1º Teste de Introdução à Programação

Duração: 2 horas

Departamento de Informática, FCT NOVA
21 de Novembro de 2020

(O teste tem **18** páginas e **3** grupos)

Número: _____ **Nome:** _____

Regras do Teste

- Responda a cada pergunta na caixa atribuída para o efeito.
- Identifique todas as folhas do enunciado no local apropriado.
- As respostas podem ser escritas a lápis.
- Em cima da mesa, só pode ter o documento de identificação e material de escrita (caneta, lápis, borracha).
- O teste é sem consulta. Não pode consultar quaisquer elementos para além deste enunciado.
- Não pode usar dispositivos eletrónicos (como calculadoras, telemóveis, tablets, smartwatches e portáteis).
- Não pode ter folhas de rascunho. Use os espaços para rascunhos no enunciado.
- Não pode desagrafar o enunciado.
- Antes de começar a resolver cada grupo, leia o enunciado das perguntas do grupo com atenção, do princípio até ao fim.
- Não há esclarecimento de dúvidas. Se suspeitar que o enunciado tem algum erro, deve avisar o docente.
- Só pode sair da sala quando o teste terminar.

No Final do Teste

- Verifique que todas as folhas estão identificadas com o seu número e o seu nome.

Grupo 1 [1+1+2 valores]

Pergunta 1.1 [1 valor] Considere a seguinte classe:

```
public class Limit {
    private static final int BOUND_INF = -100;
    private static final int BOUND_SUP = 100;
    private int state;

    public Limit( ) {
        state = 0;
    }

    public void change( int val ) {
        state = state + val;
    }

    public void reset( ) {
        if (state < BOUND_INF || state > BOUND_SUP)
            state = 0;
    }

    public int get( ) {
        return state;
    }
}
```

Indique nas caixas abaixo o que seria escrito na consola após a execução de cada grupo de instruções, executadas em sequência pela ordem indicada. Caso não seja escrito nada, escreva "Não escreve nada".

```
Limit x = new Limit();
```

```
x.change(51);
System.out.println(x.get());
```

```
x.reset();
```

```
x.change(51);
System.out.println(x.get());
```

```
x.reset();
System.out.println(x.get());
```

Pergunta 1.2 [1 valor] Considere a seguinte classe:

```
public class Horror {
    public static final int MY_CONST = 1000;
    public int sTaTe;

    public horror( ) {
        sTaTe = MY_CONST;
    }

    public change( int value ) {
        sTaTe = sTaTe + value;
    }

    public void reset( ) {
        if (state < MY_CONST or state > MY_CONST)
            state = MY_CONST
    }
}
```

Indique todos os erros que seriam assinalados por um compilador de Java:

(Esta página é para rascunho)

Pergunta 1.3 [2 valores] Considere a seguinte classe:

```
public class NatFromSeqAlg {
    private int number;

    public NatFromSeqAlg( ) {
        number = 0;
    }

    // @pre:  value >= 0 && value < 10
    public void oneMoreAlg( int value ) {
        number = number*10 + value;
    }

    public int getNumber( ) {
        return number;
    }

    public int countNumberOfAlgs( ) {
        int res = 1;
        int i = number;
        while (i >= 10) {
            res++;
            i = i / 10;
        }
        return res;
    }
}
```

Indique nas caixas abaixo o que seria escrito na consola após a execução de cada grupo de instruções, executadas em sequência pela ordem indicada. Caso não seja escrito nada, escreva "Não escreve nada".

```
NatFromSeqAlg number = new NatFromSeqAlg();
System.out.println(number.getNumber());
```

```
number.oneMoreAlg(4);
number.oneMoreAlg(2);
System.out.println(number.countNumberOfAlgs());
```

```
System.out.println(number.getNumber());
```

```
number.oneMoreAlg(42);
System.out.println(number.getNumber());
```

(Esta zona é para rascunho)

Grupo 2 [2.5+5+5 valores]

Pretende-se implementar uma versão simples do jogo de ténis de mesa entre dois jogadores. Em cada jogada, um (e só um) dos jogadores pode ganhar 1 ponto.

No ténis de mesa, nenhum jogo termina empatado. Regra geral, o vencedor é o primeiro jogador a obter 11 pontos. Mas, para o jogo terminar (e haver um vencedor), tem de haver uma diferença de pelo menos 2 pontos entre os jogadores. Ou seja, se o resultado alguma vez for 10–10, o jogo tem de continuar até que um dos jogadores consiga uma vantagem de 2 pontos. Nesse momento, o jogo termina e o vencedor é o jogador que tiver mais 2 pontos do que o outro.

Um jogador (`Player`) é identificado pelo nome (não podendo haver jogadores diferentes com o mesmo nome) e tem as seguintes funcionalidades:

- diz o seu nome (método `getName`, que devolve uma `String`);
- diz quantos pontos tem no momento corrente do jogo (método `getPoints`, que devolve um inteiro);
- adiciona mais um ponto à sua pontuação (método `addPoint`, que não devolve nada).

Pretende-se que implemente três classes: `Player` (na alínea (a)), `TableTennis` (na alínea (b)) e `Main` (na alínea (c)). Em todas, preencha apenas as caixas em branco, considerando todo o restante código necessário implementado, de acordo com este enunciado. **Pode preencher qualquer caixa mesmo que não tenha preenchido outras.**

Pergunta 2 (a) [2.5 valores]

```
public class Player {
    /** Constantes (se existir alguma) */
```

```
    /** Variáveis de instância */
```

```
    /**
     * Construtor:
     * @param o nome do jogador (String)
     * @pre:
     */
```

Parte restante da classe `Player`:

```
}
```

(Esta página é para rascunho)

Pergunta 2 (b) [5 valores]

Note que a classe `TableTennis` possui "métodos repetidos", um para cada um dos dois jogadores (os nomes dos métodos diferem apenas nos algarismos 1 e 2). Só se pedem os métodos para o jogador 1. Pode definir métodos auxiliares: possui uma caixa para esse efeito.

```
public class TableTennis {
    /** Constantes e variáveis de instância */
```

```
/**
 * Construtor:
 * @param name1 - o nome do jogador 1 (String)
 * @param name2 - o nome do jogador 2 (String)
 * @pre:
 */
```

```
/**
 * @return nome do jogador 1
 */
public String player1name( ) {



}
```

```
/**
 * @return número de pontos do jogador 1
 */
public int player1points( ) {



}
```

```
/**
 * Adiciona mais um ponto ao jogador 1
 */
public void addPointToPlayer1( ) {




}
```

```
/**
 * Diz se o jogo já terminou
 * @return true, se o jogo acabou; false, no caso contrário
 */
public boolean isGameOver( ) {









}
```

```
/**
 * @return nome do vencedor do jogo
 * @pre:
 */
public String nameOfWinner( ) {









}
```

```
/**
 * Nota:  as jogadas correspondem aos comandos APJ1 e APJ2
 *        referidos na alínea (c)
 * @return número de jogadas já efetuadas no jogo
 */
```

```
public int numberOfMoves( ) {




}
```

Assuma que os seguintes métodos existem (mas não os implemente):

```
public String player2name( ) { ··· }
public int player2points( ) { ··· }
public void addPointToPlayer2( ) { ··· }
```

Coloque na caixa abaixo os métodos auxiliares que entenda conveniente acrescentar a `TableTennis`:

```

}
```

(Esta página é para rascunho)

Pergunta 2 (c) [5 valores]

Note que o programa não termina quando o jogo chega ao fim. É necessário introduzir o comando SAIR para o programa terminar. Porém, quando o jogo está concluído, os comandos que acrescentam pontos aos jogadores (APJ1 e APJ2) possuem um comportamento distinto de quando o jogo decorre.

```
import java.util.Scanner;
public class Main {
    /** Constantes */
    private static final String PROMPT = "> ";
```

```
    /**
     * Programa principal:
     * - leitura dos nomes dos 2 jogadores
     * - criação de um objeto da classe TableTennis
     * - chamada do método interpretador de comandos
     * @param args
     */
    public static void main( String[] args ) {
        Scanner in = new Scanner(System.in);
        System.out.print("Nome do jogador 1:  ");
        String playerName1 = in.nextLine();
        System.out.print("Nome do jogador 2:  ");
        String playerName2 = in.nextLine();
        if (                                                    )
            System.out.println("Os 2 jogadores nao podem ter o mesmo nome.");
        else
            interpreter(in,                                     );
        in.close();
    }

    /**
     * Implementação do comando AJUDA
     */
    private static void help( ) {
        System.out.println(SAIR  + "\tSair do programa (terminar execucao).");
        System.out.println(AJUDA + "\tListar os comandos.");
        System.out.println(APJ1  + "\tAcrescentar 1 Ponto ao Jogador 1.");
        System.out.println(APJ2  + "\tAcrescentar 1 Ponto ao Jogador 2.");
        System.out.println(MP    + "\tMostrar os Pontos dos 2 jogadores.");
    }
```

```
/**
 * Leitura do próximo comando
 * @param in - o input
 */
private static String nextCommand( Scanner in ) {
    System.out.print(PROMPT);
    return in.nextLine().toUpperCase();
}
```

```
/**
 * Jogada - acrescenta 1 Ponto ao Jogador 1
 * Se o jogo já terminou, só escreve na consola:
 *    O jogo ja terminou.
 * Caso contrário, concretiza a jogada e escreve 2 linhas com a forma:
 *    <int> jogadas ate agora.
 *    Jogador <nome> agora tem <int> pontos.
 * Caso o jogo termine com esta jogada, escreve mais 2 linhas:
 *    O jogo terminou.
 *    Vencedor: <nome>
 * @param tt - o objeto ténis de mesa
 */
private static void processAPJ1( TableTennis tt ) {




}
```

```
/**
 * Implementação do comando SAIR.
 * Se existir um vencedor, escreve na consola uma linha da forma:
 *    Vencedor: <nome>
 * Se o jogo ainda não tinha terminado, escreve:
 *    Jogo sem vencedor.
 * @param tt - o objeto ténis de mesa
 */
```

```
private static void processExit( TableTennis tt ) {









}
```

Na implementação do interpretador de comandos pedida abaixo, considere que os comandos APJ2 e MP não existem, devendo prever o caso de comandos inválidos.

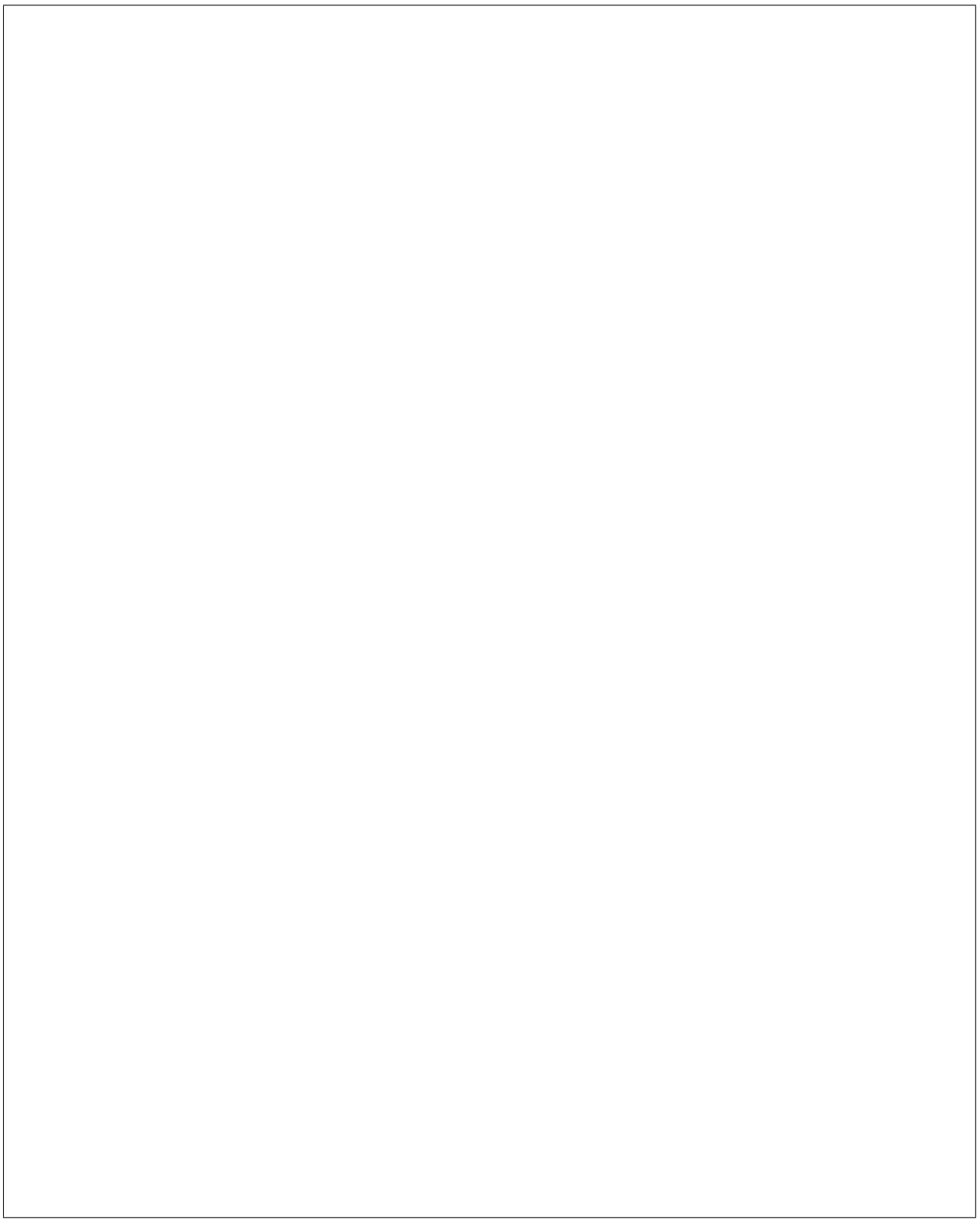
```
/**
 * Implementação parcial do interpretador de comandos
 * Caso o comando não seja AJUDA, APJ1 ou SAIR,
 * escreve na consola:
 *    Comando desconhecido.
 * @param in - o input
 * @param tt - o objeto ténis de mesa
 * @pre:  os nomes dos 2 jogadores são distintos
 */
```

```
private static void interpreter( Scanner in, TableTennis tt ) {

















}
```

Assuma que os seguintes métodos existem (mas não os implemente):

```
private static void processAPJ2( TableTennis tt) { ··· }
private static void processMP( TableTennis tt ) { ··· }
```

Coloque na caixa abaixo os métodos auxiliares que entenda conveniente acrescentar à classe `Main`:



}

Grupo 3 [3.5 valores]

Escreva um método chamado `dayOfYear` para determinar o número de ordem de uma data dentro de um ano. Queremos saber se essa data representa o 1º dia do ano, o 2º dia do ano, etc.

O método `dayOfYear` aceita três argumentos: (1) dia, (2) mês e (3) indicação se o ano é bissexto (*leap year*). O resultado é um inteiro.

Exemplos de utilização:

`dayOfYear(1, 1, false)`	retorna 1	(1 de janeiro de um ano comum)
`dayOfYear(1, 3, false)`	retorna 60	(1 de março de um ano comum)
`dayOfYear(1, 3, true)`	retorna 61	(1 de março de um ano bissexto)
`dayOfYear(31, 12, true)`	retorna 366	(31 de dezembro de um ano bissexto)

Para guardar o número de dias de cada mês, use o vetor de inteiros `lengths`, inicializado com os comprimentos dos 12 meses do ano. (Em Java, para inicializar um vetor com uma sequência de valores fixos, basta escrever essa sequência entre chavetas, como se mostra abaixo.)

Considere que a data é válida. Por esse motivo, defina a pré-condição do método.

```
/**
 * Calcula o número de ordem de uma data dentro de um ano
 * @param day - o dia, representado por um valor de 1 até 31
 * @param month - o mês, representado por um valor de 1 até 12
 * @param isLeapYear - true se o ano for bissexto; false no caso contrário
 * @return o número de ordem da data no ano
 * @pre:
 */
public static int dayOfYear( int day, int month, boolean isLeapYear ) {
    int[] lengths = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31};









}
```

(Esta página é para rascunho)