# UNIVERSIDADE FEDERAL DE MINAS GERAIS

## PROGRAMAÇÃO MODULAR – 2017/01

Trabalho Prático 1 – Banco Imobiliário Modular

Marina Moreira - 2015004348

Ronald Pereira- 2015004437

## Introdução:

Banco Imobiliário é um jogo de tabuleiro lançado em 1944 pela Brinquedos Estrela. É o brinquedo mais bem-sucedido em vendas na história do Brasil, com mais de 30 milhões de unidades vendidas. O jogo consiste na compra e venda de propriedades como bairro, casas, hotéis, empresas, de forma que vença o jogador que não for à falência.

O objetivo do trabalho é implementar um jogo de banco imobiliário simples, com n jogadores e um tabuleiro com n casas. Um jogador sempre deve comprar um imóvel que ainda não tenha dono, caso possua dinheiro suficiente para isso, e só perde se cair em um imóvel alheio cujo aluguel é maior que seu saldo. Dada essas regras básicas, também é um objetivo do projeto utilizar dos conceitos de Programação Orientada a Objetos, tais como classes e objetos utilizando como ferramenta a linguagem de programação Java, na qual o problema foi abstraído e modelado, e, por fim, resolvido.

## Implementação:

O problema foi modularizado em 8 classes:

- BancoImobiliario: classe que contém o método main, usado para executar o código.
- Escritor: classe encarregada de gerar a saída do programa no formato esperado, utilizando métodos públicos de outras classes para acessar os dados do resultado do jogo.
- Imovel: representa os diferentes imóveis do jogo a partir dos atributos identificador, preço, aluguel e dono. Encarregada de inicializar um novo imóvel, calcular aluguel e devolver o imóvel para o banco.

- Jogador: representa cada jogador da partida a partir dos atributos identificador, saldo, aluguel pago, aluguel recebido, total de compras, voltas completas no tabuleiro, vezes passadas, posição no tabuleiro e se já perdeu ou não. Dessa classe espera-se que seja capaz de inicializar um novo jogador, calcular a próxima posição em que ele deve estar de acordo com o dado e posicioná-lo nesta, compre imoveis, pague e receba aluguel, retire um jogador do jogo de acordo com as regras. Essa classe representa o que serão os principais objetos do jogo e seus métodos são traduzidos em ações essenciais para que este ocorra, assim como para adquirir as estatísticas desejadas no resultado.
- Tabuleiro: representa, apesar do nome, apenas uma casa do tabuleiro. Como veremos mais adiante, um vetor de elementos do tipo Tabuleiro forma o tabuleiro do jogo, com os atributos posição e tipo da posição, significando respectivamente o número da casa e o que se encontra nela (qualquer imóvel, passe a vez ou start). O propósito desta classe é apenas criar e retornar uma posição no tabuleiro para que possa ser usada por outras classes e em outros métodos.
- Jogada: representa um evento do jogo que diz respeito a um jogador específico: a sua jogada. Suas informações são o identificador do jogador que está jogando, o identificador da própria jogada (necessário para as estatísticas desejadas) e quanto o jogador tirou no dado.
- Leitor: esta é a classe que lê a entrada e cria todos os objetos e dados que serão utilizados ao longo do jogo. Para isso, Leitor cria vetores de Tabuleiro, Imóvel, Jogador e Jogada, para tornar fácil o acesso posterior a esses objetos, já que toda a informação referente a uma partida é recebida de uma vez, e depois computada. Além disso, instancia um objeto da última classe mencionada, e executa um de seus métodos, passando os vetores e alguns outros dados como parâmetros.
- JogaJogo: classe filha de Jogada. Isso significa que JogaJogo possui os mesmos atributos e métodos da classe Jogada, e implementa alguns próprios. Como não há herança múltipla em java, não é possível que JogaJogo herde informação de mais de uma classe, mesmo que necessite de tais informações para operar, como é o caso. Por isso é necessário que Leitor passe seus dados como parâmetro. Escolhemos que a herança viesse de Jogada e não de Leitor porque as informações contidas no primeiro são mais latentes. Herdar de Leitor e passar jogadas como parâmetro exigiriam múltiplas chamadas ao método principal jogaJogo, chamado apenas uma vez pela classe Leitor, que de fato executa todos os passos do jogo descritos na entrada, gerando as estatísticas desejadas para a saída.

Os atributos de todas as classes foram declarados protected, por motivos de segurança, com uma exceção que é a classe Jogadas, cujos atributos são protected para que sua classe filha pudesse acessá-los. Por esse motivo, estão presentes vários métodos de retorno, porque apesar de os dados estarem satisfatoriamente modularizados, a execução do código exige um certo nível de colaboração entre elas. Os métodos de retorno permitem isso, uma vez que dão acesso de leitura de um atributo protected a uma classe alheia.

## Testes:

Decidimos rodar 3 casos distintos que testam o programa em pontos de maior possibilidade de erros

Teste 1:

Tabuleiro

tabuleiro.txt

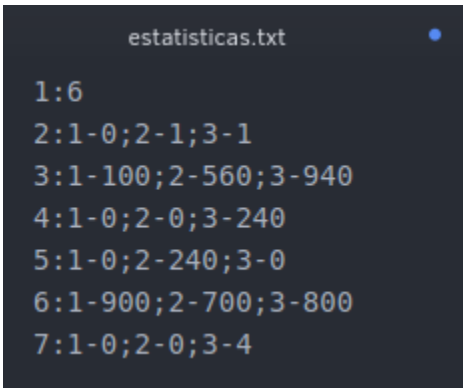
```
15
1;1;1;
2;2;3;5;300;80
3;3;3;5;300;80
4;4;3;5;300;80
5;5;2
6;6;3;1;150;70
7;7;3;1;150;70
8;8;2
9;9;3;2;200;50
10;10;3;2;200;50
11;11;2
12;12;3;4;250;60
13;13;3;3;500;90
14;14;2
15;15;3;3;500;90
```

Jogadas

jogadas.txt

```
19%3%1000
1;1;1
2;2;5
3;3;4
4;1;1
5;2;1
6;3;3
7;1;1
8;2;2
9;3;3
10;1;2
11;2;1
12;3;4
13;1;6
14;2;3
15;3;3
16;1;3
17;2;5
18;3;2
DUMP
```

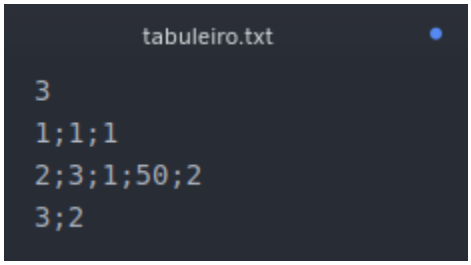
Saída



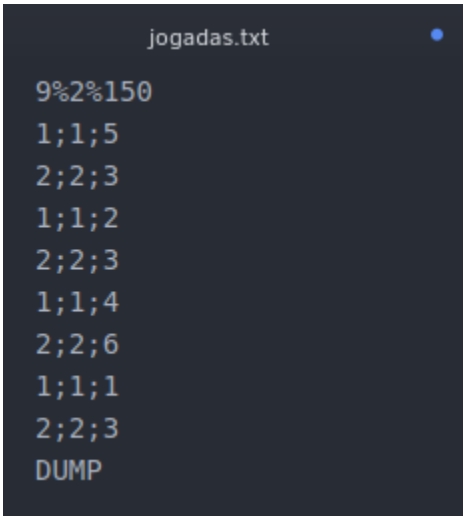

Esse teste representa o caso em que o jogo termina antes do final do arquivo de jogadas, ou seja, dois jogadores perdem antes do DUMP.

Teste 2:

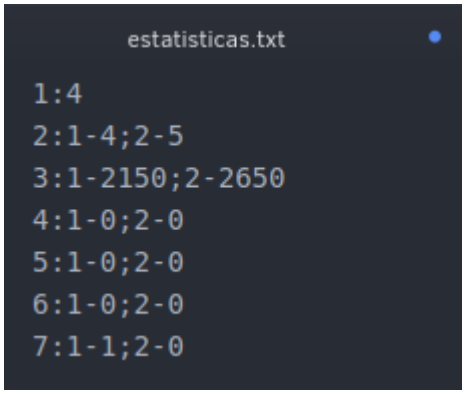
Tabuleiro




Jogadas




Saída



Este teste representa o cenário em que o tabuleiro é menor do que 6 casas, ou seja, a chance de um jogador dar a volta em uma jogada é alta.

Teste 3:

Tabuleiro

tabuleiro.txt

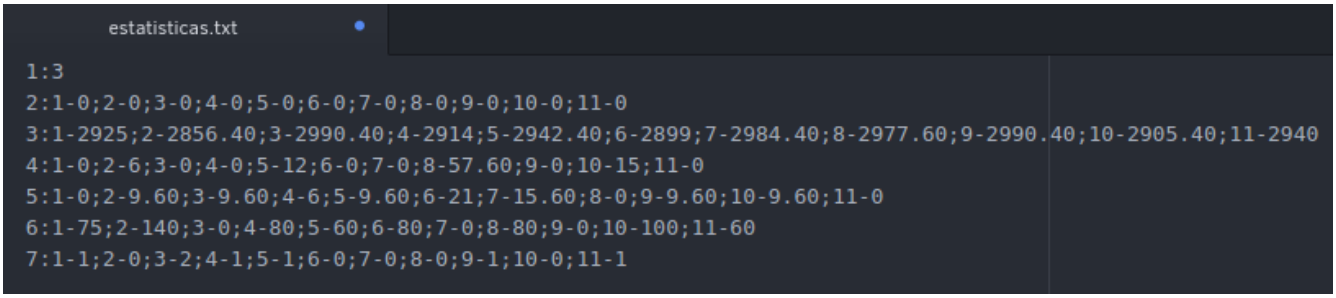
```
36
1;1;1
2;23;3;5;100;15
3;17;2
4;9;3;4;80;12
5;12;3;4;80;12
6;22;2
7;25;3;1;50;5
8;4;3;2;60;10
9;7;3;4;80;12
10;15;3;3;75;13
11;20;3;5;100;15
12;21;3;4;80;12
13;36;2
14;33;3;3;75;13
15;8;2
16;14;3;3;75;13
17;24;2
18;26;1;50;5
19;5;3;2;60;10
20;19;3;1;50;5
21;35;3;3;75;13
22;6;2
23;16;3;1;50;5
24;18;3;2;60;10
25;3;3;2;60;10
26;13;3;5;100;15
27;30;3;2;60;10
28;29;3;2;60;10
29;27;3;4;80;12
30;34;3;3;75;13
31;2;2
32;31;2
33;32;2
34;28;3;2;60;10
35;10;3;4;80;12
36;11;5;100;15
```

Jogadas

jogadas.txt

```
29%11%3000
1;1;5
2;2;2
3;3;1
4;4;1
5;5;3
6;6;2
7;7;3
8;8;6
9;9;6
10;10;6
11;11;1
12;1;5
13;2;4
14;3;5
15;4;2
16;5;2
17;6;6
18;7;3
19;8;4
20;9;1
21;10;6
22;11;3
23;1;4
24;2;5
25;3;1
26;4;6
27;5;1
28;6;4
DUMP
```

Saída




O último teste foi feito para testar a capacidade do programa de executar uma entrada grande, com muitos jogadores e várias rodadas, além do fato de a entrada do tabuleiro estar fora de ordem.

O programa executou como esperado em todos os testes e os resultados encontrados foram satisfatorios.

## Conclusão:

O problema proposto em si não era de grande dificuldade, o maior desafio foi por em prática os conceitos recentemente aprendidos de orientação a objetos. A decisão de quais informações devem ser agrupadas juntas e quais seriam as responsabilidades de cada classe foi o principal obstáculo. Uma vez superado, restou a visualização de como o código em si seria executado, ou seja, as colaborações e ligações entre classes. De restante, apenas dificuldades sintáticas de uma linguagem não muito familiar, que foram facilmente superadas.

## Bibliografia:


- https://stackoverflow.com/
- https://www.youtube.com/watch?v=gsy5GqwWqjw&list=PLesCEcYj003Rfzs39Y4Bs_chpkE276-gD&index=1
- https://www.oracle.com/java/index.html
- https://docs.oracle.com/javase/tutorial/