Sistema de Locadora de Jogos Digitais

João Victor de Souza Calassio, 18/0033808
João Pedro Assunção Coutinho, 18/0019813

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CIC 117889 - Tecnicas de Programação 1

joao.victors1203@gmail.com, jpedro0801@gmail.com

Abstract. *This report covers the implementation of a renting games system, in Java programming language.*

Resumo. *Esse relatório trata da implementação de um sistema de locação de jogos, em linguagem de programação Java.*

1. Introdução

Neste trabalho, o grupo teve como objetivo desenvolver um sistema de locadora de jogos, utilizando a orientação a objetos [4]. Inicialmente, foram desenvolvidos os modelos conceituais, com as descrições das classes de objeto, dos relacionamentos entre essas classes, das operações que podem ser realizadas no sistema, e a sequência dessas operações. Em seguida, utilizando dos modelos conceituais, o sistema foi implementado através da linguagem de programação Java. Além do modelo conceitual para o sistema principal, que foi implementado, também foram desenvolvidos modelos conceituais para as outras três propostas de sistemas.

2. Modelo conceitual

O modelo conceitual foi desenvolvido utilizando diagramas UML, construídos através do software Astah UML. Os diagramas desenvolvidos foram:

2.1. Diagrama de casos de uso

O diagrama de casos de uso tem como objetivo representar o comportamento externo do sistema, com suas funcionalidades, e com foco nos atores e nas ações que os atores podem realizar no sistema Como o sistema desenvolvido trata de locação de jogos, temos dois atores:

1. Cliente
2. Funcionário

Um cliente tem como ações alugar ou devolver jogos, e um funcionário tem como ações manter e gerenciar os aluguéis de jogos, conforme mostra a Figura 1

2.2. Diagrama de classes

O diagrama de classes descreve quais são conceitos e atributos referentes às classes do sistema, além dos relacionamentos entre as classes, como exibido na Figura 2. A classe básica para o aluguel de jogos é a classe *Locacao*, que controla as ações de alugar e devolver.

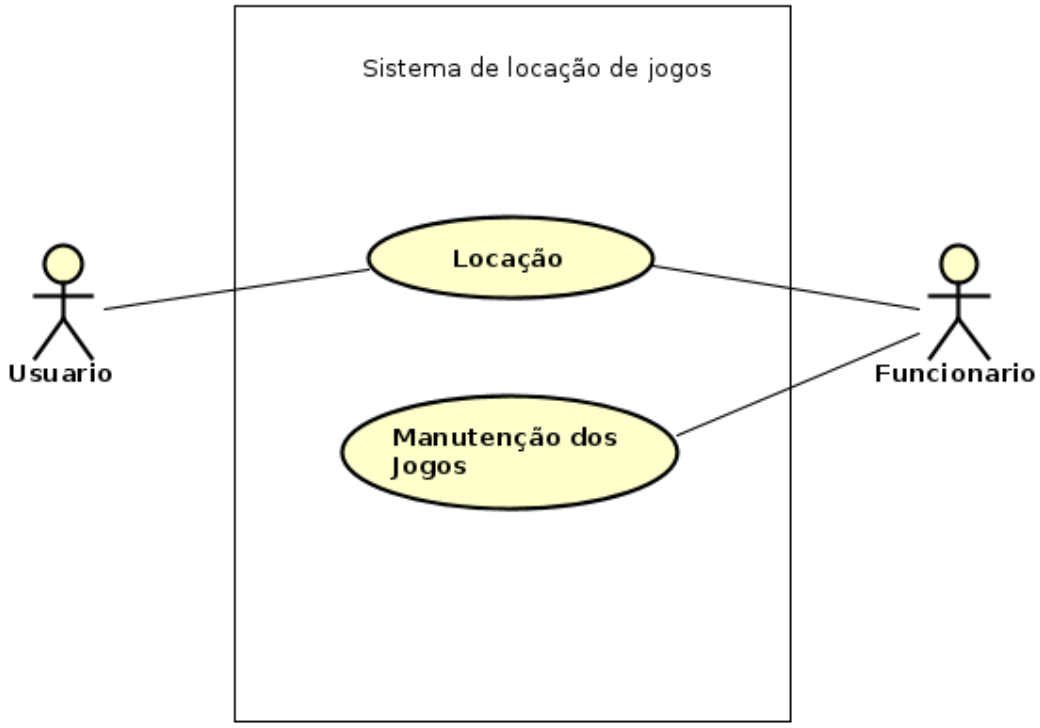

Figura 1. Diagrama de casos de uso utilizado

2.3. Diagrama de instâncias

O diagrama de instâncias representa um exemplo de funcionamento prático do diagrama de classes, ou seja, as instâncias dos objetos descritos no diagrama de classes, com seus relacionamentos, conforme mostra a Figura 3.

2.4. Diagrama sequencial

O diagrama sequencial mostra a comunicação entre os atores e o sistema, além de mostrar quais são os métodos que são necessários para realizar determinada ação. As ações para um determinado serviço, são representadas na ordem em que acontecem, com as mensagens transmitidas entre os elementos envolvidos. Os atores (mesmos do diagrama de casos de uso) solicitam serviços para o sistema, que inicia um processo representado no diagrama sequencial. As instâncias das classes são representadas através das *Lifelines*. O diagrama sequencial utilizado está exibido na Figura 4

3. Implementação

3.1. Package:locadora

É o principal pacote do sistema, contendo a classe ”Main”(classe em que se inicia a execução do programa), e as outras classes essenciais, sendo elas: ”Jogo”, ”Locacao”, ”Cliente”e ”Plataforma”.

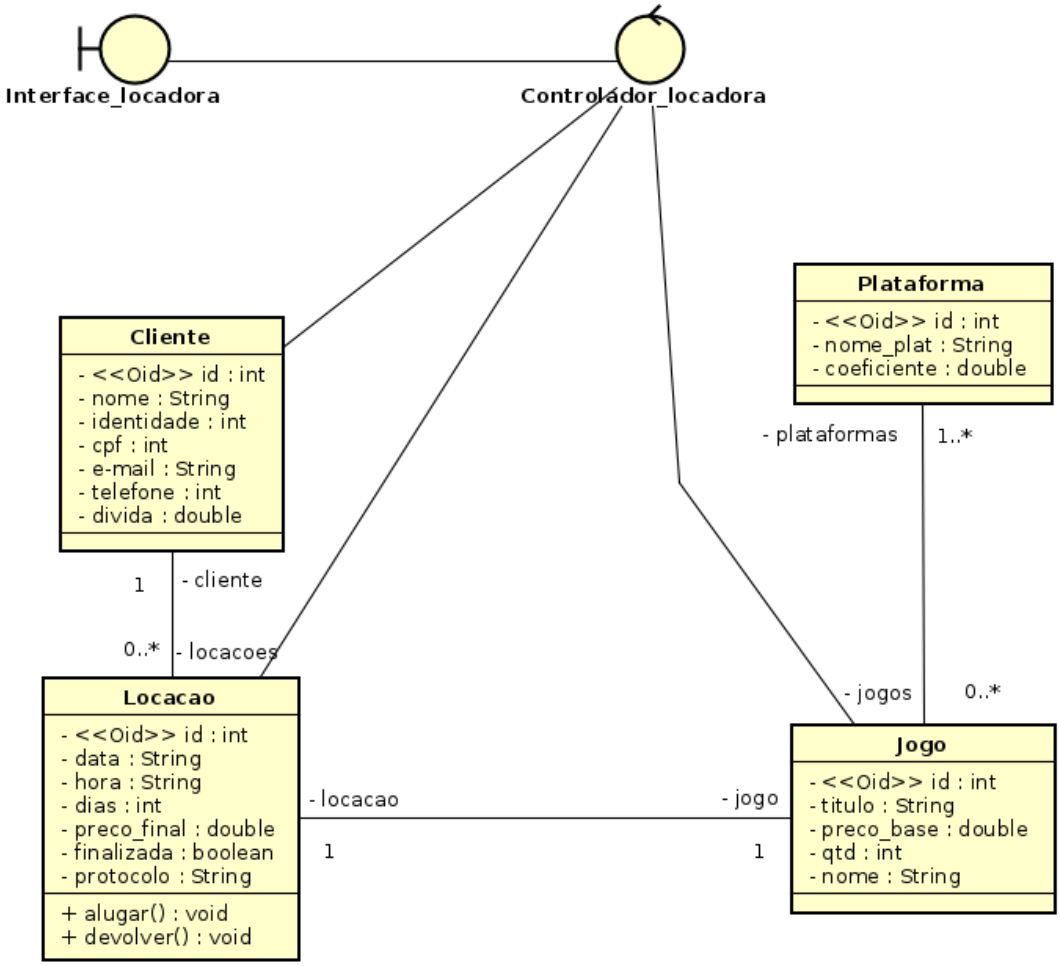

Figura 2. Diagrama de classes utilizado

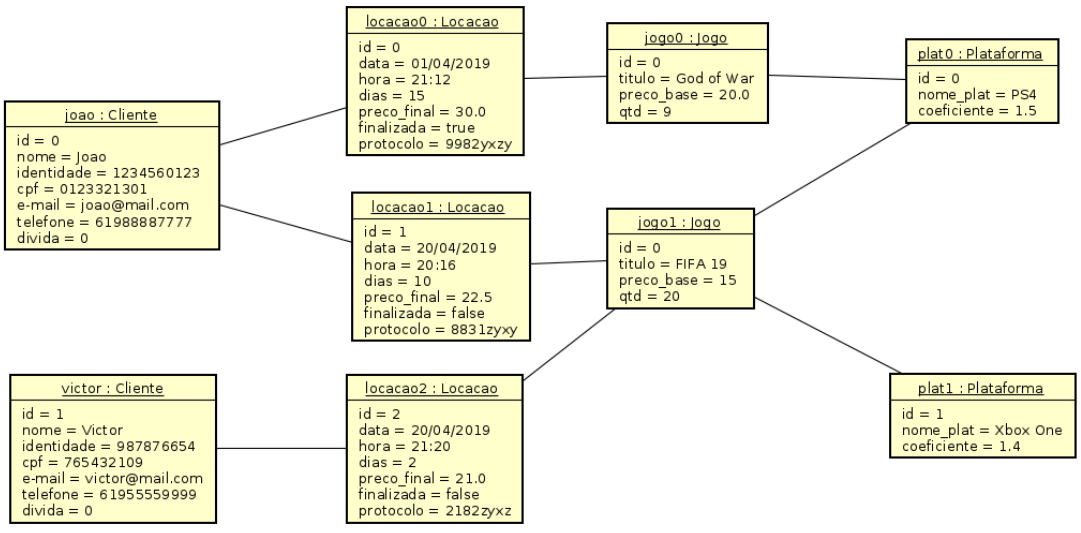

Figura 3. Diagrama de instâncias utilizado.

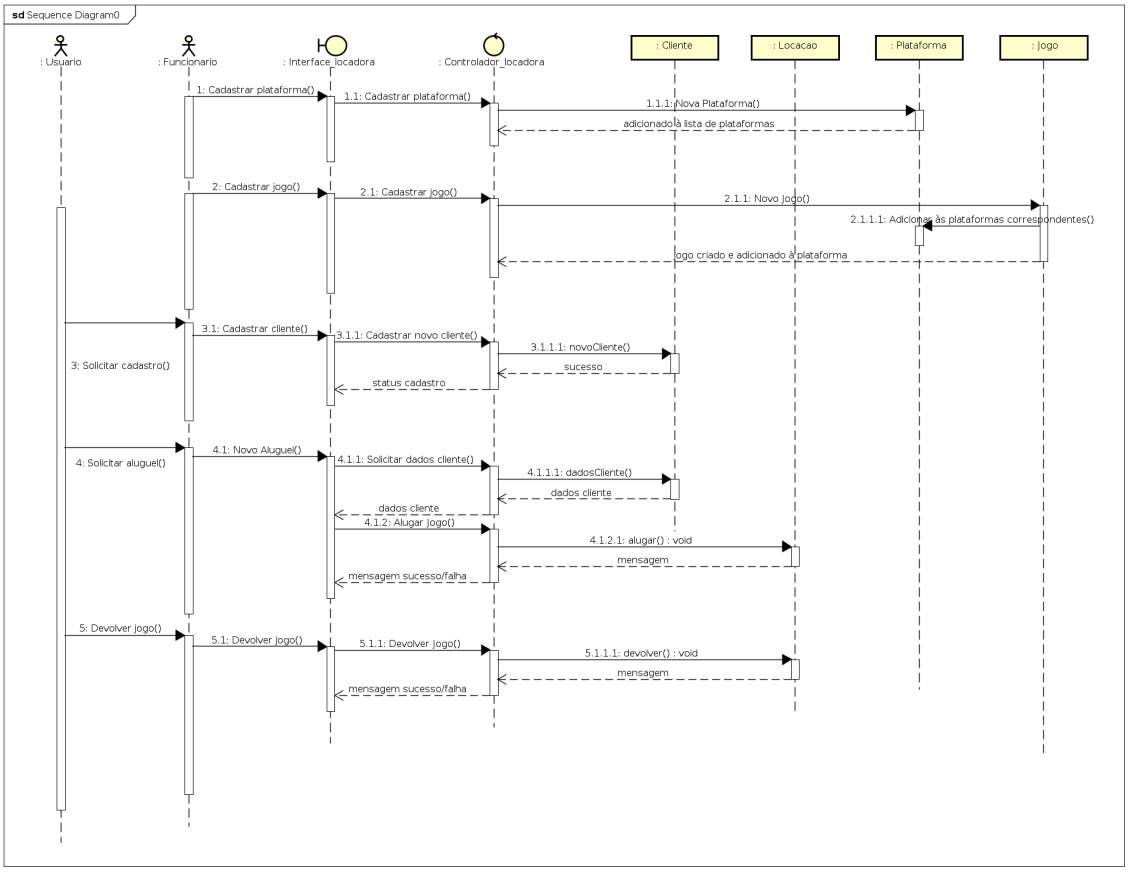

Figura 4. Diagrama sequencial utilizado

3.1.1. Class:Jogo

Contém os atributos que definem um jogo, como título, preço de aluguel(diário), quantidade desse mesmo jogo disponíveis e a Plataforma a qual esse jogo está associado. Contém ainda métodos modificadores de atributos, toString e métodos para diminuir ou aumentar a quantidade disponível de um jogo.

3.1.2. Class:Locacao

Contém os atributos que definem uma locação, como registros de datas, horas, preços, protocolos, o total de dias que se pretende ficar com o jogo, e um Jogo a qual essa locação está associada. Possui métodos modificadores de atributos e toString, além de um método para cálculo de preço final (utilizando, em geral, a fórmula :preço do jogo * coeficiente da plataforma * número de dias totais), além de métodos que determinam os comportamentos alugar e devolver, por exemplo. Nessa classe são utilizados imports do java.time, utilizados para recuperar o dia e data atual, e fazer a contagem de dias de forma mais precisa, a fim de acrescentar mais realismo ao sistema, tornado-o mais próximo de um sistema realmente utilizável.

3.1.3. Class:Cliente

Contém atributos que definem um cliente, como nome, RG, CPF, Email, telefone, e uma lista de locações associada à esse cliente. Possui ainda métodos modificadores de atributos, toString, além de um método para cálculo de dívida, que soma os preços finais de todas locações ainda não finalizadas(pagas); um método para adicionar uma nova locação à lista de locações do cliente, e um método para listar as locações ainda não finalizadas.

3.1.4. Class:Plataforma

Contém atributos que definem uma plataforma de jogos, como nome, coeficiente que multiplica o preço de um jogo, e uma lista de jogos associada. Possui além dos métodos que modificam variáveis privadas e do toString, um método que retorna a referência para a lista de jogos associada à plataforma em questão, um método para adicionar um jogo a essa lista, e um método para remover.

3.1.5. Class:Main

Contém o método main, por onde se inicia a execução do programa. Aqui se instanciam os objetos implementados no sistema, e chamam-se alguns de seus métodos, afim de testá-los e demonstrar sua funcionalidade.

3.2. Package:Test

Pacote onde se reúnem as classes de teste unitário, que testam métodos importantes de cada classe, utilizando a framework JUNIT 4 [1]. Não testamos métodos getters, setters

ou toString pois não julgamos necessário, dado a simplicidade da tarefa desempenhada por esses métodos. Possui as classes ”JogoTest”, ”LocacaoTest”, ”ClienteTest”, ”PlataformaTest”.

3.2.1. Class:JogoTest

Testa os métodos addQtd e subQtd, responsáveis por aumentar ou diminuir o número de um determinado jogo disponível.

3.2.2. Class:LocacaoTest

Testa o método que determina o preço final da locação em questão, além de testar os método alugar e devolver.

3.2.3. Class:ClienteTest

Testa o método responsável pelo cálculo da dívida total de um cliente, e o método reponsável por adicionar uma nova locacão à lista de locações associada ao cliente.

3.2.4. Class:PlataformaTest

Testa os métodos que adicionam ou removem jogos da lista de jogos associada à plataforma de jogos em questão.

3.2.5. Class:AlugarExTest

Testa a excecção AlugarEx, através de um teste que sempre deve lançar essa exceção.

3.2.6. Class:DevolverExTest

Testa a excecção DevolverEx, através de um teste que sempre deve lançar essa exceção.

3.2.7. Class:PrecoExTest

Testa a excecção PrecoEx, através de um teste que sempre deve lançar essa exceção.

3.3. Package:exception

Pacote onde se reúnem as classes para tratamento de excessões do sistema. Todas as classes são subclasses da superclasse java.lang.Exception, presente no Java.

3.3.1. Class:AlugarEx

Classe utilizada para tratar excessões referentes ao aluguel de jogos.

3.3.2. Class:DevolverEx

Classe utilizada para tratar excessões referentes à devolução de jogos.

3.3.3. Class:PrecoEx

Classe utilizada para tratar excessões referentes ao cálculo do preço final/atual de uma locação específica.

3.4. Package:interfaces

Pacote que contém interfaces de classe que são utilizadas no sistema.

3.5. Class:Aluguel

Inteface de classe que contém as assinaturas para os principais métodos para gerenciamento de uma locação. Esses métodos são:

- void : alugar()
- void : devolver()
- double : PrecoFinal()

Por ser uma inteface, todos os métodos são abstratos e são implementados pelas classes que implementam essa interface.

4. Conclusão

Conclui-se após a finalização do trabalho, que utilizar o modelo e seguir as etapas corretamente para o desenvolvimento de software, torna o trabalho mais intuitivo, possibilitando dividir melhor as tarefas e encontrar melhores soluções para os problemas, evitando maiores dificuldades durante a etapa de implementação e teste. As maiores dificuldades na realização do trabalho foram estabelecer relacionamentos, entre as entidades envolvidas, e isso já é extremamente facilitado com o uso dos diagramas e modelos conceituais.

Referências

[1] JUnit. JUnit - About. `https://junit.org/junit4/`. [Online; accessed April-2019].

[2] Oracle. Java 8 se documentation. `https://docs.oracle.com/javase/8/docs/`. [Online; accessed April-2019].

[3] E. Platform. Eclipse platform help system. `https://help.eclipse.org/2019-03/index.jsp`. [Online; accessed April-2019].

[4] M. Weisfield. *The Object-Oriented Thought Process*. Addison-Wesley - Pearson Education, 3rd edition, 2008.