Referencial Teórico OO

Breno H. de Lima Freitas, Guilherme Bertoluchi
Departamento de Ciência da Computação
Universidade Federal de Lavras (UFLA) – Lavras, MG – Brasil
brenobbg@computacao.ufla.br gbertoluchi@computacao.ufla.br

Resumo. *Este documento explica alguns conceitos de Programação Orientada a Objetos que tivemos conhecimento dentro deste semestre, e que usamos em nosso trabalho final, um aplicativo para Android. Sua função é adicionar filmes/livros que já foram assistidos/lidos e dar uma breve opinião de como este item o agradou, uma descrição, e seus dados técnicos.*

1. Objetivos e Funcionalidades

A tela inicial do aplicativo contém uma lista de filmes/livros ainda não assistidos/lidos, porém já cadastrados e ordenados pela prioridade escolhida pelo usuário no cadastro do item. Terá também 3 opções de botões, um para cadastrar livro, outro para filme e por fim, ver todos os itens cadastrados.

Ao clicar em cadastrar filme, o usuário é direcionado para uma tela que contém diversos campos de texto a serem preenchidos. Há uma opção de consumido, onde se marcada irá mostrar a opção de avaliação do item a ser cadastrado. A tela livro tem a mesma ideia, porém existe algumas características diferentes, que são únicas para livro. Ao finalizar a entrada de dados, existirá um botão cadastrar, assim, este item será incluído na lista de item cadastrados.

A Tela que mostra todos os itens cadastrados tem a funcionalidade de mostrar os nomes dos itens, porém quando tocamos neles, um diálogo é aberto, contendo todos os detalhes do item clicado, juntamente com o ícone correspondente ao item, situado no título do diálogo. Na parte inferior, há um botão para remover o item, e outro botão para alterar o status de consumido, para “Sim” ou “Não”. Quando o usuário faz essa alteração, a lista da tela principal se adapta, escondendo ou mostrando os respectivos itens, de acordo com o novo valor desse atributo.

A versão do Android mínima necessária para rodar o aplicativo é a 4.0.3, popularmente conhecida como “Ice Cream Sandwich”, de API Level 15.

2. Referencial Teórico

Em Orientação a Objetos, existem vários conceitos que facilitam o trabalho do programador, nos quais os que temos que ter uma base são: Herança, Polimorfismo, Encapsulamento, Associação, Agregação, Composição, Classes Abstratas, Interface e por ultimo não menos importante Tratamento de Erros.

A primeira noção básica que temos que ter seria o Encapsulamento, pois é onde existe a menor célula de orientação a objeto, que no caso é a classe. Nela existe atributos e métodos, nestes podemos utilizar a visibilidade, que no caso, se for público, qualquer outra classe que tiver um construtor desta, poderá alterar o valor desta variável. Assim, adotamos um padrão para atributos que seria, torná-los todos privados, e para outra classe conseguir acessar, teríamos que fazer métodos públicos que faziam a ligação com os atributos da classe, que são os famosos Getters e Setters.

Com o aumento de códigos repetidos, onde criamos várias classes com quase todo comportamento igual, e temos que ficar duplicando os códigos, mudando poucas coisas, surgiu então a Herança, que conceitualmente é feita de superClasses e subClasses. Onde todos os atributos da superClasse, são passados para a subClasse. Um bom exemplo seria a superClasse Animal, que tem como atributos, nroDePatas, raca, e nome, e a subClasse Cachorro, que então herdou todos esses atributos ficando com, nroDePatas, raça, nome, corDosPelos, cuja este é atributo da própria classe.

Neste contexto, existe também um outro conceito de OO, que é um dos mais importantes, cuja se intitula Polimorfismo. Nele podemos utilizar as variáveis polimórficas. Essas variáveis são quando usamos do conceito que um cachorro é sempre um animal. Caso tivermos uma outra classe chamada Gato, cuja também herdaria de Animal, poderíamos guardar variáveis em uma lista, do tipo Animal, pois as duas outras classes são Animais.

Ao pensarmos que as classes se relacionam, criando um objeto de outra, ou até mesmo um List, podemos definir tipos de relacionamento, onde um exemplo é a classe Carro, que tem um relacionamento com a classe Motor. Lê-se Um carro tem um Motor, onde o motor é único e exclusivo de carro, e o mais importante é que quando o Carro é destruído, seguidamente o Motor também é, pois não faria sentido existir um Motor sem um Carro. Essa relação é de Composição, onde o Todo/parte determina o tempo de vida do outro.

Uma relação com as mesmas características porém o Todo não determina tempo de vida da outra classe, se chama Agregação. Exemplo seria o Estoque, e os Produtos. Os produtos pertenceriam ao Estoque, porém eles poderiam ser enviados para outra cidade, e assim não teriam mais relação com esse Estoque, mas os dois continuariam existindo.

Existe também um outro tipo de relação, onde podemos pensar em uma relação de que nenhum Todo tem uma parte, apenas se relacionam, e nenhum controla o tempo de vida do outro. Um bom exemplo seria do professor e aluno, os dois se relacionam, porém não faz sentido falar que o professor é parte do aluno, ou aluno é composto por um professor, apenas se relacionam com as aulas.

Na Orientação a Objetos, podemos sobrescrever métodos, assim quando herdamos de uma superClasse tal método, podemos utilizar este com a implementação que quisermos em nossa classe atual, partindo deste conceito temos os métodos abstratos, que são feitos na superClasse apenas com a chamada deles, onde foram feitos para serem sobrescritos, porém se uma classe tiver um método abstrato, ela obrigatoriamente teria que ser abstrata, cuja não pode haver construtor dessa classe. Pode existir classe abstrata sem métodos abstratos, o que não pode é ao contrário.

Para não precisar criar uma classe abstrata toda vez que precisamos utilizar um método abstrato, tem-se a Interface, onde todos seus métodos são abstratos da mesma forma que os da classe abstrata, porém também são públicos e estáticos. Uma das vantagens de utilizar a interface é que podemos ter herança múltipla, onde podemos herdar de mais de uma interface. Utilizamos este conceito quando queremos “assinar um contrato”, onde as nossas classes precisariam utilizar esses métodos, e o mais importante, onde classes futuras utilizariam destes também, sem precisar mexer no código.

Na programação existe diversos tipos de erros, onde uns são tratados e outros não, e existem erros também que são da própria JVM(*java virtual machine*) cuja pode acontecer de acessar uma memória utilizada por outro programa, entre outras coisas, estas, não são nem lançadas exceções. O lançamento de exceções é separado em dois tópicos, um tratadas, cuja o próprio compilador obriga o programador a tratar, que no caso pode ser quando tentamos dar a entrada de dados a partir de um arquivo. Já as exceções não tratadas são por exemplo uma divisão por 0.

Agora, em questão de melhoramento do código, o design pattern é de uma grande ajuda, pois, os conceitos que o englobam, facilitam a vida do programador e dos outros programadores que pegariam o código posteriormente. Os detalhes de maior importância são : ter baixo acoplamento; ter uma alta coesão dos métodos/classes; separar interface de usuário, regra de negocio e acesso a dados; evitar duplicação de códigos.

3. Aplicação dos conceitos de OO

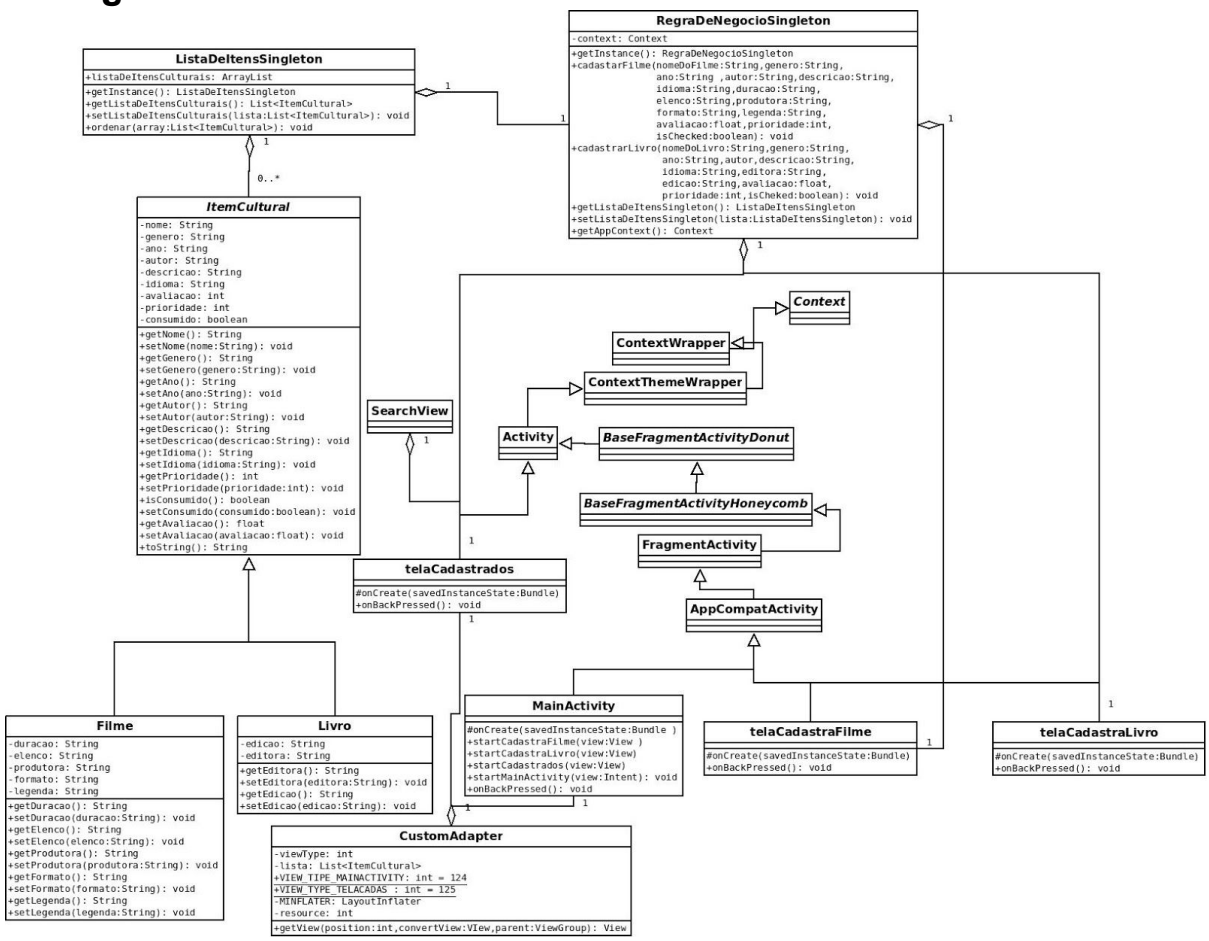
Em nosso trabalho utilizamos a Herança principalmente nas classes ItemCultural, Filme, Livro, onde ItemCultural seria a superClasse das outras duas, e caso quiséssemos colocar mais um item cultural, por exemplo série, não teríamos que mexer em nada, apenas fazer uma outra classe, e uma nova tela, que no caso seria a interface de usuário.

Já o polimorfismo é quando tratamos de um objeto de Filme/Livro como ItemCultural, onde existe uma lista de Itens Culturais que guardam este mesmo.

A utilização da agregação foi feita a a partir do relacionamento que a interface de usuário tem com a regra de negocio, onde no trabalho foi feito essa relação de todas as classes da interface com a RegraDeNegocio, que é onde comandamos todo o processo.

Um outro conceito no qual faz uso no projeto, seria o Design pattern, cuja foi aplicado na classe RegraDeNegocio, só é necessário instanciar ela uma única vez, assim utilizamos o Singleton, no qual o algoritmo faz uma lógica de sempre pegar a mesma instância.

4. Diagrama UML



5. Instruções para compilar e executar o protótipo:

Para compilar o projeto, deve-se ter o Android Studio devidamente instalado namáquina, juntamente com a SDK. No nosso caso, usamos a versão 1.5.1. Deve-se colocar a pasta do projeto dentro do seu diretório /AndroidStudioProjects. Também possível visualizar o projeto através do GitHub.

Para executar o protótipo diretamente em um dispositivo, transfira o arquivo seenme.apk, na pasta APK do projeto, para o seu device, faça a instalação e abra o aplicativo recém instalado.

O JavaDoc está na pasta do projeto, e pode ser através do index.hml no seu navegador. O UML em maior resolução também se encontra na pasta do projeto, no diretório UML.

6. Referência Bibliográfica

TERRA, Ricardo – **Linguagem Java.** [Projeção Visual]. Acessível em : http://pt.slideshare.net/rterrabh/apostila-java-completa.

ALVES, Júlio César – **GCC110 Programação Orientada a Objetos : Herança e Polimorfismo.** [Projeção Visual]. Acessível em http://alunos.dcc.ufla.br/.

ALVES, Júlio César – **GCC110 Programação Orientada a Objetos : Comp. - Agreg. - Associação**. [Projeção Visual]. Acessível em http://alunos.dcc.ufla.br/.

ALVES, Júlio César – **GCC110 Programação Orientada a Objetos : Encapsulamento Visibilidade**. [Projeção Visual]. Acessível em http://alunos.dcc.ufla.br/.

ALVES, Júlio César – **GCC110 Programação Orientada a Objetos : Tratamento de Exceções**. [Projeção Visual]. Acessível em http://alunos.dcc.ufla.br/.