Formabase
Aprenda passo a passo

Java
Exercicio - Jogo do Galo

Metodologia

- Interprete o documento calmamente e com atenção.
- Acompanhe a execução do exercício no seu computador.
- Não hesite em consultar o formador para o esclarecimento de qualquer questão.
- Não prossiga para o ponto seguinte sem ter compreendido totalmente o ponto anterior.
- Caso seja necessário, execute várias vezes o exercício até ter compreendido totalmente o processo.

Conteúdo programático

1. Jogo do Galo

Vamos utilizar os conceitos anteriores reproduzindo o famoso jogo do galo, de maneira a ter uma representação de arrays bidimensionais familiar e simples. O desenvolvimento deste pequeno jogo servirá para praticar os vários conceitos abordados até agora, bem como observar as diversas aplicações de ciclos `for` encadeados e a utilização de arrays bidimensionais.

- **Crie** uma classe de nome `Galo.java`

› **Introduza** o seguinte código:

```
public class Galo {
    static char[][] tabuleiro;

    public static void main(String[] args) {

    }
}
```

Note que a declaração do array foi feita fora do método `main`, de forma a ficar acessível em todos os locais. Lembre-se que quando termina um método deixa de ter acesso às variáveis que foram declaradas nele.

A variável `tabuleiro` representa as 9 quadrículas de um jogo do galo, em que cada uma contém o respetivo valor. Os valores utilizados serão o 'X' , 'O' e o espaço ' '.

Segue-se um método de inicialização do tabuleiro, definindo todas as suas posições como estando vazias.

› **Adicione** à classe o método abaixo:

```
public static void inicializacaoTabuleiro(){
    tabuleiro = new char[3][3]; //cria e define o tamanho do array

    for (int linha = 0;linha < 3;linha++){
        for(int coluna = 0;coluna < 3;coluna++){
            tabuleiro[linha][coluna] = ' '; //vazio
        }
    }
}
```

Agora é preciso que o código principal `main` execute esta função.

- **Insira** dentro do método `main` a instrução:

```
inicializacaoTabuleiro();
```

Neste momento garantimos que o tabuleiro do galo está a ser inicializado logo no início do programa. Agora é necessário um método para mostrar como se encontra o tabuleiro no momento.

- **Adicione** mais um método com o código:

```
public static void mostraTabuleiro(){
    for (int linha = 0;linha < 3;linha++){
        System.out.println("-------------");
        System.out.print("| ");

        for(int coluna = 0;coluna < 3;coluna++){
            System.out.print(tabuleiro[linha][coluna] + " | ");
        }

        System.out.println();
    }

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

No código adicionado existem várias linhas que servem apenas para formatação, com o objetivo de dar um aspeto de grelha.

Esta é a instrução que escreve as linhas horizontais da grelha:

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

As duas instruções a seguir são as que escrevem as linhas verticais da grelha:

```
System.out.print("| ");
System.out.print(tabuleiro[linha][coluna] + " | ");
```

- **Adicione** ao final do método `main` a chamada ao método que adicionou

```
mostraTabuleiro();
```

- **Teste** a classe

> A grelha que vê neste momento a ser escrita no ecrã encontra-se vazia pois ainda não criámos código para a preencher.

Passemos ao preenchimento do tabuleiro. Para este fim iremos criar um método que solicita ao utilizador a linha e coluna onde pretende jogar, guardando no array o respetivo símbolo. Este método irá devolver um valor booleano que representa se a jogada feita foi válida ou não.

- **Adicione** um novo método com o seguinte código:

```
public static boolean fazerJogada(char simbolo){
    Scanner teclado = new Scanner(System.in);
```

```
    System.out.println("Insira a linha onde quer jogar");
    int linha = teclado.nextInt();

    if((linha < 1) || (linha > 3)){ //linha invalida
        return false;
    }

    System.out.println("Insira a coluna onde quer jogar");
    int coluna = teclado.nextInt();

    if((coluna < 1) || (coluna > 3)){ //coluna invalida
        return false;
    }

    //guardar a jogada que foi feita
    tabuleiro[linha-1][coluna-1] = simbolo;

    return true; //devolver sucesso
}
```

O valor booleano que este método devolve serve para solicitar uma nova jogada caso a corrente não seja válida. A validação de jogada é feita dentro do método, testando os índices para a linha e para a coluna, como é indicado no exemplo abaixo:

```
if((linha < 1) || (linha > 3)){
if((coluna < 1) || (coluna > 3)){
```

Desta forma foi restringido o conjunto de valores válidos aos números que se encontram entre **1** e **3**.

A atribuição do símbolo numa determinada posição tem uma particularidade:

```
tabuleiro[linha-1][coluna-1] = simbolo;
```

Como deve ter notado, ambos os índices estão decrementados numa unidade. Isto foi feito porque o nosso array `tabuleiro` com tamanho 3 contém apenas os índices de 0 a 2. Logo, se o jogador escolhesse jogar na linha 3 coluna 3, ficaria fora do quadro gerando um erro. Como para o jogador o quadro começa no um e não no zero, a diferença entre a indicação visual das linhas/colunas e dos índices válidos é de uma unidade.

Vamos agora utilizar as funções de forma a ver o funcionamento do jogo.

- **Adicione** ao fim do método `main` as seguintes instruções:

```
fazerJogada('X');
mostraTabuleiro();
```

- **Compile** e **teste** a classe

  - Confirme que agora consegue fazer uma jogada e ver o quadro com essa jogada feita.

Vamos agora modificar o método `main` de forma a que este peça jogadas sucessivamente. Para isso recorremos a um ciclo `while` e a uma variável booleana que nos indica se já chegámos ao estado final do jogo.

- **Altere** o método `main` para ficar da seguinte forma:

```
public static void main(String[] args) {
    inicializacaoTabuleiro ();

    char simbolocorrente = 'X';
    boolean jogoterminado = false;
```

```
    while (jogoterminado == false){
        fazerJogada(simbolocorrente);

        //alternar o simbolo
        if(simbolocorrente == 'X'){
            simbolocorrente = 'O';
        }
        else {
            simbolocorrente = 'X';
        }

        mostraTabuleiro();
    }
}
```

Compile e **teste** o programa

Repare que o programa neste momento não tem fim. Faltam funcionalidades para analisar se o quadro está numa posição de vitória/empate/derrota. O próprio método que faz a jogada "fazerJogada" não valida ainda se a posição que o jogador escolheu já está preenchida, nem é pedida nova jogada caso a última seja incorreta.

```
char simbolocorrente = 'X';
```

Foi introduzido o conceito do símbolo do jogador que está a jogar, representado por uma variável. Esta é depois passada como parâmetro ao método `fazerJogada` para que este possa assinalar o símbolo correto no tabuleiro.

```
if(simbolocorrente == 'X'){
    simbolocorrente = 'O';
}
else {
```

```
    simbolocorrente = 'X';
}
```

Este bloco de `if` e `else` alterna a variável `simbolocorrente` entre o valor 'X' e 'O'. Isto faz com que cada jogada seja executada com o símbolo inverso da jogada anterior.

Falta agora fazer com que a variável `jogoterminado` contenha o valor **true** ou **false**, consoante o jogo esteja num estado terminal ou não.

› **Introduza** o seguinte código dentro ciclo `while` do método `main` a seguir à instrução `fazerJogada(simbolocorrente);`

```
jogoterminado = jogoAcabado();
```

Esta instrução permite controlar o fim do ciclo `while`. Quando o método `jogoAcabado` devolver o valor verdadeiro (`true`), o ciclo `while` será terminado e consequentemente também o jogo. Logo, este método deverá apenas devolver `true` quando o jogo chegar a um estado de vitória para um dos lados, ou empate.

› **Crie** o método `jogoAcabado` tendo em conta as seguintes especificações:

- Este método deverá validar se todas as células da primeira linha são iguais, o mesmo para a segunda e terceira linha. Este procedimento terá que ser feito também para as colunas e as diagonais. Estas verificações definem a vitória. Após esta verificação, e caso não haja vitória, deverá ver se todas as células estão preenchidas para perceber se existe um caso de empate.

- Em todas as verificações acima ele deve devolver o valor `true` caso esteja num caso terminal e escrever esse estado no ecrã.

- A validação poderá ser feita com comparações simples através de vários `if`'s. No entanto, convém fazer uso de ciclos `for` de maneira a tornar o método flexível, algo que será necessário um pouco mais à frente.

- **Compile** e **teste** a classe

  Confirme que está a funcionar da forma correta, terminando o jogo com identificação dos casos de vitória e empate.

Falta agora apenas definir as jogadas inválidas e fazer com que o programa volte a pedir uma jogada caso a anterior seja inválida. O método `fazerJogada` já devolve um valor que indica se a jogada é válida, portanto é necessário alterar alguns pormenores no método `main`.

- **Substitua** a instrução `fazerJogada(simbolocorrente);` no método `main` para as seguintes instruções:

```
boolean jogadavalida = fazerJogada(simbolocorrente);

while(jogadavalida == false){
    System.out.println("Jogada invalida, jogue de novo");
    jogadavalida = fazerJogada(simbolocorrente);
}
```

- **Teste** este novo código

  Confirme que agora são validadas as referências para as linhas e colunas que se encontram fora do tabuleiro, portanto acima de 3 e abaixo de 1.

Em alternativa poderia ser usado um código mais compacto cuja funcionalidade era exatamente a mesma:

```
while(fazerJogada(simbolocorrente) == false){
    System.out.println("Jogada invalida, jogue de novo");
}
```

Vamos agora validar se a jogada incide sobre uma posição do tabuleiro já preenchida:

- **Altere** o método `fazerJogada` **inserindo** o código que se encontra destacado:

```
public static boolean fazerJogada(char simbolo){
    Scanner teclado = new Scanner(System.in);

    System.out.println("Insira a linha onde quer jogar");
    int linha = teclado.nextInt();
    if((linha < 1) || (linha > 3)){ //linha invalida
        return false;
    }

    System.out.println("Insira a coluna onde quer jogar");
    int coluna = teclado.nextInt();

    if((coluna < 1) || (coluna > 3)){ //coluna invalida
        return false;
    }

   if(tabuleiro[linha-1][coluna-1] != ' '){ //posicao ja preenchida
      return false;
   }

   //guardar a jogada que foi feita
    tabuleiro[linha-1][coluna-1] = simbolo;

    return true; //devolver sucesso
}
```

- **Verifique** que agora o jogo já não permite jogar sobre uma quadrícula preenchida

De maneira a podermos recordar matéria abordada em módulos anteriores, vamos introduzir no jogo o conceito de constantes. O tamanho do tabuleiro é, neste momento estático e utilizado regularmente nos ciclos que o percorrem. Mas, transformando este valor numa constante, aumentamos a flexibilidade do programa, caso eventualmente queiramos alterar o tamanho do tabuleiro.

- **Insira** no início da classe abaixo da declaração da variável `tabuleiro` a instrução:

```
static final int TAMANHO = 3;
```

- **Altere** agora **todas as ocorrências** do número **3** no código por `TAMANHO`

- **Teste** a classe

  Confirme que o funcionamento desta se mantém igual.

Como agora as condições de paragem dos ciclos estão dependentes da constante "TAMANHO", ao alterarmos o seu valor, modificamos o comportamento do programa.

- **Altere** o valor da constante `TAMANHO` para **4** e **teste** a classe

A maior parte do código está já adaptado para lidar com qualquer dimensão de tabuleiro graças ao uso da constante, mas existem ainda métodos que necessitam de algumas alterações. É o caso do método `jogoAcabado` caso não tenham sido usados ciclos `for`, e o método `mostraTabuleiro` cuja impressão das linhas deverá acompanhar o tamanho do tabuleiro.

- **Altere** os **métodos** mencionados acima de maneira a se adaptarem ao tamanho do tabuleiro