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 Frequência
Data: 2/6/2017 Duração: 2h00 sem tolerância

Classificação das questões:

Questões	1	2	3	4		5				6			
Alíneas				1	2	1	2	3	4	1	2	3	4
Classificação	11x0,4 (-0,1 errada)	1	1	1	1,5	1	1	1,5	1,5	1	1	1,5	1,5

Nome:_____Número:_____

Curso:_____

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

1.1. Pretende-se uma classe Socio, para representar um sócio de um clube, que tenha o número de sócio, a data (dia,/mês/ ano) da inscrição e se tem as cotas em dia ou não.

1.1.1. A classe mais indicada para isso seria:

❑
```
class Socio {
   private int numero;
   private int diaInscricao;
   private int mesInscricao, anoinscricao;
   private int cotasPagas;
}
```

❑
```
class Socio {
   private int numero;
   private Data inscricao;
   private boolean cotasPagas;
}
```

❑
```
class Socio {
   private int numero;
   private String dataInscricao;
   private String cotasPagas;
}
```

❑
```
class Socio {
   private int numero;
   private Data inscricao;
   private int cotasPagas;
}
```

1.1.2. O melhor construtor para esta classe seria:

❑ `public void Socio( int num, int dInsc, int mInsc, int aInsc, int cp )`
❑ `public Socio( int num, int dInsc, int mInsc, int aInsc, int cp )`
❑ `public Socio( int n, Data inscr )`
❑ `public void Socio( int n, Data inscr )`

1.1.3. Para criar um Clube, que tenha vários sócios, a melhor solução seria:

❑
```
class Clube {
   private Vector<Socio> socios;
}
```

❑
```
class Clube {
   private Socio socios;
}
```

❑
```
class Socio {
   private Vector<Socio> socios;
}

class Clube {
   Socio socios;
}
```

❑
```
class Socio {
   private int numero[];
   // um array para restantes variáveis
}

class Clube {
   Socio socios;
}
```

1.1.4. A relação entre Clube e Socio é:

❑ Agregação por referência.
❑ Agregação por valor (composição).
❑ Uso ou dependência.
❑ Herança.

1.2. O import, em java:

❑ só é obrigatório se se usar classes de outro package.
❑ só é obrigatório se se usar classes do mesmo package.
❑ é sempre obrigatório.
❑ não é obrigatório.

1.3. Em Java, os inteiros (int), como parâmetros de entrada em métodos:
- ❑ usam sempre a passagem por cópia.
- ❑ usam sempre a passagem por referência.
- ❑ podem usar a passagem por cópia ou referência.
- ❑ não podem ser usados, só os Integers.

1.4. Se o mudar o nome a uma variável numa classe, obrigar a alterações no código das classes suas clientes, isso significa que:
- ❑ se falhou na abstração.
- ❑ se falhou no encapsulamento.
- ❑ se falhou no polimorfismo.
- ❑ se falhou na modularidade.

1.5. Quando, numa associação, se tem multiplicidade múltipla (*), isso tem implicações nos getters e setters?
- ❑ Não.
- ❑ Sim, porque se devem usar add/remove/get de conteúdos.
- ❑ Sim, porque não se usam getters, só setters.
- ❑ Sim, porque os gettters/setters passam a ter múltiplos parâmetros.

1.6. Considere o seguinte código:

```
interface ExameTeorico {
  int numeroPerguntas = 20;
  void faz( );
  void meteRasteira( );
  void corrige( );
}
```

1.6.1. A linha `int numeroPerguntas = 20;`:
- ❑ define uma variável.
- ❑ define uma constante.
- ❑ origina um erro, pois as interfaces não podem declarar código.
- ❑ origina um erro, pois as inicializações têm de ser feitas num construtor.

1.6.2. Só um dos excertos seguintes está correto, qual?
- ❑ `class Frequencia extends ExameTeorico { ... }`
- ❑ `class Frequencia implements ExameTeorico { ... }`
- ❑ `class Frequencia implements interface ExameTeorico { ... }`
- ❑ `class Frequencia implements class ExameTeorico { ... }`

1.7. Para apanhar as exceções ArrayIndexOutOfBoundsException, NullPointerException e NumberFormatException, por forma a tratar todas do mesmo modo, faz-se:

❑
```
try {
   // ...
}
catch( ArrayIndexOutOfBoundsException )
catch( NullPointerException )
catch( NumberFormatException e ){
   // ...
}
```

❑
```
try {
   // ...
}
catch( Exception e ){
   // ...
}
```

❑
```
try {
   // ...
}
catch( ArrayIndexOutOfBoundsException,
       NullPointerException,
       NumberFormatException e ){
   // ...
}
```

❑
```
try {
   // ...
}
catch( ArrayIndexOutOfBoundsException |
       NullPointerException |
       NumberFormatException e ){
   // ...
}
```

2. Implemente os setters e getters da classe Socio.

3. Escreva um excerto de código em que cria e inicializa os seguintes 3 sócios (assuma que quando um sócio é criado as cotas estão em dia): Sócio número 1, inscrito em 10/1/1903, sócio número 1000, inscrito em 23/11/1950 e sócio número 100000, inscrito em 8/5/2000.

4. O gestor de projeto achou que, na classe Socio, guardar apenas a informação se tem as cotas pagas é muito pouco. Assim mandou remover a variável cotasPagas e, em seu lugar, guardar o último mês e ano de cotas pagas. Assim, a qualquer momento, sabe-se se o sócio tem as cotas pagas, sem andar todos os meses a atualizar a variável cotasPagas de todos os sócios.

   4.1. Reescreva a classe Socio (só as variáveis) de modo a refletir esta alteração.

   4.2. Implemente o método temCotasPagas de acordo com a nova solução. (para saber a data atual basta usar a classe Data e cria um objeto desta classe com o construtor por defeito). Assume-se que as cotas estão pagas até ao mês seguinte do último mês em que pagou. Exemplo, se pagou em maio de 2017, só terá cotas por pagar em julho de 2017, já que durante junho ainda pode pagar as cotas desse mês.

5. O clube gostou da aplicação dos sócios e pediu um novo módulo: a criação de uma comissão eleitoral, para gerir o processo de eleição do novo presidente. Essa comissão é composta por 10 membros, que deverão ter mais de 10 anos de sócio. Os candidatos a presidente devem ser sócios há mais de 20 anos. Quer os membros quer os candidatos devem ter as cotas em dia. Para votar cada sócio dispõe de vários votos consoante a sua antiguidade. A tabela de votos é a seguinte:

Anos de sócio	até 5 anos	até 10 anos	até 15 anos	Até 25 anos	Mais de 25 anos
Número de votos	1	3	6	12	24

   5.1. Crie uma exceção SocioNaoConformeException, que será usada nas alíneas seguintes em situações de erro. Essa exceção deve ser do tipo checked. Defina apenas o cabeçalho da exceção.

   5.2. Elabore o método getNumVotos(Socio s), que indica a quantos votos tem direito o sócio. Se não tiver as cotas em dia, tem direito a zero votos.

   5.3. Elabore o método addMembro(Socio s) que associe um membro à comissão eleitoral.

   5.4. Elabore o método addCandidato(Socio s) que regista a candidatura de um sócio a presidente.

6. 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:

```
class Elevador {
  private Image img;
  private Mundo mundo;
  private Point posicao;

  private int velx, vely; // velocidade

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

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

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

  private boolean bateu(HiperMarco h) {}
  private boolean bateuFundo(HiperMarco h) {}
  private boolean bateuTopo(HiperMarco h) {}
  private boolean bateuEsq(HiperMarco h) { }
  private boolean bateuDir(HiperMarco h) { }
```

```
class Abismo {
  private Image img;
  private Mundo mundo;
  private Point posicao;

  void mover( HiperMarco m ){
    // é estático, não faz nada
  }

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

  void colidiu( HiperMarco h ){
    if( bateu( h ) ){
       h.perdeVida( 1 );
       mundo.restartLevel( );
    }
  }

  private boolean bateu(HiperMarco h) {}
  private boolean bateuFundo(HiperMarco h) {}
  private boolean bateuTopo(HiperMarco h) {}
  private boolean bateuEsq(HiperMarco h) { }
  private boolean bateuDir(HiperMarco h) { }
}
```

```
}

class Caixote {
  private Image img;
  private Mundo mundo;
  private Point posicao;

  private int nToques = 5;

  void mover( HiperMarco m ){
    // é estático, não faz nada
  }

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

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

  private boolean bateu(HiperMarco h) {}
  private boolean bateuFundo(HiperMarco h) {}
  private boolean bateuTopo(HiperMarco h) {}
  private boolean bateuEsq(HiperMarco h) { }
  private boolean bateuDir(HiperMarco h) { }
}
```

```
class PlantaVenenosa {
  private Image img;
  private Mundo mundo;
  private Point posicao;

  private boolean subir; // subir ou descer?

  void mover( HiperMarco m ){
    if( subir )
       posicao.translate( 0, -5 );
    else
       posicao.translate( 0, 5 );
    // ...
  }

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

  void colidiu( HiperMarco h ){
    if( bateu( h ) && subir  ){
       h.perdeVida( 1 );
    }
    else if( bateuTopo( h ) && !subir ){
       dye();
    }
  }

  private void dye() { }

  private boolean bateu(HiperMarco h) {}
  private boolean bateuFundo(HiperMarco h) {}
  private boolean bateuTopo(HiperMarco h) {}
  private boolean bateuEsq(HiperMarco h) { }
  private boolean bateuDir(HiperMarco h) { }
}
```

Ao ver este código sugeriu o uso de herança e polimorfismo. Depois de explicar a sua solução, os seus colegas ganharam um enorme respeito pelas suas capacidades de programador. Considere que os métodos bateu têm implementação igual em todas as classes.

6.1. Desenhe o diagrama de classes da hierarquia de herança (e só desta) com todos os métodos e variáveis referidos no código. Não se esqueça da interface.

6.2. Implemente a interface.

6.3. Implemente a superclasse

6.4. Rescreva a classe Elevador após a aplicação da herança.

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