões, papel e quadros brancos. Essas ferramentas, porém, são demasiadamente simples, gerando modelos grosseiros e incompletos e sem possibilidade de serem automatizados ou de serem usados para a automatização de processos futuros.

Logo, métodos ágeis não são totalmente desprovidos de documentação e podem se favorecer do uso de ferramentas CASE mais sofisticadas. A proposta deste trabalho, então, é explorar as premissas da metodologia ágil, de modo a desenvolver um editor de diagrama de classes com alto nível de usabilidade e com um conjunto necessário e suficiente de funcionalidades para auxiliar o desenvolvimento de software de forma ágil, reduzindo os custos do uso da própria ferramenta.

Antes da popularização de tablets e smartphones, sistemas computacionais pessoais sofisticados capazes de suportar ferramentas de desenvolvimento resumiam-se a computadores de mesa e laptops, ambos comumente controlados por mouse e teclado. O paradigma de interação com computadores, contudo, mudou e se diversificou, aparelhos móveis e a utilização de telas sensíveis ao toque substituindo totalmente ou trabalhando em conjunto com mouse e teclado se tornaram populares e amplamente aceitos, de acordo com o IBGE (2016), os acessos à Internet feitos por celulares apenas já ultrapassam os feitos por microcomputadores no Brasil, o que mostra que estes aparelhos têm se tornado mais e mais populares. Esse novo paradigma de interação com computadores e a sua popularização trazem novas possibilidades de desenvolvimento de ferramentas, com formas de interação diferentes e requisitos de usabilidade diferentes.

2. Desenvolvimento

Na loja de aplicativos do Android, não é possível encontrar muitos exemplos de aplicativos para criação e edição de diagramas de classe, sendo que nenhum deles se

seja, o modelo vê um diagrama de classes como um grafo, como relações entre classes, mas sem se preocupar em como elas serão visualizadas.

3. Interação com o Usuário

O *AgileClass* é composto por cinco telas principais: a inicial, a de edição de classes, a de edição de atributos em classes, a de edição de operações em classes e a de edição de relacionamentos.

A primeira visão que o usuário tem do sistema é a tela inicial. É nela que ele realiza as principais e mais básicas funções do aplicativo, como mostra a figura 2. Essa tela é composta por três áreas estáticas, como mostra a figura 33: a barra do aplicativo (marcada como 1), a lista de botões (marcada como 2) e a área de edição (marcada como 3).

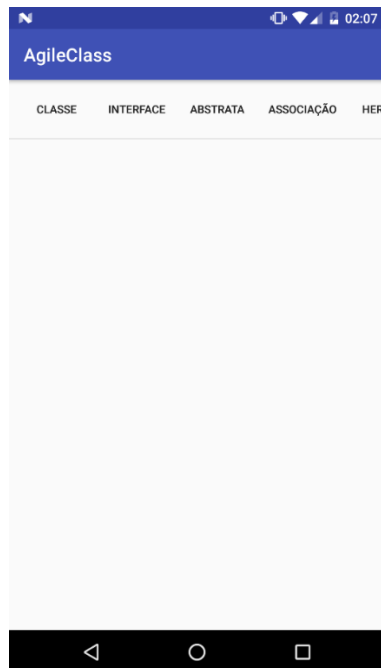


Figura 2. Tela inicial do AgileClass

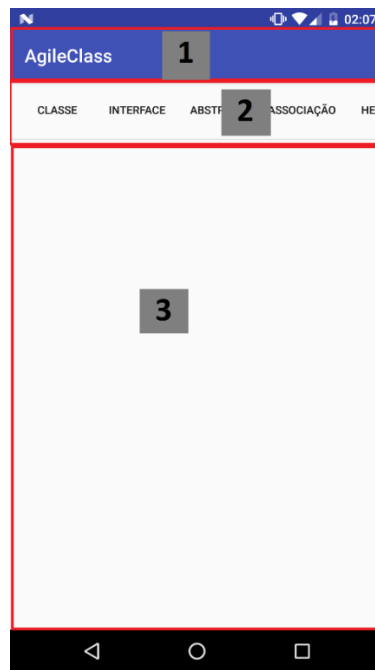


Figura 3. Áreas principais da tela inicial

A barra do aplicativo é uma área comumente encontrada em aplicativos para Android, é criada automaticamente pelo ambiente de desenvolvimento. Nela encontra-se o nome do aplicativo e pode conter um menu principal, ou menu do aplicativo, onde constam funcionalidades que afetam o sistema por inteiro.

A lista de botões é uma lista horizontal (no modo retrato) ou vertical (no modo paisagem) que contém os botões que alteram o modo de atuação da área de edição. Isto é, cada botão representa uma entidade e tocar em um certo botão e depois na área de edição criará a entidade selecionada. Por exemplo, se o botão “classe” for selecionado, um toque na área de edição criará uma classe.

A escolha por colocar esses botões na lista e não na barra do aplicativo, como é comum em outros aplicativos, se deve pela busca por um sistema escalável quanto ao tamanho da tela. Colocando os botões em uma lista, é possível acrescentar suporte a scroll na lista, isto é, caso não seja possível acomodar todos os botões na tela, eles podem ser acessados deslizando a lista para a posição desejada. A barra de aplicativo dispõe de pouco espaço e não seria possível colocar todos os botões para todas as entidades suportadas em uma tela de tamanho normal ou menor. Uma alternativa seria agrupar os botões em um menu acessível a partir de outros botões. Isso, porém cria a necessidade de o usuário realizar mais um toque.

A edição de informações das classes e relacionamentos se dá pelas telas de edição. Como o nome propõe, elas permitem editar componentes do diagrama, como mostrado na figura 4. Dentre essas telas, a principal é a de edição de classes. Seus componentes principais são o campo para edição do nome da classe, botões de rádio para modificar o tipo de classe (concreta, interface, abstrata ou enumeração) e duas listas, uma com os atributos e outra com as operações dela. Nas duas listas, seus elementos são botões que, quando clicados, exibem as telas de edição de atributo ou operação.

Para editar atributos e operações, as telas projetadas são similares. Elas contêm campos onde o usuário pode informar o nome do campo, sua visibilidade e informações específicas, como valor inicial para atributos ou parâmetros para operações.

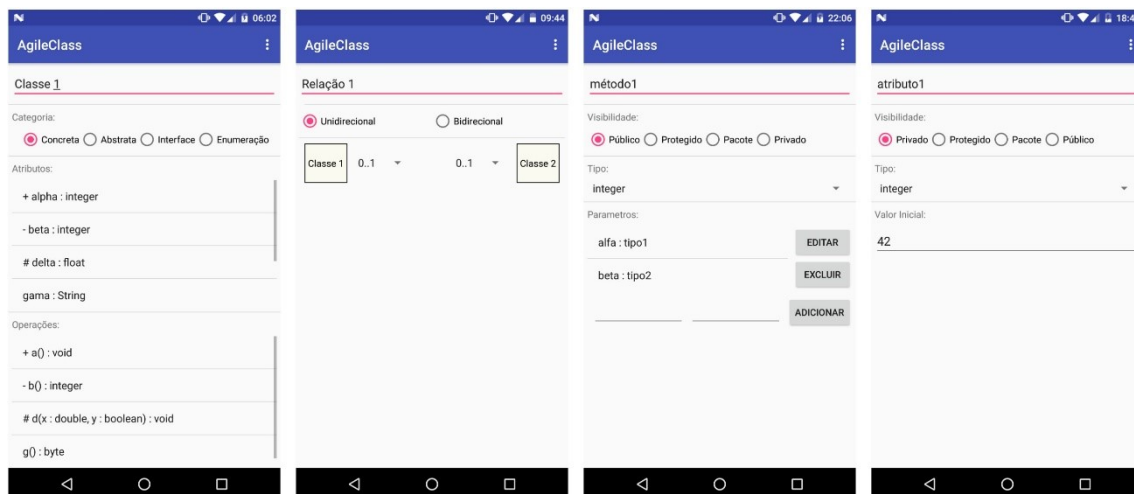


Figure 4. Exemplos de telas de edição de classe, relacionamento, operação e atributo, respectivamente.

Por fim, a tela de edição de relacionamentos permite editar as linhas e setas que ligam classes. Os itens mostrados, isto é, as características do relacionamento que podem ser alterados dependem do tipo de relacionamento. Por exemplo, uma relação de herança não possui multiplicidade, logo, sua tela de edição não possui esse campo.

Pode-se dizer que as telas de edição representam conjuntos de ações, como proposto por Shneiderman (2016) quando ele afirma que o desenvolvedor deve projetar diálogos para que tenham final. Isto é, cada tela de edição representa uma sequência agrupada de ações que o usuário deve tomar para deixar cada classe e relacionamento como ele achar melhor. Com o uso delas, também se melhora a visibilidade do estado do sistema, visto que a tela do dispositivo é totalmente tomada pela tela de edição vigente, em concordância com as recomendações tanto de Shneiderman, quanto de Nielsen (1995) e Bastien e Scapin (1993).

As entradas do usuário se dão principalmente através de toques na tela sensível ao toque. O sistema é capaz de reconhecer toques simples, que remetem a cliques com um mouse, a gestos com os dedos. A API Android considera alguns gestos como sendo os mais básicos e oferece algum suporte para implementação padrão deles. Esses gestos incluem toque, toque duplo, scroll e fling (arremesso). Quando a aplicação registra um objeto como sendo Observador de Gestos, ele passa a receber eventos, isto é, chamadas de métodos, quando algum desses gestos é detectado pelo sistema operacional. Interceptar esses eventos permite então que o aplicativo seja programado a reagir de acordo. A API também permite registrar um View como sendo um Observador de Toque Duplo, isso permite a esse View realizar ações diferentes caso ele receba um toque simples ou um toque duplo. A identificação de o que configura um toque simples e o que configura um toque duplo ficam a cargo do sistema operacional, mas pode ser sobrescrito pelo desenvolvedor.

No AgileClass foram implementados gestos comumente usados em outras aplicações e com significados bem conhecidos. Por exemplo, se o usuário realiza um gesto de scroll é possível afirmar que ele deseja que o conteúdo na tela deslize em uma determinada direção. Com isso em mente, quando um gesto desse tipo é realizado na área de edição de diagramas, o diagrama é deslocado na direção desejada. Quando o usuário toca em algum elemento, pode-se afirmar que ele deseja selecionar aquele elemento e interagir com ele especificamente. Dessa forma, ao tocar em uma classe ou relacionamento, essa entidade é selecionada. Quando uma entidade é selecionada, ela pode ser movida.

O mais importante gesto, porém, é o de criação de entidades. O usuário precisa tocar em um dos botões da lista de botões e então tocar na área de edição e isso criará uma entidade na tela. A entidade criada dependerá do botão selecionado. Essa forma de interação segue exemplo de outros aplicativos, como Facebook. Nesse aplicativo, é possível editar imagens adicionando figurinhas a elas. Para tanto, o usuário primeiro seleciona uma figurinha com um toque sobre a desejada e depois toca na tela onde ele deseja que a figurinha fique. Logo, esta forma de uso do AgileClass se adequa a metáforas já exploradas em outros aplicativos no mercado e aumenta a probabilidade de ela ser bem aceita e rapidamente aprendida.

Como é comum no ambiente Android, um toque longo na tela exhibe um menu de contexto sobre o elemento que recebeu o toque longo. No AgileClass, se uma classe receber um toque longo, um menu surge que possibilita editá-la ou removê-la. A decisão por seguir esse modelo foi tomada por se buscar manter o look and feel do sistema Android. Essa decisão é compatível com Nielsen quando ele fala sobre consistência e padrões, visto que está se buscando seguir uma convenção da plataforma usada. Assim, pode-se afirmar que um usuário acostumado com o sistema operacional Android aprenderá mais rapidamente a usar um aplicativo se o estilo de uso deste se mantiver consistente com o estilo de uso daquele.

Para este projeto, porém, também foram implementados alguns gestos pouco usuais, que realizem funções da interface de forma mais ágil, funcionando como um atalho. Por exemplo, um toque duplo na área de edição permite centralizar a tela no seu ponto central original. Esse gesto alinha-se à heurística de Nielsen sobre flexibilidade e eficiência de uso, pois se trata de um acelerador, isto é, um atalho.

4. Compatibilidade

Por fazer uso da linguagem Java, uma linguagem que busca ser inerentemente independente de máquina e portátil, desenvolvedores para o sistema Android não precisam se preocupar com a forma como seu código será executado. Os únicos componentes que precisam de cuidados específicos quanto à “máquina” que executa o aplicativo são os componentes visuais. Esses podem mudar a cada nova versão do ambiente de desenvolvimento Android.

A fim de tirar proveito da portabilidade da linguagem Java e de mitigar os efeitos das mudanças na API do Android, o desenvolvimento deste projeto deu prioridade, especialmente no modelo do projeto, a componentes padrões da linguagem Java. Esses componentes estão nativamente disponíveis no ambiente de

desenvolvimento e é possível afirmar que serão constantes através do tempo, isto é, não sofrerão mudanças que comprometam sua assinatura, lógica e/ou definição no ciclo de vida próximo da linguagem Java.

Por outro lado, o sistema Android está em constante mudança, tanto sua API quanto interface gráfica. A fim de desenvolver código que seja executado de forma consistente através das múltiplas versões do sistema, é recomendado o uso das bibliotecas de suporte que a API oferece. Contudo, mesmo usando componentes presentes na biblioteca de suporte, algumas funcionalidades precisam ser adaptadas pelo desenvolvedor para diferentes versões da API. Por exemplo, como mostrado no tutorial oficial do ambiente de desenvolvimento para implementar o gesto de “arremesso” (“fling” em inglês). Como mostra a figura 5, a recomendação é escrever código específico para cada versão e verificar em tempo de execução qual a versão do sistema sendo executada e então executar o código apropriado.

```
if (Build.VERSION.SDK_INT >= 11) {  
    scrollAnimator.setDuration(scroller.getDuration());  
    scrollAnimator.start();  
}
```

Figure 5. Código que verifica a versão do sistema Android sendo executada em tem de execução.

5. Interface Escalável

O sistema *Android* é utilizado em uma grande gama de dispositivos e suporta diversos tipos de *hardware*. A fim de oferecer uma interação com o usuário constante, o desenvolvedor deve considerar diferentes tamanhos de tela, dentre outras possíveis especificidades de *hardware*. A fim de sumarizar a miríade de possíveis variações de tamanho, o ambiente de desenvolvimento separa as telas em quatro grupos: pequena, normal, grande e extragrande. Essas classes representam intervalos de polegadas do tamanho da tela, como mostra a figura 6.

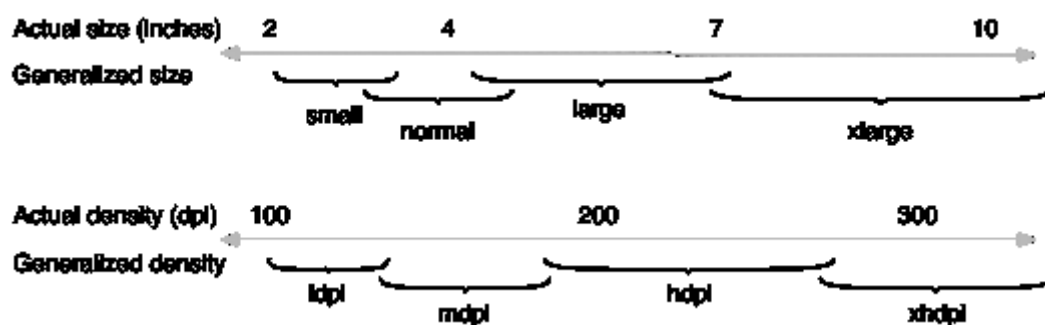


Figure 6. Ilustração de como o Android mapeia tamanhos e densidades reais em tamanhos e densidades generalizadas.

Os componentes estáticos de um layout podem ser descritos em um arquivo XML. Esse arquivo é guardado em uma subpasta do diretório “layout” do projeto. O ambiente de desenvolvimento Android oferece nativamente uma forma de utilizar esses arquivos XML adaptando-os para diferentes tamanhos de telas e diferentes densidades de pixels. Não só para telas diferentes, mas também orientações diferentes. Dessa forma o desenvolvedor pode desenhar sua aplicação tendo em vista diferentes formas como ela

vai aparecer na tela e o sistema operacional se encarrega de realizar a transição de uma para outra.

A forma como isso é implementado no ambiente de desenvolvimento é com diretórios “layout” com nomes com diferentes sufixos. Normalmente a pasta tem como nome só “layout” e contém ali os arquivos XML relevantes. Caso o desenvolvedor julgue necessário especificar ele mesmo um layout diferente para uma tela de tamanho grande, por exemplo, ele pode criar uma pasta chamada “layout-large” e colocar ali dentro o arquivo XML relevante. Outros sufixos suportados incluem “small”, “xlarge” e “land”, para telas pequenas e extragrandes e para a orientação paisagem (horizontal).

6. Conclusão

Neste trabalho foi desenvolvido um aplicativo para o sistema operacional Android, cuja principal funcionalidade é a criação e edição de diagramas de classe como propostos pela UML. Esse aplicativo, chamado de AgileClass, busca como diferencial ser ágil e simples, isto é, permitir que seus usuários consigam utilizá-lo com rapidez, sem necessidade de treinamento, e sem, contudo, deixar de trazer as principais funcionalidades esperadas de uma ferramenta desse tipo.

O AgileClass foi desenvolvido considerando uma interação com seus usuários via tela sensível ao toque. Ele também leva em conta a grande variedade de especificações de hardware que suportam a plataforma Android, nas quais o aplicativo será potencialmente executado. Isto é, o AgileClass busca ser eficiente em processamento e memória e busca ser escalável em sua interface, visto que o sistema Android pode ser executado em telas de celulares pequenos, até tablets e televisores.

Em comparação com as alternativas presentes no mercado, o AgileClass se destaca por seu suporte a gestos, que é mais expressivo que a maioria dos demais aplicativos, mas sem ir contra heurísticas de usabilidade. Também pode-se citar seu foco exclusivo em desenvolvimento de diagramas de classe, pouco explorado entre as alternativas, o que deixa o sistema mais focado e contendo funcionalidades necessárias e suficientes.

Referências

- Ambler, S. (2009) “Simple Tools for Software Modeling -OR- It's "Use the Simplest Tool" not "Use Simple Tools"Simple Tools for Software Modeling -OR- It's "Use the Simplest Tool" not "Use Simple Tools"”,
<http://www.agilemodeling.com/essays/simpleTools.htm>.
- Bastien, J. M. Christian; Scapin, Dominique L. (1993) “Ergonomic Criteria for the Evaluation of Human-Computer Interfaces”, http://www.cocoaheads.fr/wp-content/uploads/files/Ergonomic_Criteria.pdf.
- IBGE (2016) “PNAD TIC: em 2014, pela primeira vez, celulares superaram microcomputadores no acesso domiciliar à Internet. 2016”,
<http://saladeimprensa.ibge.gov.br/noticias?view=noticia&id=1&busca=1&idnoticia=3133>.

Iivari, J. (1996) "Why are CASE tools not used?", In: Communications Of The Acm, p. 94-103, <http://dx.doi.org/10.1145/236156.236183>.

Nielsen, J (1995) "10 Usability Heuristics for User Interface Design", <https://www.nngroup.com/articles/ten-usability-heuristics/>.

Shneiderman, B. (2013) "Shneiderman's "Eight Golden Rules of Interface Design"", <http://www.designprinciplesftw.com/collections/shneidermans-eight-golden-rules-of-interface-design>.

8 *Apêndice 2: Código*

8.1 *Diretório:*

app/src/main/java/com/pamplona/rodolfo/agileclass/modelo

8.1.1 **Agregacao.java**

```
package com.pamplona.rodolfo.agileclass.modelo;

public class Agregacao extends Associacao {
    public Agregacao(Diagrama d, long id, String nome, Classe
origem, Multiplicidade m1, Multiplicidade m2, Classe destino) {
        super(d, id, nome, origem, m1, m2, destino);
    }
}
```

8.1.2 **Associacao.java**

```
package com.pamplona.rodolfo.agileclass.modelo;

public class Associacao extends Relacionamento {

    private final String nome;

    private final Multiplicidade m1;
    private final Multiplicidade m2;

    public Associacao(Diagrama d, long id, String nome, Classe
origem, Multiplicidade m1, Multiplicidade m2, Classe destino,
Navegacao direcao) {
        super(d, id, origem, destino, direcao);

        this.nome = nome;

        this.m1 = m1;
        this.m2 = m2;
    }

    public Associacao(Diagrama d, long id, String nome, Classe
origem, Multiplicidade m1, Multiplicidade m2, Classe destino) {
        this(d, id, nome, origem, m1, m2, destino,
Navegacao.UNIDIRECIONAL);
    }

    public Associacao(Associacao a, String nome, Multiplicidade m1,
Multiplicidade m2) {
        this(a.diagrama(), a.id(), nome, a.origem(), m1, m2,
a.destino());
    }

    public String nome() {
        return nome;
    }

    public Multiplicidade multiplicidade1() {
        return m1;
    }
}
```

```

        public Multiplicidade multiplicidade2() {
            return m2;
        }
    }
}

```

8.1.3 Atributo.java

```

package com.pamplona.rodolfo.agileclass.modelo;

public class Atributo extends Membro {
    private final Tipo t;
    private final String valorInicial;

    public Atributo(Classe c, int posicao, Visibilidade v, Tipo t,
String nome, String valorInicial) {
        super(c, posicao, v, nome);

        this.t = t;
        this.valorInicial = valorInicial;
    }

    public Atributo(Classe c, int posicao, Visibilidade v, Tipo t,
String nome) {
        this(c, posicao, v, t, nome, null);
    }

    public Atributo(Atributo a, Visibilidade v, Tipo t, String
nome, String valorInicial) {
        this(a.classe(), a.posicao(), v, t, nome, valorInicial);
    }

    @Override
    public String toString() {
        return super.toString() + (valorInicial == null ? " " : " =
" + valorInicial + " ") + " : " + t.nome();
    }
}

```

8.1.4 Categoria.java

```

package com.pamplona.rodolfo.agileclass.modelo;

public enum Categoria {
    CONCRETA, ABSTRATA, INTERFACE
}

```

8.1.5 Classe.java

```

package com.pamplona.rodolfo.agileclass.modelo;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

```

```

public class Classe extends Tipo {
    private final Categoria c;

    private final List<Relacionamento> rs;
    private final List<Atributo> as;
    private final List<Operacao> os;

    public Classe(Diagrama d, long id, String nome, Categoria c) {
        super(d, id, nome);

        this.c = c;

        this.rs = new LinkedList<>();
        this.as = new LinkedList<>();
        this.os = new LinkedList<>();
    }

    public Classe(Classe t, String nome, Categoria c) {
        this(t.diagrama(), t.id(), nome, c);
    }

    private Atributo atributo(int posicao) {
        Atributo a = as.get(posicao);

        if (a == null)
            throw new IllegalArgumentException("Atributo " +
posicao + " inexistente");

        return a;
    }

    private Operacao operacao(int posicao) {
        Operacao o = os.get(posicao);

        if (o == null)
            throw new IllegalArgumentException("Operacao " +
posicao + " inexistente");

        return o;
    }

    protected int adicionarAtributo(Visibilidade v, Tipo t, String
nome, String valorInicial) {
        Atributo a = new Atributo(this, rs.size(), v, t, nome,
valorInicial);

        as.add(a);

        return as.size() - 1;
    }

    protected int adicionarAtributo(Visibilidade v, Tipo t, String
nome) {
        return adicionarAtributo(v, t, nome, null);
    }
}

```

```

        protected void editarAtributo(int posicao, Visibilidade v, Tipo
t, String nome, String valorInicial) {
            Atributo a = atributo(posicao);

            a = new Atributo(a, v, t, nome, valorInicial);

            as.set(posicao, a);
        }

        protected void removerAtributo(int posicao) {
            atributo(posicao);

            as.remove(posicao);
        }

        protected int adicionarOperacao(Visibilidade v, String nome,
Map<String, Tipo> parametros, Tipo retorno) {
            Operacao o = new Operacao(this, os.size(), v, nome,
parametros, retorno);

            os.add(o);

            return os.size() - 1;
        }

        protected void editarOperacao(int posicao, Visibilidade v,
String nome, Map<String, Tipo> parametros, Tipo retorno) {
            Operacao o = operacao(posicao);

            o = new Operacao(o, v, nome, parametros, retorno);

            os.set(posicao, o);
        }

        protected void removerOperacao(int posicao) {
            operacao(posicao);

            os.remove(posicao);
        }

        public List<List<String>> formaTextual() {
            List<List<String>> t = new ArrayList<>(3);

            t.add(0, new ArrayList<>(Arrays.asList(nome())));

            List<String> atributos = new ArrayList<>(as.size());
            for (Atributo a : as) {
                atributos.add(a.toString());
            }

            List<String> operacoes = new ArrayList<>(os.size());
            for (Operacao o : os) {
                operacoes.add(o.toString());
            }

            t.add(1, atributos);
            t.add(2, operacoes);
        }

```

```

        return t;
    }
}

```

8.1.6 Componente.java

```

package com.pamplona.rodolfo.agileclass.modelo;

public abstract class Componente {
    private final Diagrama d;
    private final long id;

    public Componente(Diagrama d, long id) {
        this.d = d;
        this.id = id;
    }

    protected Diagrama diagrama() {
        return d;
    }

    protected long id() {
        return id;
    }
}

```

8.1.7 Composicao.java

```

package com.pamplona.rodolfo.agileclass.modelo;

public class Composicao extends Associacao {
    public Composicao(Diagrama d, long id, String nome, Classe
origem, Multiplicidade m1, Multiplicidade m2, Classe destino) {
        super(d, id, nome, origem, m1, m2, destino);
    }
}

```

8.1.8 Diagrama.java

```

package com.pamplona.rodolfo.agileclass.modelo;

import android.util.Log;

import java.util.ArrayList;
import java.util.Collection;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

public class Diagrama {
    private static final String TAG = "#modelo.Diagrama";

    protected final Multiplicidade _0_1;
    protected final Multiplicidade _0_N;
    protected final Multiplicidade _1_1;
}

```

```

protected final Multiplicidade _1_N;
protected final Multiplicidade _N_N;

private final String nome;

private final Map<Long, Tipo> ts;
private long ultimoIndiceTipos = -1;
private final int nTiposPadrao;

private final Map<Long, Relacionamento> rs;
private long ultimoIndiceRelacionamento = -1;

private final Map<Long, Multiplicidade> ms;
private long ultimoIndiceMultiplicidade = -1;

public Diagrama(String nome) {
    this.nome = nome;

    ts = new HashMap<>();
    rs = new HashMap();
    ms = new HashMap<>();

    nTiposPadrao = criarTiposPadrao();

    // Multiplicidades Padrao
    _0_1 = new Multiplicidade(this, 0, "0", "1");
    _0_N = new Multiplicidade(this, 1, "0", "*");
    _1_1 = new Multiplicidade(this, 2, "1", "1");
    _1_N = new Multiplicidade(this, 3, "1", "*");
    _N_N = new Multiplicidade(this, 4, "*", "*");

    ms.put(0l, _0_1);
    ms.put(1l, _0_N);
    ms.put(2l, _1_1);
    ms.put(3l, _1_N);
    ms.put(4l, _N_N);
    ultimoIndiceMultiplicidade = 4;
}

private int criarTiposPadrao() {
    String[] padroes = {"byte", "short", "integer", "long",
"float", "double", "character", "string", "boolean", "void"};
    for (int i = 0; i < padroes.length; i++)
        ts.put(++ultimoIndiceTipos, new Tipo(this,
ultimoIndiceTipos, padroes[i]) {
        });

    return padroes.length;
}

private Tipo tipo(long id) {
    Tipo t = ts.get(id);

    Log.d(TAG, "tipo: tipos=" + ts);

    for (long x = 0l; x < ts.size(); x++)

```

```

        Log.d(TAG, "tipo: " + ts.get(x));

        if (t == null)
            throw new IllegalArgumentException("Tipo " + id + "
inexistente");

        return t;
    }

    private Classe classe(long id) {
        Tipo t = tipo(id);

        if (!(t instanceof Classe))
            throw new IllegalArgumentException("Classe " + id + "
inexistente");

        return (Classe) t;
    }

    private Multiplicidade multiplicidade(long id) {
        Multiplicidade m = ms.get(id);

        if (m == null)
            throw new IllegalArgumentException("Multiplicidade " +
id + " inexistente");

        return m;
    }

    private Relacionamento relacionamento(long id) {
        Relacionamento r = rs.get(id);

        if (r == null)
            throw new IllegalArgumentException("Relacionamento " +
id + " inexistente");

        return r;
    }

    public List<Tipo> tiposPadrao() {
        List<Tipo> tp = new LinkedList<>();

        for (long i = 0; i < nTiposPadrao; i++)
            tp.add(ts.get(i));

        return tp;
    }

    public int numeroDeClasses() {
        return ts.size() - nTiposPadrao;
    }

    //-----
    // CRUD de tudo
    //-----

```

```

// adicionarClasse
public long adicionarClasse(String nome, Categoria c) {
    long i = ++ultimoIndiceTipos;

    Classe t = new Classe(this, i, nome, c);

    ts.put(i, t);

    return i;
}

// pegarClasse
public List<List<String>> pegarTextoDaClasse(long id) {
    Classe c = classe(id);

    return c.formaTextual();
}

// editarClasse
public void editarClasse(long id, String nome, Categoria c) {
    Classe t = classe(id);

    t = new Classe(t, nome, c);

    ts.remove(id);

    ts.put(id, t);
}

// removerClasse
public void removerClasse(long id) {
    classe(id);

    ts.remove(id);
}

// adicionarAtributo
public int adicionarAtributo(long classe, Visibilidade v, long
tipo, String nome, String valorInicial) {
    Classe c = classe(classe);
    Tipo t = tipo(tipo);

    return c.adicionarAtributo(v, t, nome, valorInicial);
}

///// pegarAtributo

// editarAtributo
public void editarAtributo(long classe, int posicao,
Visibilidade v, long tipo, String nome, String valorInicial) {
    Classe c = classe(classe);
    Tipo t = tipo(tipo);

    c.editarAtributo(posicao, v, t, nome, valorInicial);
}

// removerAtributo
public void removerAtributo(long classe, int posicao) {

```

```

        Classe c = classe(classe);

        c.removerAtributo(posicao);
    }

    // adicionarOperacao
    public int adicionarOperacao(long classe, Visibilidade v,
String nome, Map<String, Long> parametros, long retorno) {
        Classe c = classe(classe);
        Tipo r = tipo(retorno);

        Map<String, Tipo> ps = new HashMap<>();
        for (String s : parametros.keySet()) {
            ps.put(s, tipo(parametros.get(s)));
        }

        return c.adicionarOperacao(v, nome, ps, r);
    }

    //// pegarOperacao

    // editarOperacao
    public void editarOperacao(long classe, int posicao,
Visibilidade v, String nome, Map<String, Long> parametros, long
retorno) {
        Classe c = classe(classe);
        Tipo r = tipo(retorno);

        Map<String, Tipo> ps = new HashMap<>();
        for (String s : parametros.keySet()) {
            ps.put(s, tipo(parametros.get(s)));
        }

        c.editarOperacao(posicao, v, nome, ps, r);
    }

    // removerOperacao
    public void removerOperacao(long classe, int posicao) {
        Classe c = classe(classe);

        c.removerOperacao(posicao);
    }

    // adicionarHeranca
    public long adicionarHeranca(long classeOrigem, long
classeDestino) {
        long i = ++ultimoIndiceRelacionamento;

        Classe o = classe(classeOrigem);
        Classe d = classe(classeDestino);

        Relacionamento r = new Heranca(this, i, o, d);

        rs.put(i, r);

        return i;
    }

```

```

        ///// pegarHeranca
        ///// editarHeranca
        ///// removerHeranca

        // adicionarImplementacao
        public long adicionarImplementacao(long classeOrigem, long
classeDestino) {
            long i = ++ultimoIndiceRelacionamento;

            Classe o = classe(classeOrigem);
            Classe d = classe(classeDestino);

            Relacionamento r = new Implementacao(this, i, o, d);

            rs.put(i, r);

            return i;
        }

        ///// pegarImplementacao
        ///// editarImplementacao
        ///// removerImplementacao

        // adicionarAssociacao
        public long adicionarAssociacao(String nome, long classeOrigem,
long multiplicidade1, long multiplicidade2, long classeDestino,
Navegacao direcao) {
            long i = ++ultimoIndiceRelacionamento;

            Classe o = classe(classeOrigem);
            Classe d = classe(classeDestino);

            Multiplicidade m1 = multiplicidade(multiplicidade1);
            Multiplicidade m2 = multiplicidade(multiplicidade2);

            Relacionamento r;
            if (direcao == null)
                r = new Associacao(this, i, nome, o, m1, m2, d);
            else
                r = new Associacao(this, i, nome, o, m1, m2, d,
direcao);

            rs.put(i, r);

            return i;
        }

        ///// pegarAssociacao

        // editarAssociacao
        public void editarAssociacao(long id, String nome, long
multiplicidade1, long multiplicidade2) {
            Relacionamento r = relacionamento(id);

            if (!(r instanceof Associacao))
                throw new IllegalArgumentException("Associacao " + id +
" inexistente");
        }

```

```

        Multiplicidade m1 = multiplicidade(multiplicidade1);
        Multiplicidade m2 = multiplicidade(multiplicidade2);

        Associacao a = (Associacao) r;

        rs.remove(id);

        a = new Associacao(a, nome, m1, m2);

        rs.put(id, a);
    }

    ///// removerAssociacao

    // adicionarAgregacao
    public long adicionarAgregacao(String nome, long classeOrigem,
    long multiplicidade1, long multiplicidade2, long classeDestino) {
        long i = ++ultimoIndiceRelacionamento;

        Classe o = classe(classeOrigem);
        Classe d = classe(classeDestino);

        Multiplicidade m1 = multiplicidade(multiplicidade1);
        Multiplicidade m2 = multiplicidade(multiplicidade2);

        Relacionamento r = new Agregacao(this, i, nome, o, m1, m2,
d);

        rs.put(i, r);

        return i;
    }

    ///// pegarAgregacao
    ///// editarAgregacao
    ///// removerAgregacao

    // adicionarComposicao
    public long adicionarComposicao(String nome, long classeOrigem,
    long multiplicidade1, long multiplicidade2, long classeDestino) {
        long i = ++ultimoIndiceRelacionamento;

        Classe o = classe(classeOrigem);
        Classe d = classe(classeDestino);

        Multiplicidade m1 = multiplicidade(multiplicidade1);
        Multiplicidade m2 = multiplicidade(multiplicidade2);

        Relacionamento r = new Composicao(this, i, nome, o, m1, m2,
d);

        rs.put(i, r);

        return i;
    }

    ///// pegarComposicao
    ///// editarComposicao

```

```

///// removerComposicao

public long[] classesDoRelacionamento(long id) {
    Relacionamento r = relacionamento(id);

    long[] cs = {r.origem().id(), r.destino().id()};

    return cs;
}

public String[] multiplicidadesDaAssociacao(long id) {
    Relacionamento r = relacionamento(id);

    if (!(r instanceof Associacao))
        throw new IllegalArgumentException("Relacionamento " +
id + " não é uma Associação");

    Associacao a = (Associacao) r;

    String[] ms = {a.multiplicidade1().toString(),
a.multiplicidade2().toString()};

    return ms;
}

public void removerRelacionamento(long id) {
    relacionamento(id);

    rs.remove(id);
}

// adicionarMultiplicidade
public long adicionarMultiplicidade(String de, String ate) {
    long i = ++ultimoIndiceMultiplicidade;

    Multiplicidade m = new Multiplicidade(this, i, de, ate);

    ms.put(i, m);

    return i;
}

// pegarTodasMultiplicidade
public List<Multiplicidade> multiplicidades() {
    Collection<Multiplicidade> values = ms.values();
    List<Multiplicidade> mults = new
ArrayList<>(values.size());

    for (Multiplicidade m : values)
        if (!mults.contains(m)) {
            mults.add(m);
        }

    return mults;
}
}

```

8.1.9 Generalizacao.java

```
package com.pamplona.rodolfo.agileclass.modelo;

public abstract class Generalizacao extends Relacionamento {

    public Generalizacao(Diagrama d, long id, Classe origem, Classe destino) {
        super(d, id, origem, destino);
    }
}
```

8.1.10 Heranca.java

```
package com.pamplona.rodolfo.agileclass.modelo;

public class Heranca extends Generalizacao {

    public Heranca(Diagrama d, long id, Classe origem, Classe destino) {
        super(d, id, origem, destino);
    }
}
```

8.1.11 Implementacao.java

```
package com.pamplona.rodolfo.agileclass.modelo;

public class Implementacao extends Generalizacao {

    public Implementacao(Diagrama d, long id, Classe origem, Classe destino) {
        super(d, id, origem, destino);
    }
}
```

8.1.12 Membro.java

```
package com.pamplona.rodolfo.agileclass.modelo;

public abstract class Membro {
    private final Classe c;
    private final int p;

    private final Visibilidade v;
    private final String nome;

    public Membro(Classe c, int posicao, Visibilidade v, String nome) {
        this.c = c;
        this.p = posicao;

        this.v = v;
        this.nome = nome;
    }
}
```

```

protected Classe classe() {
    return c;
}

protected int posicao() {
    return p;
}

@Override
public String toString() {
    return v.simbolo() + " " + nome;
}
}

```

8.1.13 Multiplicidade.java

```

package com.pamplona.rodolfo.agileclass.modelo;

public class Multiplicidade extends Componente {
    private final String de;
    private final String ate;

    public Multiplicidade(Diagrama d, long id, String de, String
ate) {
        super(d, id);

        if (de == null || ate == null)
            throw new NullPointerException("\"de\" e \"ate\" devem
ser não nulos: de=" + de + ", ate=" + ate);

        this.de = de;
        this.ate = ate;
    }

    public String de() {
        return de;
    }

    public String ate() {
        return ate;
    }

    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof Multiplicidade))
            return false;

        Multiplicidade m = (Multiplicidade) obj;

        return de.equals(m.de) && ate.equals(m.ate);
    }

    @Override
    public int hashCode() {
        return (de + ate).hashCode();
    }
}

```

```

@Override
public String toString() {
    if (de.equals(ate)) return de;
    else return de + ".." + ate;
}
}

```

8.1.14 Navegacao.java

```
package com.pamplona.rodolfo.agileclass.modelo;
```

```

public enum Navegacao {
    UNIDIRECIONAL, BIDIRECIONAL
}

```

8.1.15 Operacao.java

```
package com.pamplona.rodolfo.agileclass.modelo;
```

```

import java.util.HashMap;
import java.util.Map;

```

```

public class Operacao extends Membro {
    private final Map<String, Tipo> ps;
    private final Tipo r;

    public Operacao(Classe c, int posicao, Visibilidade v, String
nome, Map<String, Tipo> parametros, Tipo retorno) {
        super(c, posicao, v, nome);

        this.ps = new HashMap<>(parametros);
        this.r = retorno;
    }

    public Operacao(Operacao o, Visibilidade v, String nome,
Map<String, Tipo> parametros, Tipo retorno) {
        this(o.classe(), o.posicao(), v, nome, parametros,
retorno);
    }

    @Override
    public String toString() {
        String s = super.toString();

        if (ps != null && !ps.isEmpty()) {
            s += "(";

            for (String n : ps.keySet())
                s += n + " : " + ps.get(n);

            s += ")";
        }

        s += " : " + r.nome();
    }
}

```

```

        return s;
    }
}

```

8.1.16 Projeto.java

```

package com.pamplona.rodolfo.agileclass.modelo;

import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

public class Projeto {
    private static final List<String> nomesUsados = new
LinkedList<>(); // FIXME

    private final Map<String, Diagrama> ds;

    public Projeto(String nome) throws IllegalArgumentException {
        if (nomesUsados.contains(nome))
            throw new IllegalArgumentException("Nome de parâmetro
já usado");

        ds = new HashMap<>();
    }

    public Diagrama adicionarDiagrama(String nome) throws
IllegalArgumentException {
        if (ds.containsKey(nome))
            throw new IllegalArgumentException("Diagrama já
existe");

        Diagrama d = new Diagrama(nome);

        ds.put(nome, d);

        return d;
    }

    public Diagrama diagrama(String nome) {
        return ds.get(nome);
    }

    public int numeroDeDiagramas() {
        return ds.size();
    }
}

```

8.1.17 Relacionamento.java

```

package com.pamplona.rodolfo.agileclass.modelo;

public class Relacionamento extends Componente {
    private final Classe o;
    private final Classe d;
}

```

```

        private final Navegacao dir;

        public Relacionamento(Diagrama d, long id, Classe origem,
Classe destino) {
            this(d, id, origem, destino, Navegacao.UNIDIRECIONAL);
        }

        public Relacionamento(Diagrama d, long id, Classe origem,
Classe destino, Navegacao direcao) {
            super(d, id);

            this.o = origem;
            this.d = destino;

            this.dir = direcao;
        }

        public Classe origem() {
            return o;
        }

        public Classe destino() {
            return d;
        }

        public Navegacao navegacao() {
            return dir;
        }
    }

```

8.1.18 Tipo.java

```

package com.pamplona.rodolfo.agileclass.modelo;

public abstract class Tipo extends Componente {
    private final String nome;

    public Tipo(Diagrama d, long id, String nome) {
        super(d, id);

        this.nome = nome;
    }

    public String nome() {
        return nome;
    }

    @Override
    public String toString() {
        return nome();
    }
}

```

8.1.19 Visibilidade.java

```

package com.pamplona.rodolfo.agileclass.modelo;

```

```

public enum Visibilidade {
    PUBLICA('+'), PRIVADA('-'), PROTEGIDA('#'), PACOTE('~');

    private final char simbolo;

    Visibilidade(char simbolo) {
        this.simbolo = simbolo;
    }

    public char simbolo() {
        return simbolo;
    }
}

```

8.2 Diretório: *app/src/main/java/com/pamplona/rodolfo/agileclass/ui*

8.2.1 AgileClass.java

```

package com.pamplona.rodolfo.agileclass.ui;

import android.content.res.Configuration;
import android.content.res.Resources;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.FragmentTransaction;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.AppCompatButton;
import android.util.Log;
import android.view.ContextMenu;
import android.view.GestureDetector;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.View;
import android.view.ViewGroup;

import com.pamplona.rodolfo.agileclass.R;
import com.pamplona.rodolfo.agileclass.modelo.Diagrama;
import com.pamplona.rodolfo.agileclass.modelo.Projeto;

import java.util.Arrays;
import java.util.HashMap;
import java.util.LinkedList;
import java.util.List;
import java.util.Map;

public class AgileClass extends AppCompatActivity implements
    EdicaoClasse.OnFragmentInteractionListener {
    private static final String TAG = "#ui.AgileClass";

    private AreaEdicao areaEdicao;
    private GrupoBotoes botoes;

    private List<Projeto> projetos;
    private Diagrama diagramaAtual;

    private MenuItem centralizar;

```

```

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    Log.d(TAG, "onCreate() called with: savedInstanceState = ["
+ savedInstanceState + "]");

    setContentView(R.layout.activity_agile_class);

    if (true) {
        projetos = new LinkedList<>();
        projetos.add(new Projeto("Sem nome " + (projetos.size()
+ 1)));
        String nomeDiag = "Sem nome " +
(projetos.get(0).numeroDeDiagramas() + 1);
        projetos.get(0).adicionarDiagrama(nomeDiag);
        diagramaAtual = projetos.get(0).diagrama(nomeDiag);

        Resources res = getResources();

        areaEdicao = (AreaEdicao) findViewById(R.id.canvas);

        ViewGroup menuBotoes;

        if (getResources().getConfiguration().orientation ==
Configuration.ORIENTATION_LANDSCAPE) {
            menuBotoes = (ViewGroup)
findViewById(R.id.listButtonsV);
        } else {
            menuBotoes = (ViewGroup)
findViewById(R.id.ll_lista_botoes_horizontal);
        }

        Map<AppCompatButton, AreaEdicao.Modos> bs = new
HashMap<>();
        List<AreaEdicao.Modos> modos =
Arrays.asList(AreaEdicao.Modos.values());

        for (int i = 0; i < menuBotoes.getChildCount(); i++) {
            final AppCompatButton b = (AppCompatButton)
menuBotoes.getChildAt(i);

            Log.d(TAG, "onCreate: b=" + b);

            bs.put(b, modos.get(i));
        }

        botoes = new GrupoBotoes(this, bs);

        detector = new GestureDetector(this, areaEdicao);

        Log.d(TAG, "onCreate: areaEdicao=" + areaEdicao);

        centralizar = (MenuItem)
findViewById(R.id.mi_centralizar);

        Log.d(TAG, "onCreate() returned: " + "fim");
    }
}

```

```

        }
    }

    protected void registrar(Classe c) {
        Log.d(TAG, "registrar() called with: c = [" + c + "]");

        registerForContextMenu(c);
    }

    protected void desfazerSelecaoBotoes() {
        Log.d(TAG, "desfazerSelecaoBotoes() called");

        botoes.desselecionarTodos();
    }

    protected Diagrama diagramaAtual() {
        return diagramaAtual;
    }

    @Override
    public void onCreateContextMenu(ContextMenu menu, View v,
                                   ContextMenu.ContextMenuInfo
menuInfo) {
        Log.d(TAG, "onCreateContextMenu() called with: menu = [" +
menu + "], v = [" + v + "], menuInfo = [" + menuInfo + "]");

        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.classe_menu_contexto, menu);

        super.onCreateContextMenu(menu, v, menuInfo);
    }

    @Override
    public boolean onContextItemSelected(MenuItem item) {
        Log.d(TAG, "onContextItemSelected() called with: item = ["
+ item + "]");

        Classe info = (Classe) item.getMenuInfo();

        Log.d(TAG, "onContextItemSelected: info=" + info);

        switch (item.getItemId()) {
            case R.id.editarClasseItemMenu:
                editarClasse((Classe) info);

                return true;
            case R.id.removerClasseItemMenu:
                removerClasse((Classe) info);

                return true;
            default:
                return super.onContextItemSelected(item);
        }
    }

    private void editarClasse(Classe c) {
        Log.d(TAG, "editarClasse() called with: c = [" + c + "]");
    }

```

```

        setContentView(R.layout.fragment_edicao_classe);

        if (findViewById(R.id.edicaoClasse) != null) {
            Log.d(TAG, "editarClasse: oi");

            EdicaoClasse ec = new EdicaoClasse();

            ec.setArguments(getIntent().getExtras());

            FragmentTransaction transicao =
getSupportFragmentManager().beginTransaction();

            transicao.replace(R.id.edicaoClasse, ec);
            transicao.addToBackStack(null);
            transicao.commit();

getSupportFragmentManager().executePendingTransactions();
        }

        Log.d(TAG, "editarClasse() returned: " + "fim");
    }

    private void removerClasse(Classe c) {
        Log.d(TAG, "removerClasse() called with: c = [" + c +
    "]"");
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        getMenuInflater().inflate(R.menu.app_menu, menu);
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        Log.d(TAG, "onOptionsItemSelected() called with: item = ["
+ item + "]"");

        switch (item.getItemId()) {
            case R.id.mi_centralizar:
                areaEdicao.centralizar();

                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }

    @Override
    public void onBackPressed() {
        Log.d(TAG, "onBackPressed() called");

        if (getSupportFragmentManager().getBackStackEntryCount() >
0) {

```

```

        Log.d(TAG, "onBackPressed: oi");

        setContentView(R.layout.activity_agile_class);

        getSupportFragmentManager().popBackStackImmediate();

        return;
    } else {
        Log.d(TAG, "onBackPressed: tchau");

        finish();
    }
}

GestureDetector detector;

@Override
public boolean onTouchEvent(MotionEvent e) {
    Log.d(TAG, "onTouchEvent() called with: e = [" + e + "]");

    detector.onTouchEvent(e);

    return super.onTouchEvent(e);
}

protected void modo(AreaEdicao.Modo m) {
    Log.d(TAG, "modo() called with: m = [" + m + "]");

    areaEdicao.modo(m);
}

@Override
public void onFragmentInteraction(Uri uri) {
    Log.d(TAG, "onFragmentInteraction() called with: uri = [" +
uri + "]");
}
}

```

8.2.2 AreaEdicao.java

```

package com.pamplona.rodolfo.agileclass.ui;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.Rect;
import android.graphics.RectF;
import android.os.Parcel;
import android.os.Parcelable;
import android.util.AttributeSet;
import android.util.Log;
import android.view.GestureDetector;
import android.view.MotionEvent;
import android.view.ViewGroup;

import com.pamplona.rodolfo.agileclass.modelo.Categoria;

```

```

import com.pamplona.rodolfo.agileclass.modelo.Diagrama;
import com.pamplona.rodolfo.agileclass.modelo.Navegacao;
import com.pamplona.rodolfo.agileclass.ui.relacoes.Agregacao;
import com.pamplona.rodolfo.agileclass.ui.relacoes.Associacao;
import com.pamplona.rodolfo.agileclass.ui.relacoes.Composicao;
import com.pamplona.rodolfo.agileclass.ui.relacoes.Heranca;
import com.pamplona.rodolfo.agileclass.ui.relacoes.Implementacao;
import com.pamplona.rodolfo.agileclass.ui.relacoes.Relacionamento;

import java.util.ArrayList;
import java.util.LinkedList;

public class AreaEdicao extends ViewGroup implements
GestureDetector.OnDoubleTapListener,
GestureDetector.OnGestureListener {

    private static final String TAG = "#ui.AreaEdicao";

    private int largura = 0;
    private int altura = 0;

    private ArrayList<Classe> classes = new ArrayList<>();
    private ArrayList<Point> posicoes = new ArrayList<>();

    private ArrayList<Relacionamento> relacoes = new ArrayList<>();

    private LinkedList<Classe> selecionadas = new LinkedList<>();

    private AgileClass ac;

    public enum Modo {
        CLASSE, INTERFACE, ABSTRATA, ASSOCIACAO, HERANCA,
        REALIZACAO, AGREGACAO, COMPOSICAO
    }

    private Modo modo;

    public AreaEdicao(Context contexto) {
        this(contexto, null);
    }

    public AreaEdicao(Context contexto, AttributeSet atributos) {
        super(contexto, atributos);
        Log.d(TAG, "AreaEdicao() called with: ac = [" + contexto +
        "], atributos = [" + atributos + "]);

        this.ac = (AgileClass) contexto;

        android.util.Log.d(TAG, "AreaEdicao() returned: " + "fim");
    }

    protected void modo(Modo m) {
        Log.d(TAG, "modo() called with: m = [" + m + "]);

        modo = m;
    }
}

```

```

public Diagrama diagrama() {
    Log.d(TAG, "diagrama() called");

    return ac.diagramaAtual();
}

@Override
protected void onMeasure(int larguraMeasureSpec, int
alturaMeasureSpec) {
    Log.d(TAG, "onMeasure() called with: larguraMeasureSpec =
[" + larguraMeasureSpec + "], alturaMeasureSpec = [" +
alturaMeasureSpec + "]);

    int nFilhos = getChildCount();

    Log.d(TAG, "onMeasure: nFilhos=" + nFilhos);

    super.onMeasure(larguraMeasureSpec, alturaMeasureSpec);
    Log.d(TAG, "onMeasure() returned: " + "fim");
}

@Override
public void onSizeChanged(int l, int a, int lAnterior, int
aAnterior) {
    Log.d(TAG, "onSizeChanged() called with: l = [" + l + "], a
= [" + a + "], lAnterior = [" + lAnterior + "], aAnterior = [" +
aAnterior + "]);

    largura = l;
    altura = a;

    Point centro = new Point(window.centerX(),
window.centerY());

    Log.d(TAG, "onSizeChanged: centro=" + centro);

    int hNeg = largura / 2;
    int vNeg = altura / 2;
    int hPos = (largura % 2 == 0 ? largura / 2 : (largura + 1)
/ 2);
    int vPos = (altura % 2 == 0 ? altura / 2 : (altura + 1) /
2);

    window.set(centro.x - hNeg, centro.y - vNeg, centro.x +
hPos, centro.y + vPos);

    Log.d(TAG, "onSizeChanged() returned: window=" + window);
}

@Override
protected void onLayout(boolean changed, int l, int t, int r,
int b) {
    Log.d(TAG, "onLayout() called with: changed = [" + changed
+ "], l = [" + l + "], t = [" + t + "], r = [" + r + "], b = [" + b
+ "]);

    final int nFilhos = getChildCount();

```

```

Log.d(TAG, "onLayout: nFilhos=" + nFilhos);

int[] wc = new int[2];
getLocationInWindow(wc);

for (int i = 0; i < classes.size(); i++) {
    Point p = new Point(posicoes.get(i).x,
posicoes.get(i).y);
    Classe v = classes.get(i);

    v.measure(r - l, b - t);

    int vl = v.getMeasuredWidth();
    int va = v.getMeasuredHeight();

    Log.d(TAG, "onLayout: v.l=" + vl + ", v.a=" + va);

    int esqd = p.x - wc[0] - (vl % 2 == 0 ? vl / 2 : (vl +
1) / 2);
    int topo = p.y - wc[1] - (va % 2 == 0 ? va / 2 : (va +
1) / 2);
    int dirt = p.x - wc[0] + vl / 2;
    int base = p.y - wc[1] + va / 2;

    esqd -= window.left;
    dirt -= window.left;
    topo -= window.top;
    base -= window.top;
    v.layout(esqd, topo, dirt, base);

    Log.d(TAG, "onLayout: classe " + i + " na posicao " +
posicoes.get(i) + ", especificamente (" + esqd + ", " + topo + ", "
+ dirt + ", " + base + ")");
}

for (Relacionamento rela : relacoes) {
    rela.measure(r - l, b - t);

    Rect limites = rela.limites();

    Log.d(TAG, "onLayout: limites=" + limites);

    Log.d(TAG, "onLayout: wc[0]=" + wc[0] + ",wc[1]" +
wc[1]);

    int esqd = limites.left - wc[0];
    int topo = limites.top - wc[1];
    int dirt = limites.right - wc[0];
    int base = limites.bottom - wc[1];

    Log.d(TAG, "onLayout: window=" + window);

    esqd -= window.left;
    dirt -= window.left;
    topo -= window.top;
    base -= window.top;

    rela.invalidate();
}

```

```

        rela.layout(esqd, topo, dirt, base);

        Log.d(TAG, "onLayout: relacao " + rela + " em (" + esqd
+ ", " + topo + ", " + dirt + ", " + base + ")");
    }
}

private Rect window = new Rect(0, 0, 0, 0);

public void adicionarClasse(int x, int y) {
    Log.d(TAG, "adicionarClasse() called with: x = [" + x + "],
y = [" + y + "]");

    Classe.Tipo t = null;
    Categoria cat = null;
    switch (modo) {
        case CLASSE:
            t = Classe.Tipo.CONCRETA;
            cat = Categoria.CONCRETA;
            break;
        case ABSTRATA:
            t = Classe.Tipo.ABSTRATA;
            cat = Categoria.ABSTRATA;
            break;
        case INTERFACE:
            t = Classe.Tipo.INTERFACE;
            cat = Categoria.INTERFACE;
            break;
    }

    int i = posicoes.size();

    int n = ac.diagramaAtual().numeroDeClasses();
    String nomeClasse = (t == Classe.Tipo.INTERFACE ?
"Interface " : "Classe ") + (n + 1);
    long id = ac.diagramaAtual().adicionarClasse(nomeClasse,
cat);

    Classe c = new Classe(ac, this, i, id, t, nomeClasse, null,
null);

    addView(c, i);

    classes.add(c);
    posicoes.add(new Point(x, y));

    ac.registrar(c);

    ac.desfazerSelecaoBotoes();
    modo(null);
}

public void mover(int c, int difX, int difY) {
    Log.d(TAG, "mover() called with: this = [" + hashCode() +
"], c = [" + c + "], difX = [" + difX + "], difY = [" + difY +
"]");

    Point p = posicoes.get(c);

```

```

        Log.d(TAG, "mover: movendo filho " + classes.get(c) + " de
" + p + " para (" + p.x + difX + ", " + p.y + difY + ")");

        p.set(p.x + difX, p.y + difY);

        invalidate();
        requestLayout();
    }

    public void posicao(Class c, Rect r) {
        Log.d(TAG, "posicao() called with: c = [" + c + "], r = ["
+ r + "]");

        Point p = posicoes.get(c.indice());

        Log.d(TAG, "posicao: p=" + p);

        int vl = c.getMeasuredWidth();
        int va = c.getMeasuredHeight();

        Log.d(TAG, "posicao: c.l=" + vl + ", c.a=" + va);

        int esqd = p.x - (vl % 2 == 0 ? vl / 2 : (vl + 1) / 2);
        int topo = p.y - (va % 2 == 0 ? va / 2 : (va + 1) / 2);
        int dirt = p.x + vl / 2;
        int base = p.y + va / 2;

        r.set(esqd, topo, dirt, base);

        Log.d(TAG, "posicao() returned: r=(" + r.left + ", " +
r.top + ", " + r.right + ", " + r.bottom + ")");
    }

    protected void iniciarPanning(int x, int y) {
        Log.d(TAG, "iniciarPanning() called with: x = [" + x + "],
y = [" + y + "]");

        if (xAnterior != -1 || yAnterior != -1) {
            Log.d(TAG, "iniciarPanning: VISH");
        }

        xAnterior = x;
        yAnterior = y;
    }

    protected void redefinirPanning() {
        Log.d(TAG, "redefinirPanning() called");

        xAnterior = -1;
        yAnterior = -1;
    }

    public void selecionar(int i) {
        Log.d(TAG, "selecionar() called with: i = [" + i + "]");

        Classe c = classes.get(i);
    }

```

```

        if (selecionadas.size() >= 2)
            selecionadas.remove(0);

        selecionadas.add(c);

        if (selecionadas.size() == 2)
            if (modo == Modo.ASSOCIACAO || modo == Modo.HERANCA ||
modo == Modo.REALIZACAO || modo == Modo.AGREGACAO || modo ==
Modo.COMPOSICAO)
                adicionarRelacao();
    }

    public void desselecionar(int i) {
        Log.d(TAG, "desselecionar() called with: i = [" + i + "]");

        Classe c = classes.get(i);

        selecionadas.remove(c);
    }

    public void desselecionarTodas() {
        Log.d(TAG, "desselecionarTodas() called");

        for (Classe c : selecionadas)
            c.desselecionar();

        selecionadas.clear();
    }

    private int[] mults = {1, 2, 3};
    private int multi = 0;

    public void adicionarRelacao() {
        Log.d(TAG, "adicionarRelacao() called");

        Classe c1 = selecionadas.getFirst();
        Classe c2 = selecionadas.getLast();

        Relacionamento r;
        long id;
        switch (modo) {
            case ASSOCIACAO:
                id = diagrama().adicionarAssociacao("", c1.id(),
mults[multi++ % 3], mults[multi++ % 3], c2.id(),
Navegacao.UNIDIRECIONAL);

                r = new Associacao(ac, this, id, c1, c2);

                break;
            case HERANCA:
                id = diagrama().adicionarHeranca(c1.id(), c2.id());

                r = new Heranca(ac, this, id, c1, c2);

                break;
            case REALIZACAO:

```

```

        id = diagrama().adicionarImplementacao(c1.id(),
c2.id());

        r = new Implementacao(ac, this, id, c1, c2);

        break;
    case AGREGACAO:
        id = diagrama().adicionarAgregacao("", c1.id(),
mults[multi++ % 3], mults[multi++ % 3], c2.id());

        r = new Agregacao(ac, this, id, c1, c2);

        break;
    case COMPOSICAO:
        id = diagrama().adicionarComposicao("", c1.id(),
mults[multi++ % 3], mults[multi++ % 3], c2.id());

        r = new Composicao(ac, this, id, c1, c2);

        break;
    default:
        r = null;
}

addView(r);
relacoes.add(r);

c1.desselecionar();
c2.desselecionar();
desselecionar(c1.indice());
desselecionar(c2.indice());

ac.desfazerSelecaoBotoes();
modo(null);
}

//-----
// Toques
//-----

@Override
public boolean onSingleTapConfirmed(MotionEvent e) {
    Log.d(TAG, "onSingleTapConfirmed() called with: e = [" + e
+ "]);

    if (!toqueInterno(e)) {
        Log.d(TAG, "onSingleTapConfirmed() returned: " +
false);
        return false;
    }

    xAnterior = -1;
    yAnterior = -1;

    if (modo == Modo.CLASSE || modo == Modo.ABSTRATA || modo ==
Modo.INTERFACE) {

```

```

        Log.d(TAG, "onSingleTapConfirmed: adicionando " +
modo);

        adicionarClasse((int) e.getX() + window.left, (int)
e.getY() + window.top);
        Log.d(TAG, "onSingleTapConfirmed: x=" + e.getX() + ",
y=" + e.getY());
        Log.d(TAG, "onSingleTapConfirmed: rx=" + e.getRawX() +
", ry=" + e.getRawY());
    } else
        desseleccionarTodas();

    Log.d(TAG, "onSingleTapConfirmed() returned: " + true);

    return true;
}

@Override
public boolean onDoubleTap(MotionEvent e) {
    Log.d(TAG, "onDoubleTap() called with: e = [" + e + "]);

    return true;
}

protected void centralizar() {
    Log.d(TAG, "centralizar() called");

    int hNeg = largura / 2;
    int vNeg = altura / 2;
    int hPos = (largura % 2 == 0 ? largura / 2 : (largura + 1)
/ 2);
    int vPos = (altura % 2 == 0 ? altura / 2 : (altura + 1) /
2);

    window.set(-hNeg, -vNeg, hPos, vPos);

    invalidate();
    requestLayout();

    Log.d(TAG, "centralizar() returned: window=" + window);
}

@Override
public boolean onDoubleTapEvent(MotionEvent e) {
    Log.d(TAG, "onDoubleTapEvent() called with: e = [" + e +
"]");

    centralizar();

    return true;
}

private int xAnterior = -1;
private int yAnterior = -1;

@Override
public boolean onDown(MotionEvent e) {

```

```

        Log.d(TAG, "onDown() called with: e = [" + e + "]");

        float x = e.getX(), y = e.getY();

        xAnterior = (int) x;
        yAnterior = (int) y;

        Log.d(TAG, "onDown: x=" + xAnterior + ", y=" + yAnterior);

        return true;
    }

    @Override
    public void onShowPress(MotionEvent e) {
    }

    @Override
    public boolean onSingleTapUp(MotionEvent e) {
        Log.d(TAG, "onSingleTapUp() called with: e = [" + e + "]");

        xAnterior = -1;
        yAnterior = -1;

        return true;
    }

    @Override
    public boolean onScroll(MotionEvent e1, MotionEvent e2, float
distanceX, float distanceY) {
        Log.d(TAG, "onScroll() called with: e1 = [" + e1 + "], e2 =
[" + e2 + "], distanceX = [" + distanceX + "], distanceY = [" +
distanceY + "]");

        int x = (int) e2.getX();
        int y = (int) e2.getY();

        if (xAnterior == -1 | yAnterior == -1) Log.d(TAG,
"onScroll: MERDA");
        if (xAnterior == -1) xAnterior = (int) e1.getX();
        if (yAnterior == -1) yAnterior = (int) e1.getY();

        int difX = x - xAnterior;
        int difY = y - yAnterior;

        Log.d(TAG, "onScroll: difX=" + difX + ", difY=" + difY);

        window.set(window.left - difX, window.top - difY,
window.right - difX, window.bottom - difY);
        Log.d(TAG, "onScroll: nova window=" + window);

        xAnterior = x;
        yAnterior = y;

        invalidate();
        requestLayout();

        return true;
    }
}

```

```

@Override
public void onLongPress(MotionEvent e) {
    Log.d(TAG, "onLongPress() called with: e = [" + e + "]");
}

@Override
public boolean onFling(MotionEvent e1, MotionEvent e2, float
velocityX, float velocityY) {
    Log.d(TAG, "onFling() called with: e1 = [" + e1 + "], e2 =
[" + e2 + "], velocityX = [" + velocityX + "], velocityY = [" +
velocityY + "]");

    return true;
}

@Override
protected void onDraw(Canvas canvas) {
    Log.d(TAG, "onDraw() called with: canvas = [" + canvas +
"]");

    Paint p = new Paint();
    p.setStyle(Paint.Style.FILL);
    canvas.drawRect(canvas.getClipBounds(), p);

    super.onDraw(canvas);
}

private boolean toqueInterno(MotionEvent e) {
    Log.d(TAG, "toqueInterno() called with: e = [" + e + "]");

    int wc[] = new int[2];
    getLocationInWindow(wc);
    RectF areaEdicao = new RectF(wc[0], wc[1], wc[0] +
getWidth(), wc[1] + getHeight());

    float x = e.getX(), y = e.getY();

    if (x < areaEdicao.left || x >= areaEdicao.right || y <
areaEdicao.top || y >= areaEdicao.bottom) {
        Log.d(TAG, "toqueInterno: DENIED");

        Log.d(TAG, "toqueInterno() returned: " + false);
        return false;
    }

    Log.d(TAG, "toqueInterno() returned: " + true);
    return true;
}

@Override
public Parcelable onSaveInstanceState() {
    Log.d(TAG, "onSaveInstanceState() called");

    Parcelable p = super.onSaveInstanceState();

```

```

        Estado e = new Estado(p);

        e.classes = this.classes;
        e.posicoes = this.posicoes;
        e.window = this.window;

        return e;
    }

    @Override
    public void onRestoreInstanceState(Parcelable p) {
        Log.d(TAG, "onRestoreInstanceState() called with: p = [" +
p + "]");

        if (!(p instanceof Estado)) {
            super.onRestoreInstanceState(p);

            return;
        }

        Estado e = (Estado) p;
        super.onRestoreInstanceState(e.getSuperState());

        classes = new ArrayList<>(e.classes.size());

        for (Classe c : e.classes) {
            classes.add(new Classe(c, this));
        }

        posicoes = e.posicoes;
        window = e.window;

        for (Classe c : classes) {
            addView(c);
        }
    }

    static class Estado extends BaseSavedState {
        ArrayList<Classe> classes;
        ArrayList<Point> posicoes;
        Rect window;

        Estado(Parcelable p) {
            super(p);
        }

        private Estado(Parcel fonte) {
            super(fonte);

            this.classes =
fonte.readArrayList(Classe.class.getClassLoader());
            this.posicoes =
fonte.readArrayList(Point.class.getClassLoader());

            int l = fonte.readInt(), t = fonte.readInt(), r =
fonte.readInt(), b = fonte.readInt();
            this.window = new Rect(l, t, r, b);
        }
    }

```

```

        Log.d(TAG, "Estado: classes=" + this.classes);
        Log.d(TAG, "Estado: posicoes=" + this.posicoes);
        Log.d(TAG, "Estado: window=" + this.window);
    }

    @Override
    public void writeToParcel(Parcel out, int flags) {
        super.writeToParcel(out, flags);
        Classe[] cs = new Classe[this.classes.size()];
        out.writeTypedArray(this.classes.toArray(cs), 0);

        Point[] ps = new Point[this.posicoes.size()];
        out.writeTypedArray(this.posicoes.toArray(ps), 0);

        out.writeFloat(this.window.left);
        out.writeFloat(this.window.top);
        out.writeFloat(this.window.right);
        out.writeFloat(this.window.bottom);
    }

    public static final Parcelable.Creator<Estado> CREATOR =
        new Parcelable.Creator<Estado>() {

            @Override
            public Estado createFromParcel(Parcel fonte) {
                return new Estado(fonte);
            }

            @Override
            public Estado[] newArray(int size) {
                return new Estado[size];
            }
        };
    }
}

```

8.2.3 Classe.java

```

package com.pamplona.rodolfo.agileclass.ui;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Rect;
import android.graphics.RectF;
import android.graphics.Typeface;
import android.os.Parcel;
import android.os.Parcelable;
import android.util.Log;
import android.view.ContextMenu;
import android.view.MotionEvent;
import android.view.View;
import android.widget.TextView;

import com.pamplona.rodolfo.agileclass.R;

```

```

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;

public class Classe extends View implements
ContextMenu.ContextMenuInfo, Parcelable {
    private static final String TAG = "#ui.Classe";

    protected enum Tipo {
        CONCRETA(""), INTERFACE("<< interface >>"), ABSTRATA("<<
abstrata >>");

        private final String texto;

        Tipo(String t) {
            texto = t;
        }

        private String texto() {
            return texto;
        }
    }

    private Context contexto;

    private AreaEdicao ae;
    private int indice;

    private final long id;

    private Tipo tipo;
    private String nome;
    private ArrayList<String> campos;
    private ArrayList<String> metodos;

    private Paint tipoPaint;
    private Paint nomePaint;
    private Paint listaPaint;
    private Paint retanguloPaint;
    private Paint retanguloSelecioneadoPaint;

    private final static int corLinhas = Color.rgb(0x66, 0x66,
0x66);
    private final static int corSelecioneado = Color.rgb(0x00, 0x00,
0x00);
    private final static int corLetras = Color.rgb(0x00, 0x00,
0x00);
    private final static int corFundo = Color.rgb(0xfa, 0xfa,
0xf0); //ff fa f0

    private Rect bordas = new Rect();

    private boolean selecionada;

```

```

    public Classe(Context contexto, AreaEdicao canvas, int indice,
long id, Tipo tipo, String nome, List<String> campos, List<String>
metodos) {
        super(contexto);

        Log.d(TAG, "Classe() called with: contexto = [" + contexto
+ "], canvas = [" + canvas + "], indice = [" + indice + "], tipo =
[" + tipo + "], nome = [" + nome + "], campos = [" + campos + "],
metodos = [" + metodos + "]);

        this.contexto = contexto;

        this.ae = canvas;
        this.indice = indice;

        this.id = id;

        this.tipo = tipo;

        this.nome = nome;

        if (campos != null && !campos.isEmpty()) {
            this.campos = new ArrayList<>(campos);
        } else this.campos = new ArrayList<>();

        if (metodos != null && !metodos.isEmpty()) {
            this.metodos = new ArrayList<>(metodos);
        } else this.metodos = new ArrayList<>();

        init();

        tamanhoIdeal();
    }

    public Classe(Classe c, AreaEdicao canvas) {
        this(c.contexto, canvas, c.indice, c.id, c.tipo, c.nome,
c.campos, c.metodos);

        Log.d(TAG, "Classe() called with: c = [" + c + "], canvas =
[" + canvas + "]);
    }

    private void init() {
        Log.d(TAG, "init() called");

        TextView t = new TextView(contexto);

        t.setText(tipo.texto());
        tipoPaint = t.getPaint();
        tipoPaint.setColor(corLetras);
        tipoPaint.setTypeface(Typeface.create(Typeface.DEFAULT,
Typeface.ITALIC));
        tipoPaint.setTextAlign(Paint.Align.CENTER);

        t = new TextView(contexto);
        t.setText(nome);
        nomePaint = t.getPaint();
        nomePaint.setColor(corLetras);
    }

```

```

        nomePaint.setTypeface (Typeface.create (Typeface.DEFAULT,
Typeface.BOLD));
        nomePaint.setTextAlign (Paint.Align.CENTER);

        t = new TextView(contexto);
        if (campos != null && !campos.isEmpty()) {
            t.setText(campos.get(0));
            listaPaint = t.getPaint();
            listaPaint.setColor(corLetras);
        } else if (metodos != null && !metodos.isEmpty()) {
            t.setText(metodos.get(0));
            listaPaint = t.getPaint();
            listaPaint.setColor(corLetras);
        }

        retanguloPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
        retanguloPaint.setStrokeWidth(3);
        retanguloPaint.setColor(corLinhas);

        retanguloSelecioneadoPaint = new
Paint(Paint.ANTI_ALIAS_FLAG);
        retanguloSelecioneadoPaint.setStrokeWidth(15);
        retanguloSelecioneadoPaint.setColor(corSelecioneado);
        retanguloSelecioneadoPaint.setStyle(Paint.Style.STROKE);
    }

    public Tipo tipo() {
        return tipo;
    }

    public Rect limites() {
        return new Rect(bordas);
    }

    public int indice() {
        return indice;
    }

    public long id() {
        return id;
    }

    public void desselecionar() {
        Log.d(TAG, "desselecionar() called");

        selecionada = false;

        invalidate();
        requestLayout();
    }

    private void tamanhoIdeal() {
        Log.d(TAG, "tamanhoIdeal() called");

        List<Integer> larguraLinhas = new LinkedList<>();
        int alturaLinhas;

        Rect r = new Rect();

```

```

        nomePaint.getTextBounds(nome, 0, nome.length(), r);
        larguraLinhas.add(r.width());
        alturaLinhas = r.height() + 15;
        Log.d(TAG, "tamanhoIdeal: nome: w=" + r.width() + ", h=" +
r.height());

        if (tipo != Tipo.CONCRETA) {
            r.setEmpty();
            tipoPaint.getTextBounds(tipo.texto(),
0,
tipo.texto().length(), r);
            larguraLinhas.add(r.width());
            alturaLinhas += r.height() + 15;
            Log.d(TAG, "tamanhoIdeal: tipo: w=" + r.width() + ",
h=" + r.height());
        } else alturaLinhas += 5;

        for (String s : campos) {
            r.setEmpty();
            listaPaint.getTextBounds(s, 0, s.length(), r);
            larguraLinhas.add(r.width());
            alturaLinhas += r.height() + 10;
        }

        for (String s : metodos) {
            r.setEmpty();
            listaPaint.getTextBounds(s, 0, s.length(), r);
            larguraLinhas.add(r.width());
            alturaLinhas += r.height() + 10;
        }

        Log.d(TAG, "tamanhoIdeal: larguraLinhas=" + larguraLinhas);
        Log.d(TAG, "tamanhoIdeal: alturaLinhas=" + alturaLinhas);

        int larguraMaximaTexto = Collections.max(larguraLinhas);

        if (campos == null || campos.isEmpty())
            alturaLinhas += 15;

        if (metodos == null || metodos.isEmpty())
            alturaLinhas += 15;

        bordas = new Rect(0, 0, larguraMaximaTexto + 20,
alturaLinhas + 30);

        Log.d(TAG, "tamanhoIdeal() returned: " + bordas);
    }

    @Override
    public void onMeasure(int larguraMeasureSpec, int
alturaMeasureSpec) {
        Log.d(TAG, "onMeasure() called with: larguraMeasureSpec =
[" + larguraMeasureSpec + "], alturaMeasureSpec = [" +
alturaMeasureSpec + "]");

        int lModo = MeasureSpec.getMode(larguraMeasureSpec);
        int l = MeasureSpec.getSize(larguraMeasureSpec);

```

```

int aModo = MeasureSpec.getMode(alturaMeasureSpec);
int a = MeasureSpec.getSize(alturaMeasureSpec);

Log.d(TAG, "onMeasure: l=" + l + ", a=" + a);
Log.d(TAG, "onMeasure: lM=" + lModo + ", aM=" + aModo);

tamanhoIdeal();

Log.d(TAG, "onMeasure() returned: " + bordas);

float pxW =
getResources().getDisplayMetrics().scaledDensity * bordas.width();
float pxH =
getResources().getDisplayMetrics().scaledDensity * bordas.height();

Log.d(TAG, "onMeasure: pxW=" + pxW + ", pxH=" + pxH);

setMeasuredDimension(bordas.width(), bordas.height());
}

@Override
public void onSizeChanged(int l, int a, int lAnterior, int
aAnterior) {
    Log.d(TAG, "onSizeChanged() called with: l = [" + l + "], a
= [" + a + "], lAnterior = [" + lAnterior + "], aAnterior = [" +
aAnterior + "]");

    tamanhoIdeal();
}

private int xAnterior = -1;
private int yAnterior = -1;

private boolean movendoDedo;
private boolean moveuClasse;

@Override
public boolean onTouchEvent(MotionEvent e) {
    Log.d(TAG, "onTouchEvent() called with: e = [" + e + "]");

    if (e.getAction() == MotionEvent.ACTION_UP) {
        xAnterior = -1;
        yAnterior = -1;

        if (movendoDedo) {
            ae.redefinirPanning();
            movendoDedo = false;
            return false;
        }

        if (!selecionada) {
            selecionada = true;
            ae.selecionar(indice);
        } else if (!moveuClasse) {
            selecionada = false;
            ae.desselecionar(indice);
        } else {
            moveuClasse = false;

```

```

    }

    invalidate();
    requestLayout();
} else if (e.getAction() == MotionEvent.ACTION_MOVE) {
    if (selecionada) {
        int x = (int) e.getX();
        int y = (int) e.getY();

        int difX = x - xAnterior;
        int difY = y - yAnterior;

        if (Math.abs(difX) > 5 || Math.abs(difY) > 5) {
            ae.mover(indice, difX, difY);

            moveuClasse = true;
        }
    } else if (!movendoDedo) {
        xAnterior = -1;
        yAnterior = -1;

        ae.iniciarPanning((int) e.getRawX(), (int)
e.getRawY());
        movendoDedo = true;
        return false;
    } else {
        return false;
    }
} else if (e.getAction() == MotionEvent.ACTION_DOWN) {
    xAnterior = (int) e.getX();
    yAnterior = (int) e.getY();
} else {
    xAnterior = -1;
    yAnterior = -1;
}

return true;
}

private Rect aux = new Rect();
RectF separador1 = new RectF(), separador2 = new RectF();

@Override
public void onDraw(Canvas c) {
    Log.d(TAG, "onDraw() called with: c = [" + c + "]);

    retanguloPaint.setColor(corFundo);
    retanguloPaint.setStyle(Paint.Style.FILL);
    c.drawRect(0, 0, getWidth() + 3, getHeight() + 3,
retanguloPaint);
    Log.d(TAG, "onDraw: desenhando fundo branco em " + bordas);

    retanguloPaint.setStyle(Paint.Style.STROKE);
    retanguloPaint.setColor(corLinhas);

    if (selecionada) {
        Log.d(TAG, "onDraw: selecionada");
    }
}

```

```

        c.drawRect(0, 0, getWidth(), getHeight(),
retanguloSelecionadoPaint);
        retanguloPaint.setColor(corSelecionado);
    } else {
        c.drawRect(0, 0, getWidth(), getHeight(),
retanguloPaint);
    }

    Log.d(TAG, "onDraw: desenhando retangulo em " + bordas);

    float txtX = getWidth() / 2.f;
    float txtY = 0;
    if (tipo != Tipo.CONCRETA) {
        tipoPaint.getTextBounds(tipo.texto(), 0,
tipo.texto().length(), aux);
        txtY = aux.height() + 5;
        c.drawText(tipo.texto(), txtX, txtY, tipoPaint);
        Log.d(TAG, "onDraw: desenhando tipo em (" + txtX + ", "
+ txtY + ")");
    }

    float offsetY = txtY + 5;

    aux.setEmpty();
    nomePaint.getTextBounds(nome, 0, nome.length(), aux);
    txtY = aux.height() + offsetY;
    c.drawText(nome, txtX, txtY, nomePaint);
    Log.d(TAG, "onDraw: desenhando nome em (" + txtX + ", " +
txtY + ")");

    offsetY = txtY + 15;

    separador1.set(0, offsetY, getWidth(), offsetY);
    c.drawLine(separador1.left, separador1.top,
separador1.right, separador1.bottom, retanguloPaint);
    Log.d(TAG, "onDraw: desenhando separador 1 em " +
separador1);

    offsetY += 15;

    txtX = 10;

    if (campos != null && !campos.isEmpty()) {
        for (String f : campos) {
            aux.setEmpty();
            listaPaint.getTextBounds(f, 0, f.length(), aux);
            txtY = aux.height() - 5;

            c.drawText(f, txtX, offsetY + txtY, listaPaint);
            Log.d(TAG, "onDraw: desenhando campo \"" + f + "\"
em (" + txtX + ", " + (txtY + offsetY) + ")");
            offsetY += aux.height() + 10;
        }
    } else {
        offsetY += 15;
    }

    separador2.set(0, offsetY, getWidth(), offsetY);

```

```

        c.drawLine(separador2.left,                                separador2.top,
separador2.right, separador2.bottom, retanguloPaint);
        Log.d(TAG, "onDraw: desenhando separador 2 em " +
separador2);

        if (metodos != null && !metodos.isEmpty()) {
            offsetY += 15;
            for (String m : metodos) {
                aux.setEmpty();
                listaPaint.getTextBounds(m, 0, m.length(), aux);
                txtY = aux.height() - 5;

                c.drawText(m, txtX, offsetY + txtY, listaPaint);
                Log.d(TAG, "onDraw: desenhando metodo \"" + m + "\"
em (" + txtX + ", " + (txtY + offsetY) + ")");
                offsetY += aux.height() + 10;
            }
        }

        super.onDraw(c);
    }

    @Override
    protected ContextMenu.ContextMenuInfo getContextMenuInfo() {
        return this;
    }

    @Override
    public int describeContents() {
        return 0;
    }

    @Override
    public void writeToParcel(Parcel saida, int flags) {
        saida.writeInt(indice);

        saida.writeLong(id);

        saida.writeInt(tipo == Tipo.CONCRETA ? 0 : (tipo ==
Tipo.INTERFACE ? 1 : 2));

        saida.writeString(nome);

        saida.writeInt(this.campos.size());
        String s[] = new String[this.campos.size()];
        saida.writeStringArray(this.campos.toArray(s));

        saida.writeInt(this.metodos.size());
        s = new String[this.metodos.size()];
        saida.writeStringArray(this.metodos.toArray(s));

        saida.writeFloat(bordas.left);
        saida.writeFloat(bordas.top);
        saida.writeFloat(bordas.right);
        saida.writeFloat(bordas.bottom);
    }

```

```

private Classe(Parcel fonte) {
    super(null);

    Log.d(TAG, "Estado() called with: fonte = [" + fonte +
    "]);

    ae = (AreaEdicao) findViewById(R.id.canvas);
    Log.d(TAG, "Classe: ae=" + ae);

    indice = fonte.readInt();

    id = fonte.readLong();

    switch (fonte.readInt()) {
        case 0:
            tipo = Tipo.CONCRETA;
            break;
        case 1:
            tipo = Tipo.INTERFACE;
            break;
        case 2:
            tipo = Tipo.ABSTRATA;
            break;
    }

    nome = fonte.readString();

    int cs = fonte.readInt();
    String s[] = new String[cs];
    fonte.readStringArray(s);
    campos = new ArrayList<String>(Arrays.asList(s));

    int ms = fonte.readInt();
    s = new String[ms];
    fonte.readStringArray(s);
    metodos = new ArrayList<String>(Arrays.asList(s));

    float l = fonte.readFloat(), t = fonte.readFloat(), r =
    fonte.readFloat(), b = fonte.readFloat();
}

public static final Parcelable.Creator<Classe> CREATOR = new
Parcelable.Creator<Classe>() {
    @Override
    public Classe createFromParcel(Parcel fonte) {
        return new Classe(fonte);
    }

    @Override
    public Classe[] newArray(int size) {
        return new Classe[size];
    }
};
}

```

8.2.4 EdicaoClasse.java

```
package com.pamplona.rodolfo.agileclass.ui;

import android.content.Context;
import android.net.Uri;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.util.Log;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;

import com.pamplona.rodolfo.agileclass.R;

public class EdicaoClasse extends Fragment {
    private static final String TAG = "#ui.EdicaoClasse";

    // TODO: Rename parameter arguments, choose names that match
    // the fragment initialization parameters, e.g. ARG_ITEM_NUMBER
    private static final String ARG_PARAM1 = "param1";
    private static final String ARG_PARAM2 = "param2";

    // TODO: Rename and change types of parameters
    private String mParam1;
    private String mParam2;

    private OnFragmentInteractionListener mListener;

    public EdicaoClasse() {
        // Required empty public constructor
    }

    /**
     * Use this factory method to create a new instance of
     * this fragment using the provided parameters.
     *
     * @param param1 Parameter 1.
     * @param param2 Parameter 2.
     * @return A new instance of fragment EdicaoClasse.
     */
    // TODO: Rename and change types and number of parameters
    public static EdicaoClasse newInstance(String param1, String
param2) {
        Log.d(TAG, "newInstance() called with: param1 = [" + param1
+ "], param2 = [" + param2 + "]);

        EdicaoClasse fragment = new EdicaoClasse();
        Bundle args = new Bundle();
        args.putString(ARG_PARAM1, param1);
        args.putString(ARG_PARAM2, param2);
        fragment.setArguments(args);
        return fragment;
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
```

```

        super.onCreate(savedInstanceState);
        Log.d(TAG, "onCreate() called with: savedInstanceState = ["
+ savedInstanceState + "]);

        if (getArguments() != null) {
            mParam1 = getArguments().getString(ARG_PARAM1);
            mParam2 = getArguments().getString(ARG_PARAM2);
        }

        Log.d(TAG, "onCreate() returned: " + "fim");
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup
container,
                               Bundle savedInstanceState) {
        Log.d(TAG, "onCreateView() called with: inflater = [" +
inflater + "], container = [" + container + "], savedInstanceState
= [" + savedInstanceState + "]);

        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_edicao_classe,
container, false);
    }

    // TODO: Rename method, update argument and hook method into UI
event
    public void onPressed(Uri uri) {
        Log.d(TAG, "onPressed() called with: uri = [" + uri +
""]);

        if (mListener != null) {
            mListener.onFragmentInteraction(uri);
        }
    }

    @Override
    public void onAttach(Context context) {
        Log.d(TAG, "onAttach() called with: context = [" + context
+ "]);

        super.onAttach(context);
        if (context instanceof OnFragmentInteractionListener) {
            mListener = (OnFragmentInteractionListener) context;
        } else {
            throw new RuntimeException(context.toString()
+
" must implement
OnFragmentInteractionListener");
        }

        Log.d(TAG, "onAttach() returned: " + "fim");
    }

    @Override
    public void onDetach() {
        Log.d(TAG, "onDetach() called");

        super.onDetach();
    }

```

```

        mListener = null;
    }

    /**
     * This interface must be implemented by activities that
     * contain this
     * fragment to allow an interaction in this fragment to be
     * communicated
     * to the activity and potentially other fragments contained in
     * that
     * activity.
     * <p>
     * See the Android Training lesson <a href=
     * "http://developer.android.com/training/basics/fragments/communicati
     * ng.html"
     * >Communicating with Other Fragments</a> for more
     * information.
     */
    public interface OnFragmentInteractionListener {
        // TODO: Update argument type and name
        void onFragmentInteraction(Uri uri);
    }
}

```

8.2.5 GrupoBotoes.java

```

package com.pamplona.rodolfo.agileclass.ui;

import android.support.v7.widget.AppCompatButton;
import android.util.Log;
import android.view.MotionEvent;
import android.view.View;

import java.util.HashMap;
import java.util.Map;

public class GrupoBotoes {
    private static final String TAG = "#ui.GrupoBotoes";

    private final AgileClass ac;
    private final Map<AppCompatButton, AreaEdicao.Modos> botoes;

    public GrupoBotoes(AgileClass ac, Map<AppCompatButton,
AreaEdicao.Modos> botoes) {
        Log.d(TAG, "GrupoBotoes() called with: ac = [" + ac + "],
botoes = [" + botoes + "]");

        this.ac = ac;
        this.botoes = new HashMap<>(botoes);

        for (final AppCompatButton b : this.botoes.keySet()) {
            b.setOnTouchListener(new View.OnTouchListener() {

                @Override
                public boolean onTouch(View v, MotionEvent e) {

```

```

        return toque(b, e);
    }
    });
}

protected boolean toque(AppCompatButton b, MotionEvent e) {
    Log.d(TAG, "toque() called with: b = [" + b + "], e = [" +
e + "]");

    if (e.getAction() != MotionEvent.ACTION_UP) {
        return true;
    }

    if (b.isPressed()) {
        b.setPressed(false);

        ac.moda(null);

        return true;
    }

    for (AppCompatButton x : botoes.keySet())
        x.setPressed(false);

    b.setPressed(true);

    ac.moda(botoes.get(b));

    return true;
}

protected void desselecionarTodos() {
    Log.d(TAG, "desselecionarTodos() called");

    for (AppCompatButton b : botoes.keySet())
        b.setPressed(false);
}
}

```

8.3 Diretório:

app/src/main/java/com/pamplona/rodolfo/agileclass/ui/relacoes

8.3.1 Agregacao.java

```

package com.pamplona.rodolfo.agileclass.ui.relacoes;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.Rect;
import android.util.Log;
import android.widget.TextView;

import com.pamplona.rodolfo.agileclass.ui.AreaEdicao;

```

```

import com.pamplona.rodolfo.agileclass.ui.Classe;

public class Agregacao extends Associacao {
    private static final String TAG = "#ui.Agregacao";

    private Path losangulo;

    public Agregacao(Context context, AreaEdicao ae, long id,
        Classe c1, Classe c2) {
        super(context, ae, id, c1, c2);

        Log.d(TAG, "Agregacao() called with: context = [" + context
            + "], ae = [" + ae + "], c1 = [" + c1 + "], c2 = [" + c2 + "]);

        losangulo = new Path();
    }

    @Override
    protected void desenhar(Canvas canvas) {
        Log.d(TAG, "desenhar() called with: canvas = [" + canvas +
            "]);

        int ox = origem.x - limites.left, oy = origem.y -
            limites.top;
        int dx = destino.x - limites.left, dy = destino.y -
            limites.top;

        Log.d(TAG, "desenhar: especificamente em (" + ox + ", " +
            oy + ", " + dx + ", " + dy + "));

        double angulo = Math.atan2(dx - ox, dy - oy);

        double comprimentoTriangulo = Math.min(comprimento(ox, oy,
            dx, dy) * .75, 30);
        double baseTrianguloY = dy - comprimentoTriangulo *
            Math.cos(angulo);
        double baseTrianguloX = dx - comprimentoTriangulo *
            Math.sin(angulo);

        int fimDaLinhaY = dy - (int) (comprimentoTriangulo *
            Math.cos(angulo) * 2);
        int fimDaLinhaX = dx - (int) (comprimentoTriangulo *
            Math.sin(angulo) * 2);

        Log.d(TAG, "desenhar: base=(" + baseTrianguloX + ", " +
            baseTrianguloY);

        canvas.drawLine(ox, oy, fimDaLinhaX, fimDaLinhaY,
            linhaPaint);

        if (comprimentoTriangulo * 2 <= 1)
            return;

        double baseXnorm = (dx - baseTrianguloX) /
            comprimento(baseTrianguloX, baseTrianguloY, dx, dy);
        double baseYnorm = (dy - baseTrianguloY) /
            comprimento(baseTrianguloX, baseTrianguloY, dx, dy);

```

```

        double tmp = baseXnorm;
        baseXnorm = -baseYnorm;
        baseYnorm = tmp;

        double ponto1x = (comprimentoTriangulo * .5) * baseXnorm +
baseTrianguloX;
        double ponto1y = (comprimentoTriangulo * .5) * baseYnorm +
baseTrianguloY;

        Log.d(TAG, "desenhar: ponto0=(" + dx + ", " + dy + ")");
        Log.d(TAG, "desenhar: ponto1=(" + ponto1x + ", " + ponto1y
+ ")");

        double ponto2x = -(comprimentoTriangulo * .5) * baseXnorm +
baseTrianguloX;
        double ponto2y = -(comprimentoTriangulo * .5) * baseYnorm +
baseTrianguloY;

        Log.d(TAG, "desenhar: ponto2=(" + ponto2x + ", " + ponto2y
+ ")");

        Log.d(TAG, "desenhar: ponto3=(" + fimDaLinhaX + ", " +
fimDaLinhaY + ")");

        losangulo.rewind();

        losangulo.moveTo(fimDaLinhaX, fimDaLinhaY);
        losangulo.lineTo((float) ponto1x, (float) ponto1y);
        losangulo.lineTo(dx, dy);
        losangulo.lineTo((float) ponto2x, (float) ponto2y);
        losangulo.close();

        canvas.drawPath(losangulo, linhaPaint);

        desenharMultiplicidade(canvas);

        Log.d(TAG, "desenhar() returned: " + "fim");
    }
}

```

8.3.2 Associacao.java

```

package com.pamplona.rodolfo.agileclass.ui.relacoes;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Rect;
import android.util.Log;
import android.widget.TextView;

import com.pamplona.rodolfo.agileclass.ui.AreaEdicao;
import com.pamplona.rodolfo.agileclass.ui.Classe;

public class Associacao extends Relacionamento {
    private static final String TAG = "#ui.Associacao";

```

```

    public Associacao(Context context, AreaEdicao ae, long id,
Classe c1, Classe c2) {
        super(context, ae, id, c1, c2);

        Log.d(TAG, "Associacao() called with: context = [" +
context + "], ae = [" + ae + "], id = [" + id + "], c1 = [" + c1 +
"], c2 = [" + c2 + "]);

        aux = new TextView(context);
    }

    private final Rect rM1 = new Rect(), rM2 = new Rect();
    private final TextView aux;

    protected void desenharMultiplicidade(Canvas c) {
        Log.d(TAG, "desenharMultiplicidade() called with: c = [" +
c + "]);

        String[] ms =
ae.diagrama().multiplicidadesDaAssociacao(id);

        String m1 = ms[0];
        String m2 = ms[1];

        aux.setText(m1);
        Paint p1 = aux.getPaint();
        p1.getTextBounds(m1, 0, m1.length(), rM1);
        aux.setText(m2);
        Paint p2 = aux.getPaint();
        p2.getTextBounds(m2, 0, m2.length(), rM2);

        int ox = origem.x - limites.left, oy = origem.y -
limites.top;
        int dx = destino.x - limites.left, dy = destino.y -
limites.top;

        Log.d(TAG, "desenharMultiplicidade: o=(" + ox + ", " + oy +
")");
        Log.d(TAG, "desenharMultiplicidade: d=(" + dx + ", " + dy +
")");
        Log.d(TAG, "desenharMultiplicidade: 1=" + posicao1);
        Log.d(TAG, "desenharMultiplicidade: 2=" + posicao2);
        Log.d(TAG, "desenharMultiplicidade: oHV=" +
Integer.toBinaryString(oHV));
        Log.d(TAG, "desenharMultiplicidade: dHV=" +
Integer.toBinaryString(dHV));

        int offset = 35;

        if (!aEsquerda(oHV) && aDireita(oHV)) {
            Log.d(TAG, "desenharMultiplicidade: 1");
            rM1.set(ox - rM1.width() - offset, rM1.top, ox -
rM1.width() - offset, rM1.bottom);
        } else if (aEsquerda(oHV) && !aDireita(oHV)) {
            Log.d(TAG, "desenharMultiplicidade: 2");
            rM1.set(ox + offset, rM1.top, ox + offset, rM1.bottom);
        } else {

```

```

        Log.d(TAG, "desenharMultiplicidade: 3");
        rM1.set(ox, rM1.top, ox, rM1.bottom);
    }

    if (abaixo(oHV) && !acima(oHV)) {
        Log.d(TAG, "desenharMultiplicidade: 4");
        rM1.set(rM1.left, oy - offset, rM1.right, oy - offset);
    } else if (!abaixo(oHV) && acima(oHV)) {
        Log.d(TAG, "desenharMultiplicidade: 5");
        rM1.set(rM1.left, oy + rM1.height() + offset,
rM1.right, oy + rM1.height() + offset);
    } else {
        Log.d(TAG, "desenharMultiplicidade: 6");
        rM1.set(rM1.left, oy, rM1.right, oy);
    }

    c.drawText(m1, rM1.left, rM1.bottom, p1);

    Log.d(TAG, "desenharMultiplicidade: rm1=" + rM1);

    if (!aEsquerda(dHV) && aDireita(dHV)) {
        Log.d(TAG, "desenharMultiplicidade: 7");
        rM2.set(dx - rM2.width() - offset, rM2.top, dx -
rM2.width() - offset, rM2.bottom);
    } else if (aEsquerda(dHV) && !aDireita(dHV)) {
        Log.d(TAG, "desenharMultiplicidade: 8");
        rM2.set(dx + offset, rM2.top, dx + offset, rM2.bottom);
    } else {
        Log.d(TAG, "desenharMultiplicidade: 9");
        rM2.set(dx, rM2.top, dx, rM2.bottom);
    }

    if (abaixo(dHV) && !acima(dHV)) {
        Log.d(TAG, "desenharMultiplicidade: 10");
        rM2.set(rM2.left, dy - offset, rM2.right, dy - offset);
    } else if (!abaixo(dHV) && acima(dHV)) {
        Log.d(TAG, "desenharMultiplicidade: 11");
        rM2.set(rM2.left, dy + rM2.height() + offset,
rM2.right, dy + rM2.height() + offset);
    } else {
        Log.d(TAG, "desenharMultiplicidade: 12");
        rM2.set(rM2.left, dy, rM2.right, dy);
    }

    c.drawText(m2, rM2.left, rM2.bottom, p2);

    Log.d(TAG, "desenharMultiplicidade: rm2=" + rM2);
}

@Override
protected void desenhar(Canvas canvas) {
    Log.d(TAG, "desenhar() called with: canvas = [" + canvas +
    "]);

    Log.d(TAG, "desenhar: limites=" + limites);

    int ox = origem.x - limites.left, oy = origem.y -
    limites.top;

```

```

        int dx = destino.x - limites.left, dy = destino.y -
limites.top;

        Log.d(TAG, "desenhar: especificamente em (" + ox + ", " +
oy + ", " + dx + ", " + dy + ")");

        canvas.drawLine(ox, oy, dx, dy, linhaPaint);

        double angulo = Math.atan2(dx - ox, dy - oy);

        double comprimentoSeta = Math.min(comprimento(ox, oy, dx,
dy) * .75, 30);
        double baseSetaY = dy - comprimentoSeta * Math.cos(angulo);
        double baseSetaX = dx - comprimentoSeta * Math.sin(angulo);

        if (comprimentoSeta <= 1)
            return;

        double baseXnorm = (dx - baseSetaX) /
comprimento(baseSetaX, baseSetaY, dx, dy);
        double baseYnorm = (dy - baseSetaY) /
comprimento(baseSetaX, baseSetaY, dx, dy);

        double tmp = baseXnorm;
        baseXnorm = -baseYnorm;
        baseYnorm = tmp;

        double ponto1x = (comprimentoSeta * .75) * baseXnorm +
baseSetaX;
        double ponto1y = (comprimentoSeta * .75) * baseYnorm +
baseSetaY;

        Log.d(TAG, "desenhar: ponto0=(" + dx + ", " + dy + ")");
        Log.d(TAG, "desenhar: ponto1=(" + ponto1x + ", " + ponto1y
+ ")");

        double ponto2x = -(comprimentoSeta * .75) * baseXnorm +
baseSetaX;
        double ponto2y = -(comprimentoSeta * .75) * baseYnorm +
baseSetaY;

        Log.d(TAG, "desenhar: ponto2=(" + ponto2x + ", " + ponto2y
+ ")");

        canvas.drawLine(dx, dy, (float) ponto1x, (float) ponto1y,
linhaPaint);
        canvas.drawLine(dx, dy, (float) ponto2x, (float) ponto2y,
linhaPaint);

        desenharMultiplicidade(canvas);

        Log.d(TAG, "desenhar() returned: " + "fim");
    }
}

```

8.3.3 Composicao.java

```
package com.pamplona.rodolfo.agileclass.ui.relacoes;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Path;
import android.util.Log;

import com.pamplona.rodolfo.agileclass.ui.AreaEdicao;
import com.pamplona.rodolfo.agileclass.ui.Classe;

public class Composicao extends Associacao {
    private static final String TAG = "#ui.Composicao";

    private Path losangulo;

    public Composicao(Context context, AreaEdicao ae, long id,
        Classe c1, Classe c2) {
        super(context, ae, id, c1, c2);

        Log.d(TAG, "Composicao() called with: context = [" +
            context + "], ae = [" + ae + "], c1 = [" + c1 + "], c2 = [" + c2 +
            "]);

        losangulo = new Path();
    }

    @Override
    protected void desenhar(Canvas canvas) {
        Log.d(TAG, "desenhar() called with: canvas = [" + canvas +
            "]);

        int ox = origem.x - limites.left, oy = origem.y -
            limites.top;
        int dx = destino.x - limites.left, dy = destino.y -
            limites.top;

        Log.d(TAG, "desenhar: especificamente em (" + ox + ", " +
            oy + ", " + dx + ", " + dy + "));

        double angulo = Math.atan2(dx - ox, dy - oy);

        double comprimentoTriangulo = Math.min(comprimento(ox, oy,
            dx, dy) * .75, 30);
        double baseTrianguloY = dy - comprimentoTriangulo *
            Math.cos(angulo);
        double baseTrianguloX = dx - comprimentoTriangulo *
            Math.sin(angulo);

        int fimDaLinhaY = dy - (int) (comprimentoTriangulo *
            Math.cos(angulo) * 2);
        int fimDaLinhaX = dx - (int) (comprimentoTriangulo *
            Math.sin(angulo) * 2);
```

```

        Log.d(TAG, "desenhar: base=(" + baseTrianguloX + ", " +
baseTrianguloY);

        canvas.drawLine(ox, oy, fimDaLinhaX, fimDaLinhaY,
linhaPaint);

        if (comprimentoTriangulo * 2 <= 1)
            return;

        linhaPaint.setStyle(Paint.Style.FILL_AND_STROKE);

        double baseXnorm = (dx - baseTrianguloX) /
comprimento(baseTrianguloX, baseTrianguloY, dx, dy);
        double baseYnorm = (dy - baseTrianguloY) /
comprimento(baseTrianguloX, baseTrianguloY, dx, dy);

        double tmp = baseXnorm;
        baseXnorm = -baseYnorm;
        baseYnorm = tmp;

        double ponto1x = (comprimentoTriangulo * .5) * baseXnorm +
baseTrianguloX;
        double ponto1y = (comprimentoTriangulo * .5) * baseYnorm +
baseTrianguloY;

        Log.d(TAG, "desenhar: ponto0=(" + dx + ", " + dy + ")");
        Log.d(TAG, "desenhar: ponto1=(" + ponto1x + ", " + ponto1y
+ ")");

        double ponto2x = -(comprimentoTriangulo * .5) * baseXnorm +
baseTrianguloX;
        double ponto2y = -(comprimentoTriangulo * .5) * baseYnorm +
baseTrianguloY;

        Log.d(TAG, "desenhar: ponto2=(" + ponto2x + ", " + ponto2y
+ ")");

        Log.d(TAG, "desenhar: ponto3=(" + fimDaLinhaX + ", " +
fimDaLinhaY + ")");

        losangulo.rewind();

        losangulo.moveTo(fimDaLinhaX, fimDaLinhaY);
        losangulo.lineTo((float) ponto1x, (float) ponto1y);
        losangulo.lineTo(dx, dy);
        losangulo.lineTo((float) ponto2x, (float) ponto2y);
        losangulo.close();

        canvas.drawPath(losangulo, linhaPaint);

        desenharMultiplicidade(canvas);

        Log.d(TAG, "desenhar() returned: " + "fim");
    }
}

```

8.3.4 Heranca.java

```
package com.pamplona.rodolfo.agileclass.ui.relacoes;

import android.content.Context;
import android.graphics.Canvas;
import android.util.Log;

import com.pamplona.rodolfo.agileclass.ui.AreaEdicao;
import com.pamplona.rodolfo.agileclass.ui.Classe;

public class Heranca extends Relacionamento {
    private static final String TAG = "#ui.Generalizacao";

    public Heranca(Context context, AreaEdicao ae, long id, Classe
c1, Classe c2) {
        super(context, ae, id, c1, c2);

        Log.d(TAG, "Generalizacao() called with: context = [" +
context + "], ae = [" + ae + "], c1 = [" + c1 + "], c2 = [" + c2 +
""]");
    }

    @Override
    protected void desenhar(Canvas canvas) {
        Log.d(TAG, "desenhar() called with: canvas = [" + canvas +
""]");

        int ox = origem.x - limites.left, oy = origem.y -
limites.top;
        int dx = destino.x - limites.left, dy = destino.y -
limites.top;

        double angulo = Math.atan2(dx - ox, dy - oy);

        double comprimentoSeta = Math.min(comprimento(ox, oy, dx,
dy) * .75, 30);
        double baseSetaY = dy - comprimentoSeta * Math.cos(angulo);
        double baseSetaX = dx - comprimentoSeta * Math.sin(angulo);

        Log.d(TAG, "desenhar: base=(" + baseSetaX + ", " +
baseSetaY);

        canvas.drawLine(ox, oy, (int) baseSetaX, (int) baseSetaY,
linhaPaint);

        if (comprimentoSeta <= 1)
            return;

        double baseXnorm = (dx - baseSetaX) /
comprimento(baseSetaX, baseSetaY, dx, dy);
        double baseYnorm = (dy - baseSetaY) /
comprimento(baseSetaX, baseSetaY, dx, dy);

        double tmp = baseXnorm;
        baseXnorm = -baseYnorm;
        baseYnorm = tmp;
    }
}
```

```

        double ponto1x = (comprimentoSeta * .75) * baseXnorm +
baseSetaX;
        double ponto1y = (comprimentoSeta * .75) * baseYnorm +
baseSetaY;

        Log.d(TAG, "desenhar: ponto0=(" + dx + ", " + dy + ")");
        Log.d(TAG, "desenhar: ponto1=(" + ponto1x + ", " + ponto1y
+ ")");

        double ponto2x = -(comprimentoSeta * .75) * baseXnorm +
baseSetaX;
        double ponto2y = -(comprimentoSeta * .75) * baseYnorm +
baseSetaY;

        Log.d(TAG, "desenhar: ponto2=(" + ponto2x + ", " + ponto2y
+ ")");

        canvas.drawLine(dx, dy, (float) ponto1x, (float) ponto1y,
linhaPaint);
        canvas.drawLine((float) ponto1x, (float) ponto1y, (float)
ponto2x, (float) ponto2y, linhaPaint);
        canvas.drawLine((float) ponto2x, (float) ponto2y, dx, dy,
linhaPaint);

        Log.d(TAG, "desenhar() returned: " + "fim");
    }
}

```

8.3.5 Implementacao.java

```

package com.pamplona.rodolfo.agileclass.ui.relacoes;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.DashPathEffect;
import android.graphics.Paint;
import android.graphics.Path;
import android.util.Log;

import com.pamplona.rodolfo.agileclass.ui.AreaEdicao;
import com.pamplona.rodolfo.agileclass.ui.Classe;

public class Implementacao extends Relacionamento {
    private static final String TAG = "#ui.Implementacao";

    private Path tracejado;

    private DashPathEffect efeito = new DashPathEffect(new
float[]{50, 20}, 0);

    public Implementacao(Context context, AreaEdicao ae, long id,
Classe c1, Classe c2) {
        super(context, ae, id, c1, c2);
    }
}

```

```

        Log.d(TAG, "Implementacao() called with: context = [" +
context + "], ae = [" + ae + "], c1 = [" + c1 + "], c2 = [" + c2 +
"]");

        tracejado = new Path();

        linhaPaint.setStyle(Paint.Style.STROKE);
        linhaPaint.setPathEffect(efeito);
    }

    private int c = -1;

    @Override
    protected void desenhar(Canvas canvas) {
        Log.d(TAG, "desenhar() called with: canvas = [" + canvas +
"]");

        int ox = origem.x - limites.left, oy = origem.y -
limites.top;
        int dx = destino.x - limites.left, dy = destino.y -
limites.top;

        double angulo = Math.atan2(dx - ox, dy - oy);

        double comprimentoSeta = Math.min(comprimento(ox, oy, dx,
dy) * .75, 30);
        double baseSetaY = dy - comprimentoSeta * Math.cos(angulo);
        double baseSetaX = dx - comprimentoSeta * Math.sin(angulo);

        Log.d(TAG, "desenhar: base=(" + baseSetaX + ", " +
baseSetaY);

        tracejado.rewind();
        tracejado.moveTo(ox, oy);
        tracejado.lineTo((float) baseSetaX, (float) baseSetaY);

        canvas.drawPath(tracejado, linhaPaint);

        if (comprimentoSeta <= 1)
            return;

        double baseXnorm = (dx - baseSetaX) /
comprimento(baseSetaX, baseSetaY, dx, dy);
        double baseYnorm = (dy - baseSetaY) /
comprimento(baseSetaX, baseSetaY, dx, dy);

        double tmp = baseXnorm;
        baseXnorm = -baseYnorm;
        baseYnorm = tmp;

        double ponto1x = (comprimentoSeta * .75) * baseXnorm +
baseSetaX;
        double ponto1y = (comprimentoSeta * .75) * baseYnorm +
baseSetaY;

        Log.d(TAG, "desenhar: ponto0=(" + dx + ", " + dy + ")");
        Log.d(TAG, "desenhar: ponto1=(" + ponto1x + ", " + ponto1y
+ ")");
    }

```

```

        double ponto2x = -(comprimentoSeta * .75) * baseXnorm +
baseSetaX;
        double ponto2y = -(comprimentoSeta * .75) * baseYnorm +
baseSetaY;

        Log.d(TAG, "desenhar: ponto2=(" + ponto2x + ", " + ponto2y
+ ")");

        canvas.drawLine(dx, dy, (float) ponto1x, (float) ponto1y,
linhaPaint);
        canvas.drawLine((float) ponto1x, (float) ponto1y, (float)
ponto2x, (float) ponto2y, linhaPaint);
        canvas.drawLine((float) ponto2x, (float) ponto2y, dx, dy,
linhaPaint);

        Log.d(TAG, "desenhar() returned: " + "fim");
    }
}

```

8.3.6 Relacionamento.java

```

package com.pamplona.rodolfo.agileclass.ui.relacoes;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.Rect;
import android.util.Log;
import android.view.View;

import com.pamplona.rodolfo.agileclass.ui.AreaEdicao;
import com.pamplona.rodolfo.agileclass.ui.Classe;

public abstract class Relacionamento extends View {
    private static final String TAG = "#ui.Relacionamento";

    protected final AreaEdicao ae;

    protected final long id;

    protected Rect limites = new Rect();

    protected Classe c1;
    protected Classe c2;

    protected Rect posicao1 = new Rect();
    protected Rect posicao2 = new Rect();

    protected byte oHV;
    protected byte dHV;

    private Rect posicaoEsq;
    private Rect posicaoDir;

```

```

private Rect posicaoAcima;
private Rect posicaoAbaixo;

private Point centro1;
private Point centro2;

protected Paint linhaPaint;

protected Point origem = new Point();
protected Point destino = new Point();

public Relacionamento(Context context, AreaEdicao ae, long
idNoDiagrama, Classe c1, Classe c2) {
    super(context);

    Log.d(TAG, "Relacionamento() called with: context = [" +
context + "], ae = [" + ae + "], idNoDiagrama = [" + idNoDiagrama +
"], c1 = [" + c1 + "], c2 = [" + c2 + "]);

    if (c1 == null) throw new NullPointerException("Classe c1
não pode ser nula");
    if (c2 == null) throw new NullPointerException("Classe c2
não pode ser nula");

    this.ae = ae;

    this.id = idNoDiagrama;

    this.c1 = c1;
    this.c2 = c2;

    ae.posicao(c1, posicao1);
    ae.posicao(c2, posicao2);

    centro1 = new Point(posicao1.centerX(),
posicao1.centerY());
    centro2 = new Point(posicao2.centerX(),
posicao2.centerY());

    linhaPaint = new Paint(Paint.ANTI_ALIAS_FLAG);
    linhaPaint.setStrokeWidth(5);
    linhaPaint.setColor(Color.BLACK);
    linhaPaint.setStyle(Paint.Style.STROKE);
}

public Classe c1() {
    return c1;
}

public Classe c2() {
    return c2;
}

public Rect limites() {
    return limites;
}

```

```

protected abstract void desenhar(Canvas canvas);

protected boolean aEsquerda(byte hv) {
    return (hv & 0b0010) != 0;
}

protected boolean aDireita(byte hv) {
    return (hv & 0b0001) != 0;
}

protected boolean acima(byte hv) {
    return (hv & 0b1000) != 0;
}

protected boolean abaixo(byte hv) {
    return (hv & 0b0100) != 0;
}

private Point origem() {
    Log.d(TAG, "origem() called");

    int x;
    int y;

    oHV = 0b0;

    if (centro1.x < centro2.x) {
        Log.d(TAG, "origem: c1 a esquerda de c2");

        oHV |= 0b0010;

        posicaoEsq = posicao1;
        posicaoDir = posicao2;

        x = (posicao1.right);
    } else if (centro1.x > centro2.x) {
        Log.d(TAG, "origem: c1 a direita de c2");

        oHV |= 0b0001;

        posicaoEsq = posicao2;
        posicaoDir = posicao1;

        x = posicao1.left;
    } else {
        Log.d(TAG, "origem: c1 e c2 com mesmo x");

        oHV |= 0;

        posicaoEsq = posicao1;
        posicaoDir = posicao2;

        x = posicao1.right;
    }

    if (centro1.y < centro2.y) {

```

```

        Log.d(TAG, "origem: c1 acima de c2");

        oHV |= 0b1000;

        posicaoAcima = posicao1;
        posicaoAbaixo = posicao2;

        y = posicao1.bottom;
    } else if (centro1.y > centro2.y) {
        Log.d(TAG, "origem: c1 abaixo de c2");

        oHV |= 0b0100;

        posicaoAcima = posicao2;
        posicaoAbaixo = posicao1;

        y = posicao1.top;
    } else {
        Log.d(TAG, "origem: c1 e c2 com mesmo y");

        oHV |= 0;

        posicaoAcima = posicao1;
        posicaoAbaixo = posicao2;

        y = posicao1.bottom;
    }

    origem.set(x, y);

    return origem;
}

private Point destino() {
    Log.d(TAG, "destino() called");

    int x;
    int y;

    dHV = 0b0;
    if (centro1.x < centro2.x) {
        Log.d(TAG, "destino: c2 a direita de c1");

        dHV |= 0b0001;

        x = posicao2.left;
    } else if (centro1.x > centro2.x) {
        Log.d(TAG, "destino: c2 a esquerda de c1");

        dHV |= 0b0010;

        x = posicao2.right;
    } else {
        Log.d(TAG, "destino: c2 e c1 com mesmo x");

        dHV |= 0;

        x = posicao2.left;
    }
}

```

```

    }

    if (centro1.y < centro2.y) {
        Log.d(TAG, "destino: c2 abaixo de c1");

        dHV |= 0b0100;

        y = posicao2.top;
    } else if (centro1.y > centro2.y) {
        Log.d(TAG, "destino: c2 acima de c1");

        dHV |= 0b1000;

        y = posicao2.bottom;
    } else {
        Log.d(TAG, "destino: c2 e c1 com mesmo y");

        dHV |= 0;

        y = posicao2.top;
    }

    destino.set(x, y);

    return destino;
}

private boolean intersecaoEmX() {
    return posicaoEsq.right > posicaoDir.left;
}

private boolean intersecaoEmY() {
    return posicaoAcima.bottom > posicaoAbaixo.top;
}

private boolean intersecao() {
    return intersecaoEmX() && intersecaoEmY();
}

protected static double comprimento(double ox, double oy,
double dx, double dy) {
    return Math.sqrt((dx - ox) * (dx - ox) + (dy - oy) * (dy -
oy));
}

@Override
protected final void onDraw(Canvas canvas) {
    Log.d(TAG, "onDraw() called with: canvas = [" + canvas +
    "]);

    definirTamanho();

    ae.posicao(c1, posicao1);
    ae.posicao(c2, posicao2);

    centro1.set(posicao1.centerX(), posicao1.centerY());
    centro2.set(posicao2.centerX(), posicao2.centerY());

```

```

        if (!intersecao()) {
            Log.d(TAG, "onDraw: sem intersecção");

            Log.d(TAG, "onDraw: intersecção em x=" +
intersecaoEmX());
            Log.d(TAG, "onDraw: intersecção em y=" +
intersecaoEmY());

            origem();
            destino();
        }

        if (centro1.x == centro2.x) {
            if (posicao1 == posicaoAcima) {
                Log.d(TAG, "onDraw: origem=(" + centro1.x + ", " +
posicao1.bottom + ")");
                Log.d(TAG, "onDraw: destino=(" + centro2.x + ", " +
posicao2.top + ")");

                origem.set(centro1.x, posicao1.bottom);
                destino.set(centro2.x, posicao2.top);
            } else {
                Log.d(TAG, "onDraw: origem=(" + centro1.x + ", " +
posicao1.top + ")");
                Log.d(TAG, "onDraw: destino=(" + centro2.x + ", " +
posicao2.bottom + ")");

                origem.set(centro1.x, posicao1.top);
                destino.set(centro2.x, posicao2.bottom);
            }
        } else if (centro1.y == centro2.y) {
            if (posicao1 == posicaoEsq) {
                Log.d(TAG, "onDraw: origem=(" + posicao1.right + ",
" + centro1.y + ")");
                Log.d(TAG, "onDraw: destino=(" + posicao2.left + ",
" + centro2.y + ")");

                origem.set(posicao1.right, centro1.y);
                destino.set(posicao2.left, centro2.y);
            } else {
                Log.d(TAG, "onDraw: origem=(" + posicao1.left + ",
" + centro1.y + ")");
                Log.d(TAG, "onDraw: destino=(" + posicao2.right +
", " + centro2.y + ")");

                origem.set(posicao1.left, centro1.y);
                destino.set(posicao2.right, centro2.y);
            }
        } else {
            if (centro1.y >= posicao2.top && centro1.y <=
posicao2.bottom) {
                if (posicao1 == posicaoEsq) {
                    Log.d(TAG, "onDraw: origem=(" + posicao1.right
+ ", " + centro1.y + ")");
                    Log.d(TAG, "onDraw: destino=(" + posicao2.left
+ ", " + centro1.y + ")");
                }
            }
        }
    }
}

```

```

        origem.set(posicao1.right, centro1.y);
        destino.set(posicao2.left, centro1.y);
    } else {
        Log.d(TAG, "onDraw: origem=(" + posicao1.left +
", " + centro1.y + ")");
        Log.d(TAG, "onDraw: destino=(" + posicao2.right
+ ", " + centro1.y + ")");

        origem.set(posicao1.left, centro1.y);
        destino.set(posicao2.right, centro1.y);
    }
    } else if (centro2.y >= posicao1.top && centro2.y <=
posicao1.bottom) {
        if (posicao1 == posicaoEsq) {
            Log.d(TAG, "onDraw: origem=(" + posicao1.right
+ ", " + centro2.y + ")");
            Log.d(TAG, "onDraw: destino=(" + posicao2.left
+ ", " + centro2.y + ")");

            origem.set(posicao1.right, centro2.y);
            destino.set(posicao2.left, centro2.y);
        } else {
            Log.d(TAG, "onDraw: origem=(" + posicao1.left +
", " + centro2.y + ")");
            Log.d(TAG, "onDraw: destino=(" + posicao2.right
+ ", " + centro2.y + ")");

            origem.set(posicao1.left, centro2.y);
            destino.set(posicao2.right, centro2.y);
        }
    } else if (centro1.x >= posicao2.left && centro1.x <=
posicao2.right) {
        if (posicao1 == posicaoAcima) {
            Log.d(TAG, "onDraw: origem=(" + centro1.x + ",
" + posicao1.bottom + ")");
            Log.d(TAG, "onDraw: destino=(" + centro1.x + ",
" + posicao2.top + ")");

            origem.set(centro1.x, posicao1.bottom);
            destino.set(centro1.x, posicao2.top);
        } else {
            Log.d(TAG, "onDraw: origem=(" + centro1.x + ",
" + posicao1.top + ")");
            Log.d(TAG, "onDraw: destino=(" + centro1.x + ",
" + posicao2.bottom + ")");

            origem.set(centro1.x, posicao1.top);
            destino.set(centro1.x, posicao2.bottom);
        }
    } else if (centro2.x >= posicao1.left && centro2.x <=
posicao1.right) {
        if (posicao1 == posicaoAcima) {
            Log.d(TAG, "onDraw: origem=(" + centro2.x + ",
" + posicao1.bottom + ")");
            Log.d(TAG, "onDraw: destino=(" + centro2.x + ",
" + posicao2.top + ")");

            origem.set(centro2.x, posicao1.bottom);

```

```

        destino.set(centro2.x, posicao2.top);
    } else {
        Log.d(TAG, "onDraw: origem=(" + centro2.x + ",
" + posicao1.top + ")");
        Log.d(TAG, "onDraw: destino=(" + centro2.x + ",
" + posicao2.bottom + ")");

        origem.set(centro2.x, posicao1.top);
        destino.set(centro2.x, posicao2.bottom);
    }
    } else {
        if (posicao1 == posicaoAcima) {
            if (posicao1 == posicaoEsq) {
                Log.d(TAG, "onDraw: origem=(" +
posicao1.right + ", " + centro1.y + ")");

                origem.set(posicao1.right, centro1.y);
            } else {
                Log.d(TAG, "onDraw: origem=(" +
posicao1.left + ", " + centro1.y + ")");

                origem.set(posicao1.left, centro1.y);
            }

            Log.d(TAG, "onDraw: destino=(" + centro2.x + ",
" + posicao2.top + ")");

            destino.set(centro2.x, posicao2.top);
        } else {
            Log.d(TAG, "onDraw: origem=(" + centro1.x + ",
" + posicao1.top + ")");

            origem.set(centro1.x, posicao1.top);

            if (posicao2 == posicaoEsq) {
                Log.d(TAG, "onDraw: destino=(" +
posicao2.right + ", " + centro2.y + ")");

                destino.set(posicao2.right, centro2.y);
            } else {
                Log.d(TAG, "onDraw: destino=(" +
posicao2.left + ", " + centro2.y + ")");

                destino.set(posicao2.left, centro2.y);
            }
        }
    }
}

desenhar(canvas);

//      super.onDraw(canvas);
}

private void definirTamanho() {
    Log.d(TAG, "definirTamanho() called");

    ae.posicao(c1, posicao1);

```

```

        ae.posicao(c2, posicao2);

        centro1.set(posicao1.centerX(), posicao1.centerY());
        centro2.set(posicao2.centerX(), posicao2.centerY());

        origem();
        destino();

        Log.d(TAG, "definirTamanho: 1=" + posicao1);
        Log.d(TAG, "definirTamanho: 2=" + posicao2);

        Log.d(TAG, "definirTamanho: esq=" + posicaoEsq);
        Log.d(TAG, "definirTamanho: dir=" + posicaoDir);
        Log.d(TAG, "definirTamanho: cima=" + posicaoAcima);
        Log.d(TAG, "definirTamanho: baixo=" + posicaoAbaixo);

        limites.set(Math.min(posicaoEsq.left,      posicaoDir.right),
Math.min(posicaoAcima.top, posicaoAbaixo.bottom),
                Math.max(posicaoEsq.left,      posicaoDir.right),
Math.max(posicaoAcima.top, posicaoAbaixo.bottom));

        Log.d(TAG, "definirTamanho: l=" + limites.width() + ", a="
+ limites.height());

        Log.d(TAG,      "definirTamanho()      returned:      limites="      +
limites);
    }

    @Override
    protected void onMeasure(int widthMeasureSpec, int
heightMeasureSpec) {
        Log.d(TAG, "onMeasure() called with: widthMeasureSpec = ["
+ widthMeasureSpec + "], heightMeasureSpec = [" + heightMeasureSpec
+ "]");

        definirTamanho();

        setMeasuredDimension(Math.abs(limites.width()),
Math.abs(limites.height()));

        Log.d(TAG, "onMeasure() returned: limites=" + limites);
    }

    @Override
    protected void onSizeChanged(int w, int h, int oldw, int oldh)
    {
        Log.d(TAG, "onSizeChanged() called with: w = [" + w + "], h
= [" + h + "], oldw = [" + oldw + "], oldh = [" + oldh + "]");

        definirTamanho();

        super.onSizeChanged(w, h, oldw, oldh);
    }
}

```

8.4 Diretório: *app/src/main/res/layout*

8.4.1 **fragment_edicao_atributo.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <EditText
        android:id="@+id/et_nome_atributo"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="atributo1"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <View
        android:id="@+id/separador1"
        style="@style/HDivider"
        android:layout_marginTop="8dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/et_nome_atributo" />

    <TextView
        android:id="@+id/tv_visibilidade"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/visibilidade"
        app:layout_constraintTop_toBottomOf="@+id/separador1"
        android:layout_marginLeft="8dp"
        app:layout_constraintLeft_toLeftOf="parent"
        android:layout_marginTop="8dp" />

    <HorizontalScrollView
        android:id="@+id/hsv_visibilidades"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginTop="8dp"
```

```

        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/tv_visibilidade">

    <RadioGroup
        android:id="@+id/rg_visibilidades"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="0dp"
        android:layout_marginLeft="0dp"
        android:layout_marginRight="0dp"
        android:layout_marginStart="0dp"
        android:layout_marginTop="0dp"
        android:checkedButton="@+id/rb_privado"
        android:orientation="horizontal"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/tv_visibilidade">

        <RadioButton
            android:id="@+id/rb_privado"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/privado" />

        <RadioButton
            android:id="@+id/rb_protegido"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/protegido" />

        <RadioButton
            android:id="@+id/rb_pacote"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/pacote" />

        <RadioButton
            android:id="@+id/rb_publico"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/publico" />

    </RadioGroup>
</HorizontalScrollView>

<View
    android:id="@+id/separador2"
    style="@style/HDivider"

```

```

        android:layout_marginTop="8dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/hsv_visibilidades" />

<TextView
    android:id="@+id/tv_tipo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="@string/tipo"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/separador2" />

<Spinner
    android:id="@+id/s_tipos"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:entries="@array/tipos_exemplo"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tv_tipo" />

<View
    android:id="@+id/separador3"
    style="@style/HDivider"
    android:layout_marginTop="8dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/s_tipos" />

<TextView
    android:id="@+id/tv_valor_inicial"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="@string/valor_inicial"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/separador3" />

<EditText
    android:id="@+id/et_valor_inicial"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"

```

```

        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="42"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/tv_valor_inicial" />

</android.support.constraint.ConstraintLayout>

```

8.4.2 fragment_edicao_classe.xml

```

<?xml version="1.0" encoding="utf-8"?>

<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/edicaoClasse"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.constraint.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:layout_editor_absoluteX="0dp"
        tools:layout_editor_absoluteY="0dp">

        <EditText
            android:id="@+id/et_nome_classe"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginEnd="8dp"
            android:layout_marginLeft="8dp"
            android:layout_marginRight="8dp"
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"
            android:ems="10"
            android:inputType="textPersonName"
            android:text="Classe 1"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <View
            android:id="@+id/separador1"
            style="@style/HDivider"
            android:layout_width="0dp"
            android:layout_marginTop="8dp"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"

```

```

app:layout_constraintTop_toBottomOf="@+id/et_nome_classe"
    android:layout_height="0dp" />

<TextView
    android:id="@+id/tv_categoria"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:text="@string/tv_categoria"
    app:layout_constraintLeft_toLeftOf="parent"

app:layout_constraintTop_toBottomOf="@+id/separador1" />

<HorizontalScrollView
    android:id="@+id/hsv_categorias"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/tv_categoria">

<RadioGroup
    android:id="@+id/rg_categorias"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:checkedButton="@+id/rb_concreta"
    android:orientation="horizontal"
    tools:layout_editor_absoluteX="111dp"
    tools:layout_editor_absoluteY="193dp">

    <RadioButton
        android:id="@+id/rb_concreta"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/concreta" />

    <RadioButton
        android:id="@+id/rb_abstrata"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/botao_abstrata" />

    <RadioButton
        android:id="@+id/rb_interface"

```

```

        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/botao_interface" />

        <RadioButton
            android:id="@+id/rb_enumeracao"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/enumeracao" />

    </RadioGroup>
</HorizontalScrollView>

<View
    android:id="@+id/separador2"
    style="@style/HDivider"
    android:layout_width="0dp"
    android:layout_marginTop="8dp"
    android:layout_weight="1"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/hsv_categorias"
    android:layout_height="0dp" />

<RelativeLayout
    android:id="@+id/rl_atributos_e_operacoes"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginBottom="8dp"
    android:layout_marginTop="8dp"
    android:orientation="vertical"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/separador2">

    <View
        android:id="@+id/separador3"
        style="@style/HDivider"
        android:layout_centerVertical="true"
        android:visibility="visible" />

    <TextView
        android:id="@+id/tv_atributos"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginBottom="0dp"
        android:layout_marginLeft="8dp"

```

```

        android:layout_marginTop="0dp"
        android:text="@string/atributos" />

<TextView
    android:id="@+id/tv_operacoes"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/separador3"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:text="@string/operacoes" />

<ListView
    android:id="@+id/lv_atributos"
    style="@style/Widget.AppCompat.ListView"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_above="@+id/separador3"
    android:layout_below="@+id/tv_atributos"
    android:layout_marginBottom="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginTop="8dp"
    android:footerDividersEnabled="false"
    android:headerDividersEnabled="false" />

<ListView
    android:id="@+id/lv_operacoes"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_below="@+id/tv_operacoes"
    android:layout_marginBottom="0dp"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginTop="8dp" />

</RelativeLayout>

</android.support.constraint.ConstraintLayout>

</FrameLayout>

```

8.4.3 fragment_edicao_operacoes.xml

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <EditText
        android:id="@+id/et_nome_operacao"

```

```

        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        android:ems="10"
        android:inputType="textPersonName"
        android:text="mÃ©todo1"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

<View
    android:id="@+id/separador1"
    style="@style/HDivider"
    android:layout_marginTop="8dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/et_nome_operacao" />

<TextView
    android:id="@+id/tv_visibilidade"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginStart="8dp"
    android:text="@string/visibilidade"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/separador1"
    android:layout_marginTop="8dp" />

<HorizontalScrollView
    android:id="@+id/hsv_visibilidades"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginTop="8dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/tv_visibilidade"
    android:layout_marginStart="8dp"
    android:layout_marginEnd="8dp">

<RadioGroup
    android:id="@+id/rg_visibilidades"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="0dp"
    android:layout_marginLeft="0dp"
    android:layout_marginRight="0dp"

```

```

        android:layout_marginStart="0dp"
        android:layout_marginTop="0dp"
        android:checkedButton="@+id/rb_publico"
        android:orientation="horizontal"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/tv_visibilidade">

    <RadioButton
        android:id="@+id/rb_publico"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/publico" />

    <RadioButton
        android:id="@+id/rb_protegido"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/protegido" />

    <RadioButton
        android:id="@+id/rb_pacote"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/pacote" />

    <RadioButton
        android:id="@+id/rb_privado"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/privado" />

    </RadioGroup>
</HorizontalScrollView>

<View
    android:id="@+id/separador2"
    style="@style/HDivider"
    android:layout_marginTop="8dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/hsv_visibilidades" />

<TextView
    android:id="@+id/tv_tipo"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginStart="8dp"

```

```

        android:layout_marginTop="8dp"
        android:text="@string/tipo"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/separador2" />

<Spinner
    android:id="@+id/s_tipos"
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    android:entries="@array/tipos_exemplo"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/tv_tipo" />

<View
    android:id="@+id/separador3"
    style="@style/HDivider"
    android:layout_marginTop="8dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/s_tipos" />

<TextView
    android:id="@+id/tv_parametros"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginTop="8dp"
    android:text="@string/parametros"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/separador3"
    tools:layout_editor_absoluteY="207dp" />

<RelativeLayout
    android:id="@+id/rl_parametros"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginBottom="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    android:layout_marginTop="8dp"

app:layout_constraintTop_toBottomOf="@+id/tv_parametros">

    <android.support.constraint.ConstraintLayout
        android:id="@+id/cl_parametros"
        android:layout_width="match_parent"

```

```

        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentRight="true"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        android:layout_marginBottom="4dp"

app:layout_constraintBottom_toTopOf="@+id/cl_adicionar_parametro
"

        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent">

<ListView
    android:id="@+id/lv_parametros"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginBottom="0dp"
    android:layout_marginRight="8dp"
    android:paddingBottom="8dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toLeftOf="parent"

app:layout_constraintRight_toLeftOf="@+id/ll_editar_e_excluir"
    app:layout_constraintTop_toTopOf="parent" />

<LinearLayout
    android:id="@+id/ll_editar_e_excluir"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:orientation="vertical"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent">

    <Button
        android:id="@+id/b_editar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="0dp"
        android:layout_marginRight="0dp"
        android:layout_marginTop="0dp"
        android:layout_weight="1"
        android:text="@string/editar"

app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/separador3" />

    <Button
        android:id="@+id/b_excluir"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/excluir"

app:layout_constraintRight_toRightOf="parent"

```

```

app:layout_constraintTop_toBottomOf="@+id/b_editar" />

</LinearLayout>

</android.support.constraint.ConstraintLayout>

<android.support.constraint.ConstraintLayout
    android:id="@+id/cl_adicionar_parametro"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentEnd="true"
    android:layout_alignParentLeft="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/cl_parametros"
    android:layout_marginTop="4dp"
    android:layout_toStartOf="@+id/cl_parametros"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/cl_parametros">

    <RelativeLayout
        android:id="@+id/rl_adicionar_parametro"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginLeft="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginStart="8dp"

app:layout_constraintBaseline_toBaselineOf="@+id/b_adicionar"
    app:layout_constraintLeft_toLeftOf="parent"

app:layout_constraintRight_toLeftOf="@+id/b_adicionar">

    <View
        android:id="@+id/v_centro"
        android:layout_width="0dp"
        android:layout_height="0dp"
        android:layout_centerInParent="true"
        android:visibility="invisible" />

    <EditText
        android:id="@+id/et_nome_parametro_novo"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_alignEnd="@+id/v_centro"
        android:layout_alignParentLeft="true"
        android:layout_marginRight="4dp"
        android:ems="10"
        android:inputType="textPersonName" />

```

```

        <EditText
            android:id="@+id/et_tipo_parametro_novo"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_alignParentRight="true"
            android:layout_alignStart="@+id/v_centro"
            android:layout_marginLeft="4dp"
            android:ems="10"
            android:inputType="textPersonName" />
    </RelativeLayout>

    <Button
        android:id="@+id/b_adicionar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentRight="true"
        android:layout_weight="2"
        android:text="@string/adicionar"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
</RelativeLayout>

</android.support.constraint.ConstraintLayout>

```

8.4.4 fragment_edicao_relacao.xml

```

<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <android.support.constraint.ConstraintLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <EditText
            android:id="@+id/et_nome_relacao"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_marginEnd="8dp"
            android:layout_marginLeft="8dp"
            android:layout_marginRight="8dp"
            android:layout_marginStart="8dp"
            android:layout_marginTop="8dp"
            android:ems="10"
            android:inputType="textPersonName"
            android:text="RelaÃ§Ã£o 1"
            app:layout_constraintLeft_toLeftOf="parent"

```

```

        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

<View
    android:id="@+id/separador1"
    style="@style/HDivider"
    android:layout_width="0dp"
    android:layout_marginTop="8dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/et_nome_relacao"
    android:layout_marginLeft="0dp" />

<View
    android:id="@+id/separador2"
    style="@style/HDivider"
    android:layout_width="0dp"
    android:layout_marginLeft="0dp"
    android:layout_marginRight="0dp"
    android:layout_marginTop="8dp"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/radioGroup" />

<RadioGroup
    android:layout_width="0dp"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginTop="8dp"
    android:orientation="horizontal"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/separador1"
    android:id="@+id/radioGroup"
    android:layout_marginStart="8dp"
    android:layout_marginEnd="8dp">

    <RadioButton
        android:id="@+id/rb_unidirecional"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/unidirecional" />

    <RadioButton
        android:id="@+id/rb_bidirecional"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/bidirecional"
        android:textAlignment="viewStart" />

```

```

</RadioGroup>

<android.support.constraint.ConstraintLayout
    android:id="@+id/constraintLayout"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="8dp"
    app:layout_constraintLeft_toLeftOf="parent"

app:layout_constraintTop_toBottomOf="@+id/separador2">

    <TextView
        android:id="@+id/tv_origem"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="8dp"
        android:layout_marginStart="8dp"

android:background="@drawable/representacao_classe_para_edicao_r
elacao"

        android:text="Classe 1"
        android:textColor="@android:color/black"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Spinner
        android:id="@+id/s_origem"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="8dp"
        android:layout_marginStart="8dp"
        android:entries="@array/multiplicidades_exemplo"
        android:textAlignment="center"

app:layout_constraintLeft_toRightOf="@+id/tv_origem"

app:layout_constraintBaseline_toBaselineOf="@+id/tv_origem" />

</android.support.constraint.ConstraintLayout>

<android.support.constraint.ConstraintLayout
    android:id="@+id/constraintLayout2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginEnd="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginTop="8dp"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/separador2">

```

```

        <TextView
            android:id="@+id/tv_destino"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginEnd="8dp"
            android:layout_marginRight="8dp"

android:background="@drawable/representacao_classe_para_edicao_r
elacao"

            android:text="Classe 2"
            android:textColor="@android:color/black"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent" />

        <Spinner
            android:id="@+id/s_destino"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginEnd="8dp"
            android:layout_marginRight="8dp"
            android:entries="@array/multiplicidades_exemplo"
            android:textAlignment="center"

app:layout_constraintRight_toLeftOf="@+id/tv_destino"

app:layout_constraintBaseline_toBaselineOf="@+id/tv_destino" />
        </android.support.constraint.ConstraintLayout>

    </android.support.constraint.ConstraintLayout>
</FrameLayout>

```

8.5 Diretório: *app/src/main/res/layout-land*

8.5.1 activity_agile_class.xml

```

<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"

tools:context="com.pamplona.rodolfo.agileclass.ui.AgileClass">

    <ScrollView
        android:id="@+id/scrollMenuV"
        android:layout_width="wrap_content"
        android:layout_height="0dp"
        android:layout_marginLeft="8dp"
        android:layout_marginStart="8dp"
        android:layout_marginTop="8dp"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintTop_toTopOf="parent">

        <LinearLayout

```

```

        android:id="@+id/listButtonsV"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        app:layout_constraintLeft_toLeftOf="@+id/parent"
        app:layout_constraintTop_toTopOf="parent">

        <Button
            android:id="@+id/button9"

style="@style/Widget.AppCompat.Button.Borderless"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/botao_classe"
            android:textSize="12sp" />

        <Button
            android:id="@+id/button12"

style="@style/Widget.AppCompat.Button.Borderless"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/botao_interface"
            android:textSize="12sp" />

        <Button
            android:id="@+id/button16"

style="@style/Widget.AppCompat.Button.Borderless"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/botao_abstrata"
            android:textSize="12sp" />

        <Button
            android:id="@+id/button10"

style="@style/Widget.AppCompat.Button.Borderless"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/botao_associacao"
            android:textSize="12sp" />

        <Button
            android:id="@+id/button11"

style="@style/Widget.AppCompat.Button.Borderless"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/botao_heranca"
            android:textSize="12sp" />

        <Button
            android:id="@+id/button13"

```

```

style="@style/Widget.AppCompat.Button.Borderless"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/botao_realizacao"
    android:textSize="12sp" />

    <Button
        android:id="@+id/button14"

style="@style/Widget.AppCompat.Button.Borderless"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/botao_agregacao"
    android:textSize="12sp" />

    <Button
        android:id="@+id/button15"

style="@style/Widget.AppCompat.Button.Borderless"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text="@string/botao_composicao"
    android:textSize="12sp" />

    </LinearLayout>
</ScrollView>

<View
    android:id="@+id/separadorV"
    style="@style/VDivider"
    android:layout_marginLeft="8dp"
    android:layout_marginStart="8dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toRightOf="@+id/scrollMenuV"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.0" />

<com.pamplona.rodolfo.agileclass.ui.AreaEdicao
    android:id="@+id/canvas"
    android:layout_width="0dp"
    android:layout_height="0dp"
    android:layout_marginBottom="0dp"
    android:layout_marginLeft="0dp"
    android:layout_marginRight="0dp"
    android:layout_marginTop="0dp"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintLeft_toRightOf="@+id/separadorV"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.0" />

</android.support.constraint.ConstraintLayout>

```

8.6 Diretório: *app/src/main/res/layout-port*

8.6.1 activity_agile_class.xml

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/fl_raiz"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <android.support.constraint.ConstraintLayout
        android:id="@+id/cl_tela_inicial"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:layout_editor_absoluteX="8dp"
        tools:layout_editor_absoluteY="8dp">

        <HorizontalScrollView
            android:id="@+id/sv_menu_horizontal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="0dp"
            android:layout_marginRight="0dp"
            android:layout_marginStart="0dp"
            android:layout_marginTop="0dp"
            android:layout_weight="1.58"
            android:clipToPadding="false"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent"
            app:layout_constraintTop_toTopOf="parent">

            <LinearLayout
                android:id="@+id/ll_lista_botoes_horizontal"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:orientation="horizontal"
                app:layout_constraintLeft_toLeftOf="parent"
                app:layout_constraintRight_toRightOf="parent"
                app:layout_constraintTop_toTopOf="parent">

                <Button
                    android:id="@+id/button"

                    style="@style/Widget.AppCompat.Button.Borderless"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:layout_weight="1"
                    android:text="@string/botao_classe"
                    android:textSize="12sp" />

                <Button
```

```

        android:id="@+id/button7"

style="@style/Widget.AppCompat.Button.Borderless"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/botao_interface"
        android:textSize="12sp" />

<Button
        android:id="@+id/button8"

style="@style/Widget.AppCompat.Button.Borderless"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/botao_abstrata"
        android:textSize="12sp" />

<Button
        android:id="@+id/button2"

style="@style/Widget.AppCompat.Button.Borderless"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/botao_associacao"
        android:textSize="12sp" />

<Button
        android:id="@+id/button3"

style="@style/Widget.AppCompat.Button.Borderless"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/botao_heranca"
        android:textSize="12sp" />

<Button
        android:id="@+id/button4"

style="@style/Widget.AppCompat.Button.Borderless"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1"
        android:text="@string/botao_realizacao"
        android:textSize="12sp" />

<Button
        android:id="@+id/button5"

style="@style/Widget.AppCompat.Button.Borderless"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"

```

```

        android:layout_weight="1"
        android:text="@string/botao_agregacao"
        android:textSize="12sp" />

        <Button
            android:id="@+id/button6"

style="@style/Widget.AppCompat.Button.Borderless"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="@string/botao_composicao"
            android:textSize="12sp" />
    </LinearLayout>
</HorizontalScrollView>

<View
    android:id="@+id/separador_horizontal1"
    style="@style/HDivider"
    android:layout_marginTop="0dp"
    android:visibility="visible"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/sv_menu_horizontal" />

    <com.pamplona.rodolfo.agileclass.ui.AreaEdicao
        android:id="@+id/canvas"
        android:layout_width="0dp"
        android:layout_height="0dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"

app:layout_constraintTop_toBottomOf="@+id/separador_horizontal1"
        app:layout_constraintVertical_bias="0.0" />
    </android.support.constraint.ConstraintLayout>

</FrameLayout>

```

8.7 Diretório: *app/src/main/res/menu*

8.7.1 *app_menu.xml*

```

<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/mi_centralizar"
        android:title="@string/centralizar" />
</menu>

```

8.7.2 classe_menu_contexto.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:app="http://schemas.android.com/apk/res-auto"
      xmlns:android="http://schemas.android.com/apk/res/android">

    <item
        android:id="@+id/editarClasseItemMenu"
        android:title="Editar" />
    <item
        android:id="@+id/removerClasseItemMenu"
        android:title="Remover" />
</menu>
```

8.8 Diretório: app/src/main/res/values

8.8.1 attrs.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>

    <declare-styleable name="Classe">
        <attr name="tipo" format="enum">
            <enum name="concreta" value="0" />
            <enum name="abstrata" value="1" />
            <enum name="interface_" value="2" />
        </attr>
    </declare-styleable>

    <declare-styleable name="AreaEdicao" />
</resources>
```

8.8.2 strings.xml

```
<resources>
    <string name="app_name"
translatable="false">AgileClass</string>
    <string name="botao_classe">Classe</string>
    <string name="botao_associacao">AssociaÃ$Ãfo</string>
    <string name="botao_heranca">HeranÃ$a</string>
    <string name="botao_realizacao">RealizaÃ$Ãfo</string>
    <string name="botao_agregacao">AgregaÃ$Ãfo</string>
    <string name="botao_composicao">ComposiÃ$Ãfo</string>

    <string name="botao_interface">Interface</string>
    <string name="botao_abstrata">Abstrata</string>
    <string name="classe_tipo_interface">interface</string>
    <string name="classe_tipo_abstrata">abstrata</string>
    <string name="tv_categoria">Categoria:</string>
    <string name="enumeracao">EnumeraÃ$Ãfo</string>
    <string name="concreta">Concreta</string>
    <string name="atributos">Atributos:</string>
    <string name="operacoes">OperaÃ$Ãpes:</string>
    <string name="unidirecional">Unidirecional</string>
```

```

<string name="bidirecional">Bidirecional</string>
<string name="tipo">Tipo:</string>
<string name="valor_inicial">Valor Inicial:</string>
<string name="visibilidade">Visibilidade:</string>
<string name="privado">Privado</string>
<string name="publico">P blico</string>
<string name="pacote">Pacote</string>
<string name="protegido">Protegido</string>
<string name="editar">Editar</string>
<string name="excluir">Excluir</string>
<string name="adicionar">Adicionar</string>
<string name="parametros">Parametros:</string>
<string name="centralizar">Centralizar</string>
</resources>

```

8.8.3 styles.xml

```

<resources>

    <!-- Base application theme. -->
    <style name="AppTheme"
parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item
name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>

    <style name="HDivider">
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        <item name="android:layout_width">match_parent</item>
        <item name="android:layout_height">1dp</item>
        <item
name="android:background">?android:attr/listDivider</item>
    </style>

    <style name="VDivider">
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintBottom_toBottomOf="parent"
        <item name="android:layout_width">1dp</item>
        <item name="android:layout_height">match_parent</item>
        <item
name="android:background">?android:attr/listDivider</item>
    </style>

</resources>

```