Instituto Politécnico de Castelo Branco

Escola Superior de Tecnologia.
Curso: Engª Informática + Tecnologias da Informação e Multimédia
2º Semestre do 1º Ano
Unidade Curricular: Programação II
Docente: Fernando Sérgio Barbosa
Ano Letivo 2016/2017
Época Exame
Data: 28/6/2017 Duração: 2h00 sem tolerância

Classificação das questões:

Questões	1	2			3				4			
Alíneas		1	2	3	1	2	3	4	1	2	3.1	3.2
Classificação	11x0,5 (-0,1 errada)	1	1	1,5	1,5	1	1,5	2	1	1,5	1,5	1,5

Nome:_____Número:_____

Curso:_____

1. Das seguintes afirmações escolha **a correta** (as erradas descontam 0.1). RESPONDA NO ENUNCIADO.

1.1. As variáveis de uma classe devem ser sempre private, já os métodos.
- ❑ devem ser sempre public.
- ❑ devem ser sempre protected.
- ❑ são muitas vezes public.
- ❑ nunca devem ser private.

1.2. Uma classe deve representar:
- ❑ um objeto.
- ❑ uma abstração.
- ❑ uma entidade física.
- ❑ uma entidade abstrata.

1.3. A agregação por referência implica que,:
- ❑ ao ser criado o todo, as partes também têm de ser criadas.
- ❑ o objeto contido é sempre o mesmo.
- ❑ a classe todo conhece a classe parte e vice-versa.
- ❑ se pode ir mudando de objetos parte.

1.4. Para dividir um sistema em módulos, deve-se garantir que:
- ❑ os vários módulos tenham pontos em comum.
- ❑ os vários módulos sejam o mais autónomos possível.
- ❑ todos os módulos têm um número semelhante de métodos.
- ❑ cada equipa de desenvolvimento fica com apenas um módulo.

1.5. Dois objetos da mesma classe, com o mesmo estado:
- ❑ é algo impossível de acontecer.
- ❑ reagem da mesma maneira à mesma mensagem.
- ❑ têm a mesma identidade.
- ❑ complementam-se

1.6. Uma classe abstrata:
- ❑ não serve para nada.
- ❑ só tem métodos abstratos.
- ❑ tem de declarar métodos abstratos.
- ❑ não pode ter instâncias.

1.7. O polimorfismo:
- ❑ só funciona com objetos da mesma classe.
- ❑ só é útil para vetores.
- ❑ só é possível se houver herança ou interfaces.
- ❑ só dá trabalho.

1.8. Considere o seguinte código:

```
static void metodoUm() {
   try {
      System.out.println("M1");
      rebenta();
   }
   catch (NullPointerException e) {
      System.out.println("M1.Null");
   }
   catch (NumberFormatException e) {
      System.out.println("M1.Number");
   }
   System.out.println("M1.Fim");
}

public static void main(String[] args) {
   metodoUm();
   metodoDois();
}
```

```
static void metodoDois() {
   try {
      System.out.println("M2");
      rebenta();
      System.out.println("M2.Rebenta");
   }
   catch (NumberFormatException e) {
      System.out.println("M2.Number");
   }
   catch (ArrayIndexOutOfBoundsException e) {
      System.out.println("M2.Array");
   }
   System.out.println("M2.Fim");
}
```

1.8.1. Caso o método rebenta atire a exceção `NullPointerException`, o que aparece na consola é:

- ❑
```
M1
M1.Null
M1.Number
M1.Fim
M2
erro
```
- ❑
```
M1
M1.Null
M1.Fim
M2
erro
```
- ❑
```
M1
M1.Null
M1.Fim
M2
M2.Fim
```
- ❑
```
M1
erro
```

1.8.2. Caso o método rebenta atire a exceção `NumberFormatException`, o que aparece na consola é:

- ❑
```
M1
M1.Number
M1.Fim
erro
```
- ❑
```
M1
M1.Null
M1.Number
M1.Fim
M2
M2.Number
M2.Fim
```
- ❑
```
M1
M1.Number
M1.Fim
M2
M2.Number
M2.Fim
```
- ❑
```
M1
M1.Number
M1.Fim
M2
M2.Number
M2.Rebenta
M2.Fim
```

1.8.3. Caso o método rebenta atire a exceção `ArrayIndexOutOfBoundsException`, o que aparece na consola é:

- ❑
```
M1
erro
```
- ❑
```
M1
M2
erro
```
- ❑
```
M1
M2
M2.Array
M2.Fim
```
- ❑
```
M1
M2.Array
M2.Fim
```

2. Considere a classe Livro de uma aplicação para uma biblioteca. A informação que se pretende armazenar sobre o livro é: o nome do autor, o título, o ano de edição e o número da edição (1ª edição, 2ª edição, etc).

2.1. Declare a classe Livro com as respetivas variáveis, sem esquecer os níveis de acesso corretos.

2.2. Implemente os getters e setters da classe Livro.

2.3. Implemente dois construtores para a classe Livro: um que inicialize todas as variáveis e outro que inicialize a edição como sendo a primeira edição. Escreva o menos código possível.

3. Para elaborar o sistema de empréstimos de livros para a biblioteca elaboraram-se várias classes, das quais se apresenta um excerto:

```
class Leitor {
   private String nome;
   private int numero;
   private Vector<Emprestimo> emprestimos;
}

class Emprestimo {
   private Leitor requisitante;
   private Data quando;
   private Livro emprestado;

   public boolean estaAtrasado( ) { ... }
}
```

```
class Biblioteca {
   private Vector<Leitor> leitores;
   private Vector<Emprestimo> emprestimos;

   public Vector<Leitor> getLeitoresAtrasados() { ... }
   public Vector<Emprestimo> getEmprestimosAtrasados() { ... }
}
```

3.1. Elabore o diagrama de classes (sem variáveis nem métodos), tendo em conta o excerto anterior e sem esquecer as classes Livro e Data (a classe String pode ser ignorada).

3.2. Elabore o método estaAtrasado da classe Emprestimo, sendo o tempo máximo de empréstimo de 5 dias.

3.3. Elabore o método `getEmprestimosAtrasados` da classe Biblioteca, que retorna um vetor com todos os empréstimos que estejam atrasados.

3.4. Elabore o método `getLeitoresAtrasados` da classe Biblioteca, que retorna um vetor com todos os leitores que tenham empréstimos atrasados. Se um leitor tiver mais que um livro em atraso só deve aparecer uma vez.

4. A software house que o/a contratou pretende desenvolver o jogo "HiperMarco". Neste jogo o herói percorre um mundo de plataformas apanhando artefactos e desviando-se de obstáculos. Quer os artefactos, quer os obstáculos podem ser fixos ou móveis. Os desenvolvedores do jogo criaram as primeiras classes para artefactos e obstáculos. As classes criadas foram o caixote, que após alguns toques dados pelo herói no topo ou fundo dão pontuação extra e podem despoletar algum bónus. O elevador, que permite ao herói viajar nele enquanto estiver na sua parte superior. A planta venenosa, que sobe/desce do chão e que só morre se for calcada enquanto está a descer. Se for tocada noutra situação provoca a perda de uma vida. Finalmente o abismo, que se for tocado provoca a perda de uma vida e o recomeçar do nível. O excerto dessas classes é o seguinte:

```
abstract class ElementoDefault implements Elemento {
   private Image img;
   private Mundo mundo;
   private Point posicao;

   ElementoDefault( Image img, Point pos ){ … }

   public void mover( HiperMarco m ){
   }

   public void desenhar( Graphics g ){
      img.draw( g );
   }

   protected boolean bateu(HiperMarco h)       { … }
   protected boolean bateuFundo(HiperMarco h)  { … }
   protected boolean bateuTopo(HiperMarco h)   { … }
   protected boolean bateuEsq(HiperMarco h)    { … }
   protected boolean bateuDir(HiperMarco h)    { … }
   protected Mundo getMundo() { ... }
   public Point getPosicao()  { ... }
}

class Caixote extends ElementoDefault {

   private int nToques = 5;

   Caixote(Image img, Point pos, int nToques ) ( … }

   public void colidiu( HiperMarco h ){
      if( bateuTopo( h ) || bateuFundo( h ) ){
         nToques--;
         getMundo().addScore( 20 );
      }
      if( nToques == 0 )
         getMundo().addScore( 100 );
   }
}

class Abismo extends ElementoDefault {

   public void colidiu( HiperMarco h ){
      if( bateu( h ) ){
         h.perdeVida( 1 );
         getMundo().restartLevel( );
      }
```

```
class Mundo {
   Vector<Abismo> abismos;
   Vector<Caixote> caixotes;
   Vector<PlantaVenenosa> plantas;
   Vector<Elevador> elevadores;
   HiperMarco marco;

   void atualizar(){
      marco.mover();

      for( Abismo a : abismos )
         a.mover( marco );
      for( Caixote c : caixotes )
         c.mover( marco );
      for( Elevador e : elevadores )
         e.mover( marco );
      for( PlantaVenenosa p : plantas )
         p.mover( marco );

      for( Abismo a : abismos )
         a.colidiu( marco );
      for( Caixote c : caixotes )
         c.colidiu( marco );
      for( Elevador e : elevadores )
         e.colidiu( marco );
      for( PlantaVenenosa p : plantas )
         p.colidiu( marco );
   }

   void addElevador(Elevador elevador) {}
   void addPlantaVenenosa(PlantaVenenosa p) { }
   void addAbismo(Abismo abismo) { }
   void addCaixote(Caixote caixote){ }
}

class LeitorNiveis {

   Mundo loadNivel(  int i ){
      Mundo mundo = new Mundo();
      // ...
         // ler os elementos do nível
         Image img = readNextImage();
         Point pos = getNextPosicao();
         char elem = getNextElemento();

         switch( elem ){
```

```
    }
}

class Elevador extends ElementoDefault {
    private int velx, vely;

    Elevador(Image img, Point pos, int vx, int vy) {}

    public void mover( HiperMarco m ){
       getPosicao().translate( velx, vely );
    }

    public void colidiu( HiperMarco h ){
       if( bateuTopo( h )  ){
          h.move( velx, vely);
       }
    }
}
```

```
          case 'e':
            int velx = getNextInt();
            int vely = getNextInt();
            mundo.addElevador( new Elevador( img,
pos, velx, vely) );
            break;
          case 'p':
            mundo.addPlantaVenenosa( new
PlantaVenenosa( img, pos ) );
            break;
          case 'a':
            mundo.addAbismo( new Abismo( img,
pos) );
            break;
          case 'c':
            int nToques = getNextInt();
            mundo.addCaixote( new Caixote( img,
pos, nToques ) );
            break;

          }
       // ...
       return mundo;
    }
}
```

4.1. Implemente a interface Elemento.

4.2. Implemente os construtores da classe ElementoDefault e Caixote.

4.3. Os seus colegas estavam satisfeitos por terem usado a herança e assim poupado imenso código ao criar os elementos. Ao ver este código viu que a herança estava corretamente implementada, mas o polimorfismo estava muito mal usado.

4.3.1. Rescreva a classe Mundo de modo a usar corretamente o polimorfismo

4.3.2. Rescreva a classe LeitorNiveis de modo a usar corretamente o polimorfismo

CONSULTA

Alguns métodos da classe Data que podem ser úteis

`Construtor por defeito`	Inicializa a data com a hora atual
`getDiferencaDias( Data d )`	Retorna a diferença entre duas datas em dias
`getDiferencaMeses( Data d )`	Retorna a diferença entre duas datas em meses
`getDiferencaAnos( Data d )`	Retorna a diferença entre duas datas em anos
`compareTo( Data d )`	Compara duas datas
`somaDias( int dias )`	Soma dias a uma data
`somaMeses( int meses )`	Soma meses a uma data
`somaAnos( int anos )`	Soma anos a uma data