DA1MCTA018-13SA - Programação Orientada a Objetos - Paulo Henrique Pisani - 2021.2

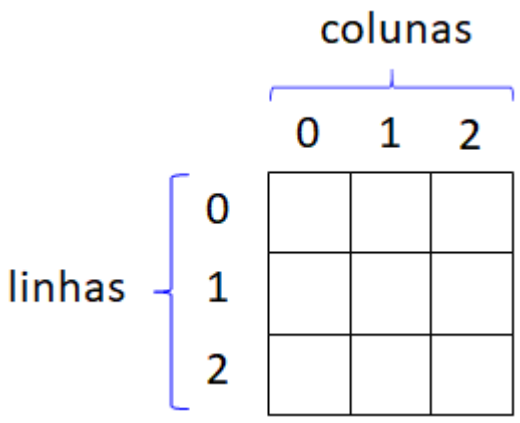
Descrição | Visualizar envios

[EP] Jogo da velha

Data de entrega: quarta, 4 Ago 2021, 23:59
Arquivos requeridos: TabuleiroEstrategia.java, Estrategias.java, Tabuleiro.java, EstrategiaJogo.java (Baixar)
Tipo de trabalho: Trabalho individual

O funcionário de uma empresa escreveu um programa para o Jogo da Velha, mas gostaria de adicionar algumas estratégias de jogo automáticas. Para isso, outro funcionário sugeriu utilizar o **padrão de projeto Strategy**, de forma que a classe com o tabuleiro utilize um algoritmo por meio de uma interface que pode ser implementada de diversas formas diferentes. Cada implementação representaria uma estratégia de jogo.

Um tabuleiro é representado por uma matriz de char conforme a seguir:



Duas estratégias que devem ser implementadas são: Estratégia A e Estratégia B (*há também uma Estratégia C que já está implementada no sistema de correção automática*). As estratégias seguem a ideia de percorrer a matriz e retornam como próxima jogada a primeira célula que encontram livre. O percurso das estratégias A e B é apresentado a seguir:

Estratégia A

1	2	3
4	5	6
7	8	9

Estratégia B

2	6	3
7	1	8
4	9	5

A classe Tabuleiro e a interface EstrategiaJogo já estão implementadas no sistema de correção automática (o código pode ser consultado na aba Editar).

Classe **Tabuleiro**:

- **public Tabuleiro()** - construtor da classe Tabuleiro. Inicializa todas as posições do tabuleiro com o caractere ponto: '.'
- **public char[][] getTabuleiro()** - retorna uma cópia dos dados do tabuleiro (matriz 3x3)
- **public char getJogadorVencedor()** - returna o jogador vencedor: 'X' ou 'O'. Se não existir vencedor, retorna ponto '.'. Caso seja empate, retorna hífen '-'.

- **public void jogar(char jogador, int linha, int coluna)** - recebe o jogador (pode ser 'X' ou 'O') e joga na posição informada. O método também imprime o tabuleiro depois de realizar a jogada.

Interface **EstrategiaJogo**:

- **int[] getProximaJogada(char[][] dadosTabuleiro, char jogador)** - a implementação deve retornar as coordenadas da próxima jogada. O retorno é um vetor de duas posições, em que a posição 0 representa a linha e a posição 1 a coluna.

Tarefa

Escreva as classes TabuleiroEstrategia, EstrategiaA e EstrategiaB conforme descrito a seguir (todas as classes devem estar no **pacote jogo**):

[acesso public] Classe **TabuleiroEstrategia** (estende a classe Tabuleiro)

- **public void setStrategyJogador1(EstrategiaJogo jogador1)** - armazena uma referência para a implementação de EstrategiaJogo para o jogador 1.
- **public void setStrategyJogador2(EstrategiaJogo jogador2)** - armazena uma referência para a implementação de EstrategiaJogo para o jogador 2.
- **public void jogarPartida()** - executa uma partida de jogo da velha completa. A cada jogada, o método getProximaJogada da estratégia correspondente é chamado e ele retorna as coordenadas para a próxima jogada. Ao receber as coordenadas, deve ser chamado o método jogar (herdado da classe Tabuleiro) passando as coordenadas e o jogador ('X' ou 'O'). O método jogarPartida deve intercalar jogadas do jogador 1 e do jogador 2 até o jogo terminar. Para saber se o jogo terminou, use o método getJogadorVencedor (herdado da classe Tabuleiro). Considere que sempre o jogador 1 começa e que ele utiliza 'X', portanto, o jogador 2 utiliza 'O'.

[acesso package] Classe **EstrategiaA** (implementa a interface EstrategiaJogo)

- **public int[] getProximaJogada(char[][] dadosTabuleiro, char jogador)** - retorna as coordenadas da próxima jogada seguindo a estratégia A.

[acesso package] Classe **EstrategiaB** (implementa a interface EstrategiaJogo)

- **public int[] getProximaJogada(char[][] dadosTabuleiro, char jogador)** - retorna as coordenadas da próxima jogada seguindo a estratégia B.

Importante: O programa principal já existe no sistema de correção automática. Submeta apenas as classes especificadas (**pacote jogo**): **TabuleiroEstrategia, EstrategiaA e EstrategiaB.**

- A classes submetidas não podem realizar impressão de dados, java.util ou utilizar import (*observação: a única forma de impressão permitida é por meio do método jogar da classe Tabuleiro, que imprime o tabuleiro a cada jogada realizada*).
- A classe Tabuleiro e a interface EstrategiaJogo já estão prontos (não altere o código desses arquivos).
- O sistema de correção também possui uma estratégia já implementada (a EstrategiaC) que é usada apenas para testar o código submetido. Não é necessário conhecer o funcionamento da estratégia C para realizar este exercício. Ao utilizar o padrão de projeto Strategy, a classe TabuleiroEstrategia deve funcionar para qualquer implementação da interface EstrategiaJogo.

Casos de teste

Formato dos casos de teste (que aparecem ao avaliar as classes no sistema de correção automática):

Entrada:

- estratégia para o jogador 1 ('X')
- estratégia para o jogador 2 ('O')

Saída:

- verificação das classes
- comandos executados e saídas obtidas

Arquivos requeridos

TabuleiroEstrategia.java

```
1  // Escreva aqui a classe TabuleiroEstrategia
2
3  package jogo;
```

Estrategias.java

```java
// Escreva aqui as classes EstrategiaA e EstrategiaB

package jogo;
```

Tabuleiro.java

```java
// Não altere este arquivo

package jogo;

public class Tabuleiro {

    private char[][] tabuleiro;

    public Tabuleiro() {
        this.tabuleiro = new char[3][3];
        for (int linha = 0; linha < 3; linha++)
            for (int coluna = 0; coluna < 3; coluna++)
                this.tabuleiro[linha][coluna] = '.';
    }

    public char[][] getTabuleiro() {
        char[][] copia = this.tabuleiro.clone();
        for (int i = 0; i < this.tabuleiro.length; i++)
            copia[i] = this.tabuleiro[i].clone();
        return copia;
    }

    public char getJogadorVencedor() {
        for (int linha = 0; linha < 3; linha++) {
            if (tabuleiro[linha][0] != '.' && tabuleiro[linha][0] == tabuleiro[linha][1] && tabuleiro[linha][1] == tabuleiro[linha][2])
                return tabuleiro[linha][0];
        }
        for (int coluna = 0; coluna < 3; coluna++) {
            if (tabuleiro[0][coluna] != '.' && tabuleiro[0][coluna] == tabuleiro[1][coluna] && tabuleiro[1][coluna] == tabuleiro[2][coluna])
                return tabuleiro[0][coluna];
        }
        if (tabuleiro[0][0] != '.' && tabuleiro[0][0] == tabuleiro[1][1] && tabuleiro[1][1] == tabuleiro[2][2])
            return tabuleiro[0][0];
        if (tabuleiro[0][2] != '.' && tabuleiro[0][2] == tabuleiro[1][1] && tabuleiro[1][1] == tabuleiro[2][0])
            return tabuleiro[0][2];

        for (int linha = 0; linha < 3; linha++)
            for (int coluna = 0; coluna < 3; coluna++)
                if (this.tabuleiro[linha][coluna] == '.')
                    return '.';

        return '-';
    }

    public void jogar(char jogador, int linha, int coluna) {
        if (jogador != 'X' && jogador != 'O')
            throw new RuntimeException(String.format("jogar(): jogador invalido: [%c].", jogador));
        if (tabuleiro[linha][coluna] != '.')
            throw new RuntimeException(String.format("jogar(): Posicao ja preenchida - linha=%d coluna=%d", linha, coluna));
        if (getJogadorVencedor() != '.')
            throw new RuntimeException(String.format("jogar(): O jogo esta encerrado.", linha, coluna));

        tabuleiro[linha][coluna] = jogador;

        for (int i = 0; i < 3; i++)
            System.out.println(String.format("%c %c %c", tabuleiro[i][0], tabuleiro[i][1], tabuleiro[i][2]));

        System.out.println("-----");
    }
}
```

EstrategiaJogo.java

```java
// Não altere este arquivo

package jogo;

public interface EstrategiaJogo {
    int[] getProximaJogada(char[][] dadosTabuleiro, char jogador);
}
```

VPL

◄ [Vídeo] Strategy

Seguir para...

[EP] REC - Jogo da velha ►

(educadores/alunos) possam criar cursos, gerenciá-los e participar de maneira colaborativa.

Universidade Federal do ABC - Moodle (2020)

Obter o aplicativo para dispositivos móveis