DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO

UNIVERSIDADE FEDERAL DE MINAS GERAIS, BRAZIL

(UFMG)

Monopoly

Uma versão simplificada escrita em Java

Belo Horizonte, 17 de Abril de 2017

Autores:

Daniel Vieira da Silva Cruz & Eugênio Pacceli Reis da Fonseca

1 - Introdução

Para o primeiro trabalho prático de Programação Modular, matéria lecionada por Douglas G. Macharet pelo Departamento de Ciência da Computação, da Universidade Federal de Minas Gerais, foi proposta a implementação de uma versão simplificada do conhecido jogo “Monopoly”, de tabuleiro.

O jogo consiste em um tabuleiro com casas e um dado. Os jogadores começam numa posição inicial conhecida, cada um rola o dado na sua vez e deve-se mover para a casa do tabuleiro equivalente à soma da posição atual do jogador com o número sorteado no dado. De acordo com a casa e da situação dessa, alguma consequência acontece para o jogador que rolou o dado.

As casas do tabuleiro podem ser do tipo inicial, “passe a vez” (nada acontece), ou uma propriedade. Uma propriedade é uma casa que pode ser comprada ou alugada por/para outros jogadores. No início do jogo, todas as propriedades do jogo pertencem à uma entidade do jogo chamada “Banco”. Todas as casas do “Banco” podem ser compradas pelos jogadores: desde que estes possuam dinheiro suficiente. Uma vez comprada, uma casa pertence a um jogador, e para todo jogador que cair nela, o dono recebe uma quantia referente ao aluguel dessa (e o jogador que caiu nela perde essa quantia).

As casas do tipo propriedade tem nomes (hospital, residência, indústria, etc) ilustrativos e valores de compra e aluguel próprios. O jogo termina quando somente um jogador possuir dinheiro (os jogadores vão perdendo dinheiro para o aluguel das casas sorteadas no dado, a não ser que sejam do banco ou deles mesmos). Os jogadores podem receber dinheiro via aluguel de suas casas, ou completando uma volta no tabuleiro.

A implementação deverá ser feita observando o uso das boas práticas do paradigma de programação orientado à objetos.

2 - Apresentação da implementação

O programa foi escrito para Java 8, testado na JDK 8u121, da Oracle.

Pacote edu.monopoly.app

Esse pacote contém a classe Main, responsável por associar os arquivos de entrada à um objeto leitor de arquivos, enviá los ao objeto emulador do jogo e pedir a execução daquela instância de jogo. A classe Main contém o ponto de entrada do programa.

Pacote edu.monopoly.exceptions

O pacote contém as exceções lançadas pela aplicação. As exceções são classes que representam situações inesperadas que podem ocorrer durante a execução do jogo, aproveitando o mecanismo de “try { } catch (Exception)” provido pelo Java. As exceções possíveis para a nossa implementação de Monopoly são:

`InsufficientGameInformationException`: Um objeto emulador de jogo não recebeu informações suficientes sobre a instância do jogo, e foi pedido para emular o mesmo.

`InvalidCellTypeException`: Uma casa do tabuleiro foi lida do arquivo de entrada do tabuleiro, mas seu tipo não consta nas especificações.

`InvalidDiceRollException`: Um número lido como sorteado de um dado, do arquivo de entrada de jogadas, não existe num dado de 6 faces.

`UnexpectedNegativeNumberException`: Uma transação (pagar ou receber aluguel, comprar propriedade, passar pela célula inicial) recebeu como entrada um valor representando dinheiro negativo.

Pacote edu.monopoly.game

O pacote está dividido em subpacotes, que contêm as classes dos conceitos existentes num jogo de Monopoly. Na raiz desse pacote se encontra o objeto `GameEmulator`, responsável por receber uma instância válida de um jogo de Monopoly, bem como uma PrintStream de saída, executar aquela instância do jogo e imprimir o resultado e estatísticas para essa stream.

`GameEmulator` mantém o controle de todos os objetos importantes à instância de uma partida de Monopoly, manipulando-os estritamente de acordo com as regras do jogo.

Uma instância do jogo é configurada, internamente, para um objeto de `GameEmulator` instanciado e já ciente de todas as informações necessárias, no método da classe chamado `configureInternalGameStart`.

O jogo é executado em `play` - método que contém o game loop -, com a ajuda de métodos auxiliares como `computePlayerMove` - executa as regras do jogo para cada jogada de jogador, decidindo o que fazer de acordo com a casa do tabuleiro que esse cair e seu estado atual -, `verifyAndComputeLaps` - mantêm a navegação dos jogadores no tabuleiro cíclica -, `hasWinner` - verifica se o jogo já terminou - e `generateStatistics` - percorre a lista de jogadores e constrói a string de saída de estatísticas do jogo com as informações obtidas de cada um-.

O jogo termina quando apenas um jogador na lista de jogadores está ativo, ou `GameEmulator` encontra um comando DUMP.

Pacote edu.monopoly.game.actors

O pacote de atores contém a classe abstrata `GameActor`, que representa entidades que interagem com as células do jogo, além de seus filhos, `Bank` e `Player`.

A `GameActor` é uma classe que pode manipular dinheiro e possuir propriedades, contendo um atributo que é uma lista de todas as células que

representam uma propriedade pertencente à alguma instância de um objeto dos filhos dessa classe.

A classe `GameActor` também não permite o “set” direto ao dinheiro que um objeto dessa pode possuir (uma vez instanciado). O acesso é feito pelos métodos `addAmount` e `subtractAmount`, ambos recebendo valores double que não podem ser negativos.

As especializações concretas dessa classe são `Bank` e `Player`. `Bank` possui dinheiro infinito e implementa o padrão de projeto Singleton, já que seu estado é o mesmo em todas as jogadas, e em todos os jogos, só precisando ser instanciado uma vez para toda a execução do programa.

A classe `Player` é mais complexa, pois é capaz de comprar e alugar propriedades, deve saber em que célula do tabuleiro está , além de que cada objeto deve manter suas estatísticas e atualizá-las durante todo o jogo.

Em `Player`, também existe um atributo booleano chamado isActive, que determina se o jogador ainda está participando do jogo ou não. O atributo isActive é mudado para falso no método de `Player` chamado `payRentTo`, quando é averiguado que aquele objeto não possui dinheiro suficiente para pagar o aluguel da propriedade recebida por parâmetro, caindo na regra do jogo que diz que ele perdeu.

Pacote edu.monopoly.game.board

Esse pacote contém a classe `Board`, que representa um tabuleiro do jogo Monopoly. O tabuleiro é manipulado por um objeto da classe `GameEmulator`, para executar a instância de um jogo.

Uma lista de `BoardCell` (do pacote definido a seguir) está presente na classe `Board`, contendo todas as células daquele tabuleiro ordenadas em ordem de posição, aproveitando o conceito de polimorfismo. Essa lista foi gerada por um objeto leitor de arquivos de entrada, e passada para o tabuleiro.

A lista não é cíclica mas a natureza do jogo é, o caminhamento, efetuado por `GameEmulator`, leva isso em consideração e controla as posições dos jogadores em

cada célula, navegando por essa lista pelo resto de (posição atual do jogador + número tirado do dado) / (tamanho da lista de células de um tabuleiro).

Pacote edu.monopoly.game.board.cells

O pacote de células contêm a abstração e implementação de todos os tipos de célula possíveis no Monopoly. Através de herança (classe mãe abstrata `BoardCell`), `StartCell`, `PropertyCell` e `PassTurnCell` especializam os detalhes de uma célula do tabuleiro de Monopoly.

A classe abstrata BoardCell define que todas as classes de célula sabem suas posições no tabuleiro.

`StartCell` representa a célula inicial do jogo, pela qual, quando o jogador passa completando uma volta, esse recebe uma quantia em dinheiro (definido como 500.00 pela documentação).

`PassTurnCell` representa a célula de passar a vez, quando um jogador cai nela.

`PropertyCell` representa as células de propriedade, que podem ser alugadas e vendidas para jogadores, e que começam pertencendo à entidade “Banco”. Cada célula de propriedade também é de um tipo, representado pelo atributo `PropertyType` da classe, esses definidos no enumerador público `PropertyType`. Escolhemos a composição invés de herança para representar essa natureza.

Uma `PropertyCell` também conhece seu dono atual, e é capaz de atualizar a lista de propriedades dos `GameActors` donos quando um novo dono é passado pelo método `setOwner(GameActor)`: o dono atual recebe a propriedade em sua lista, e o dono antigo tem essa removida.

Pacote edu.monopoly.game.commands

O pacote `commands` contém as classes que representam comandos para o `GameEmulator` do jogo. Comandos são as linhas lidas do arquivo de jogada, podendo essas serem uma jogada de um jogador ou um comando “DUMP”, que termina o jogo.

A classe `Command` é a classe abstrata pai de todos os comandos, sendo que esses possuem, em comum, um tipo listado no enumerador `CommandType`.

Um objeto de `DumpCommand` na lista de comandos a serem executados por `GameEmulator`, significa que aquela instância do jogo deve parar; já um objeto do tipo `PlayCommand` definie uma jogada para o GameEmulator, através dos setters e getters, contendo um jogador e o número que esse rolou no dado para aquela jogada.

A lista de Commands dentro de GameEmulator se encontra na ordem recebida do arquivo, e são executados um após o outro pelo emulador.

Pacote edu.monopoly.io

Esse pacote contém a interface `GameReader`, que define um contrato para uma classe leitura de arquivos de entrada, bem como `GameReaderImpl`, classe que implementa o contrato e é responsável por ler os arquivos de entrada e gerar os objetos úteis à `GameEmulator`, como `Board`, mapa de jogadores por id e a lista de `Command`’s.

Serviços públicos providos por um `GameReader`:

```
public void setSources(String boardInputAddr, String diceRollsInput);
public List<Command> generateCommandsList() throws InvalidDiceRollException, IOException;
public Board generateBoard() throws InvalidCellTypeException, IOException;
public Map<String, Player> generatePlayersList() throws IOException;
```

A classe `GameReaderImpl` possui alguns métodos internos auxiliares no comprimento do contrato acima.

3 – Testes

jogadas.txt **(entrada)**	tabuleiro.txt **(entrada)**	estatisticas.txt **(saída)**
13%3%3000 1;1;3 2;2;4 3;3;6 4;1;3 5;2;2 6;3;1 7;1;1 8;2;4 9;3;4 10;1;6 11;2;3 12;3;4 DUMP	10 1;1;1 2;2;3;2;150;20 3;3;3;1;100;10 4;4;3;4;350;30 5;5;3;1;100;10 6;6;3;2;150;20 7;7;3;3;100;10 8;8;3;5;500;10 9;9;2 10;10;3;1;100;10	1:4 2:1-1;2-1;3-1 3:1-3195.00;2-3285.00;3-267 0.00 4:1-105.00;2-0.00;3-70.00 5:1-60.00;2-115.00;3-0.00 6:1-350.00;2-100.00;3-900.00 7:1-0;2-0;3-0
13%3%3000 1;1;2 2;2;4 3;3;3 4;1;6 5;2;2 6;3;5 7;1;1 8;2;2 9;3;5 10;1;6 11;2;3 12;3;4 DUMP	10 1;1;1 2;2;3;2;150;20 3;3;3;1;150;50 4;4;3;4;350;30 5;5;3;1;220;10 6;6;3;2;150;20 7;7;3;3;100;10 8;8;3;5;450;50 9;9;2 10;10;3;1;100;20 11;11;3;2;200;30 12;12;3;3;300;40 13;13;3;4;400;50	1:4 2:1-1;2-0;3-1 3:1-3250.00;2-2402.00;3-312 8.00 4:1-0.00;2-22.00;3-0.00 5:1-0.00;2-0.00;3-22.00 6:1-250.00;2-620.00;3-350.00 7:1-1;2-1;3-1

Para rodar o trabalho: “make all”, em seguida “make run”. Os arquivos binários e as entradas estão em “.\monopolyApp”.

4 – Conclusão

O trabalho foi muito proveitoso e o grupo teve a oportunidade de implementar (esperamos que corretamente) vários dos conceitos do paradigma orientado por objetos, aprendidos em sala de aula e nos estudos, de acordo com a situação de cada subparte do problema e nosso julgamento. Executamos a tarefa tentando sempre organizar o projeto de modo que este ficasse mais modularizado o possível, e o código, reaproveitável.

5 – Bibliografia

- Documentação e especificação da API do Java 8, Oracle < https://docs.oracle.com/javase/8/docs/api/ >
- Java - Como Programar - 10ª Ed. 2016, Prentice Hall, Deitel
- Trabalho Prático 1 – Banco Imobiliário Modular, Prof. Douglas G. Macharet, disponibilizado no moodle da disciplina e no diretório raiz desse trabalho.