1º Teste de "Introdução à Programação" (2008/09)

Duração 2H

Instruções importantes:

- Responda a todos os grupos em folhas separadas.
- Identifique a folha de rosto do caderno de exame com o seu nome e número.
- Antes de começar a resolver um exercício, leia o enunciado do princípio até ao fim. Tem um período de 15 minutos para fazer perguntas sobre o teste ao docente presente na sala. Após esses 15 minutos, deve resolver ambiguidades que se mantenham da melhor forma.
- Pode usar caneta ou lápis. Recomendamos que use o lápis.
- Não é permitido consultar elementos para além deste enunciado.
- Qualquer tentativa de fraude comprovada acarretará a reprovação na disciplina.
- Não é permitido sair da sala antes que o teste termine.

I – Para cada um dos fragmentos de programas Java apresentados, assinale que linhas são executadas, e indique o valor que cada uma das variáveis contém após cada linha ser executada (para o ajudar, as linhas do bloco estão numeradas).

a)
```
1{
2   int x;
3   int y;
4   x = 0;
5   y = x + 1;
6   x = y - 1;
7   x = x + 2;
8}
```

b)
```
1  {
2    int a;
3    boolean b;
4    a = 2;
5    b = (a != a);
6    if (b)
7      a = a + 1;
8    else
9      b = (a >= 2);
10   if (a > 0) b = (a == 0);
11 }
```

c)
```
1  {
2    int a1;
3    int a2;
4    int b ;
5    a1 = 0;
6    a2 = 1;
7    b = a1 + a2;
8    a1 = b + a2;
9    a2 = a1;
10   a1 = b ;
11   b = a2;
12   if (a2 > b) {
13     if (a1 < a2) a1 = -10;
14     else a1 = 10;
15   }
16   else b = 20;
17 }
```

II – Considere que necessita de programar um livro de despesas. Para isso irá ter que definir uma classe Java `ExpenseBook`, cujos objectos representarão esse conceito, e para o que fornecerão um conjunto de operações apropriadas.

Note que neste exercício **não terá** que programar, mas apenas que definir o conjunto de operações (métodos e construtores), assim como as variáveis e constantes que achar necessárias para representar o estado de cada objecto da classe `ExpenseBook`.

Cada livro de despesas acumula o valor das compras efectuadas em cada uma de um conjunto de rubricas pré-definidas ao longo de um mês. Para isso o livro de despesas deverá poder acumular o valor de cada nova despesa efectuada.

As rubricas pré-definidas são as seguintes: alimentação, educação, energia, transportes, lazer.

Em cada momento, deve ser possível consultar o montante já gasto em cada rubrica durante o mês corrente, assim como qual é a rubrica em que já gastou mais dinheiro nesse mesmo mês.

No final do mês, deverá ser possível iniciar um novo mês, começando a acumular novos valores em cada rubrica.

No entanto, para que não se perca informação sobre o valor gasto no passado, o livro de despesas deverá ser capaz de informar o montante total gasto desde que o livro foi criado, sendo que neste caso não será necessário conhecer os valores por rubrica.

De vez em quando (por exemplo no fim do ano), pode ser necessário apagar o montante total gasto, para iniciar uma nova contagem.

COM BASE NA INFORMAÇÃO ACIMA:

Indique variáveis de objecto, métodos e construtores que a classe `ExpenseBook` deverá apresentar, para assegurar as funcionalidades pretendidas.

Para cada variável indique o seu tipo e explique a finalidade.

Para cada método indique o tipo do seu resultado e o tipo dos seus parâmetros (se existirem). Explique ainda a sua finalidade (para que serve) de forma clara e intuitiva.

Indique ainda que métodos são modificadores e que métodos são de consulta (observadores).

III – Considere a seguinte classe `Train` programada em Java.

```
class Train {
  static final String DEFAULTORG = "Lisbon";
  static final int MaxCapacity = 256;
  String org;
  String dst;
  int currentPassengers;

  Train(String destination) {
    org = DEFAULTORG;
    dst = destination;
    currentPassengers = 0;
  }

  void enter(int inPassengers) {
    int overflow;
    overflow = inPassengers+currentPassengers-MaxCapacity;
    if(overflow > 0)
      currentPassengers = MaxCapacity;
    else {
      overflow = 0;
      currentPassengers = currentPassengers + inPassengers;
    }
  }

  void exit(int outPassengers) {
    if (outPassengers > currentPassengers)
      currentPassengers = 0;
    else
      currentPassengers = currentPassengers - outPassengers;
  }

  String getPath() {
    return org+"->"+dst;
  }

  int load() {
    return currentPassengers;
  }
}
```

a) Qual poderá ser o objectivo da classe `Train`? Explique de forma clara a finalidade de cada um dos seus métodos.

b) Considere a seguinte sequência de chamada de métodos a objectos da classe `Train`. Indique, para cada chamada que devolva um resultado, que resultado é esse esse (para o ajudar, as linhas das chamadas estão numeradas).

```
1.  Train t1 = new Train("Evora");
2.  Train t2 = new Train("Porto");
3.  t2.getPath()
4.  t1.enter(200);
5.  t1.exit(20);
6.  t1.enter(90);
7.  t2.enter(15);
8.  t2.load()
9.  t1.load()
10. t2.getPath()
```

IV – Este exercício visa a construção de uma classe Java `Ranking` cujos objectos permitem classificar chegadas de competidores (por exemplo, atletas, carros de F1, bicicletas, etc.) numa meta, e indicar quem deverá subir ao pódio.

O construtor da classe `Ranking` permite definir qual o nome da competição.

Cada objecto da classe `Ranking` possui as seguintes operações:

```
String getCompetitionName()
```

indica qual o nome da competição.

```
void crossed(float time, String competitor)
```

regista a passagem na meta do competidor identificado pela string `competitor` no instante temporal `time`. Este método é chamado por um sistema automático cada vez que um competidor atravessa a meta, sendo o tempo indicado em `time` aquele que o competidor demorou a completar a prova.

IMPORTANTE: Por razões de atraso no envio de dados dos sensores, não é garantido que o método `crossed` seja chamado por ordem de chegada dos competidores. Assume-se ainda que não é possível que dois competidores atravessem a meta no mesmo instante exacto e que todas as provas terão sempre três ou mais competidores.

Os métodos seguintes só serão chamados depois de terminada a prova, ou seja, depois do método `crossed` ser executado para cada competidor que atravesse a meta.

```
int completed()
```

indica o número de competidores que atravessaram a meta.

```
float average()
```

indica o tempo médio que os competidores levaram a finalizar a prova.

```
String goldWinner()
```

indica a identificação do vencedor

```
String silverWinner()
```

indica a identificação do competidor que fez o segundo melhor tempo.

```
String bronzeWinner()
```

indica a identificação do competidor que fez o terceiro melhor tempo.

Apresentamos dois exemplos de utilização da classe `Ranking`:

Exemplo 1)

```
Ranking r = new Ranking("Corrida dos Gafanhotos");
r.crossed(2.3f,"fonseca");
r.crossed(1.2f,"lopes da silva");
r.crossed(1.0f,"zé carlos");
r.crossed(1.3f,"meireles");
r.goldWinner()
"zé carlos"
r.silverWinner()
"lopes da silva"
r.bronzeWinner()
"meireles"
r.completed()
4   (int)
r.average()
1.45(float)
```

Exemplo 2)

```
Ranking runners = new Ranking("100m Final");
runners.crossed(9.97f, "Michael Frater");
runners.crossed(10.03f, "Darvis Patton");
runners.crossed(9.69f, "Usain Bolt");
runners.crossed(9.89f, "Richard Thompson");
runners.crossed(9.91f, "Walter Dix");
runners.crossed(9.95f, "Asafa Powell");
runners.crossed(10.01f, "Marc Burns");
runners.crossed(9.93f, "Churandy Martina");
runners.goldWinner()
"Usain Bolt"   (String)
runners.silverWinner()
"Richard Thompson"   (String)
runners.bronzeWinner()
"Walter Dix"   (String)
runners.completed()
8   (int)
runners.average()
9.9225   (float)
```

Programe, em Java, a classe `Ranking`.

****

****