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 2015/2016
Época Frequência
Data: 3/6/2016 Duração: 2h00 sem tolerância

Classificação das questões:

Questões	1	2	3	4	5			6			
Alíneas					1	2	3	1	2	3	4
Classificação	13x0,4 (-0,1 errada)	1,5	1,5	1,5	1,5	1	2	1,2	1,2	2,4	1

Nome:_____Número:_____

Curso:_____

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

1.1. Considere o seguinte excerto da classe Coordenada:

```
class Coordenada {
    double latitude;
    double longitude;

    double raioTerra = 6372.795477598;

    Coordenada(double latitude, double longitude) { ... }

    double distanciaKm( Coordenada c2 ){
        // como as coordenadas estão em graus é preciso converter para radianos
        double lat1 = Math.PI * latitude / 180;
        double lat2 = Math.PI * c2.latitude / 180;
        double long1 = Math.PI * longitude / 180;
        double long2 = Math.PI * c2.longitude / 180;

        double cosenos = Math.cos( lat1)*Math.cos( lat2 );
        double senos = Math.sin( lat1)*Math.sin( lat2 );
        double difLong = Math.cos( long1 - long2 );
        return raioTerra * Math.acos( senos + cosenos * difLong );
    }

    // getters e setters
}
```

1.1.1. O método `distanciaKm( )` da classe Coordenada deve ter o nível de acesso:

- ❑ `static.`
- ❑ `public.`
- ❑ `private.`
- ❑ `final.`

1.1.2. Para declarar `raioTerra` como sendo uma constante usa-se a declaração:

- ❑ `double raioTerra = 6372.795477598 final;`
- ❑ `final double raioTerra = 6372.795477598;`
- ❑ `const double raioTerra = 6372.795477598;`
- ❑ `static double raioTerra = 6372.795477598;`

1.1.3. Para declarar `raioTerra` como sendo uma variável de classe usa-se a declaração:

- ❑ `double raioTerra = 6372.795477598 class;`
- ❑ `final double raioTerra = 6372.795477598;`
- ❑ `const double raioTerra = 6372.795477598;`
- ❑ `static double raioTerra = 6372.795477598;`

1.2. Considere o seguinte excerto da classe PointOfInterest:

```
package p2.gps;

class PointOfInterest {
    private String nome;
    private Coordenada coord;
    private ArrayList<String> categorias = new ArrayList<String>();
    private boolean favorito;

    public PointOfInterest(String nome, Coordenada coord, boolean favorito) {...}
    public PointOfInterest(String nome, Coordenada coord ) { ... }

    public boolean temCategoria( String categoria ){ ... }
    // getters e setters
}
```

1.2.1. Qual a relação entre a classe PointOfInterest e a classe Coordenada?

❑ Herança ❑ Uso ❑ Agregação ❑ Associação

1.2.2. A implementação correta do método temCategoria seria:

❑ `return categorias.contains( categoria );`

❑ `return categorias.equals( categoria );`

❑ `return categoria.equals( categorias );`

❑ `return categorias == categoria;`

1.2.3. Para usar esta classe num programa, qual dos seguintes códigos NÃO funciona?:

❑
```
import p2.gps.*;
// ...
PointOfInterest poi;
```

❑
```
import p2.gps;
// ...
PointOfInterest poi;
```

❑
```
import p2.gps.PointOfInterest;
// ...
PointOfInterest poi;
```

❑
```
// ...
p2.gps.PointOfInterest poi;
```

1.3. O overload de métodos permite que:

❑ duas classes tenham métodos com o mesmo nome.
❑ dois métodos, da mesma classe, tenham o mesmo nome.
❑ a subclasse possa redefinir métodos da superclasse.
❑ uma classe abstrata tenha métodos sem implementação.

1.4. Segundo o conceito de abstração:

❑ uma classe deve ter sempre em conta o que os clientes pretendem dela.
❑ uma classe deve ter sempre em atenção que a sua implementação pode mudar.
❑ uma classe deve pertencer sempre a um package.
❑ uma classe abstrata tem métodos sem implementação.

1.5. Quando se deve declarar uma variável como protected?

❑ Só em superclasses, para que as subclasses lhe possam aceder.
❑ Sempre, porque a classe pode vir a ter subclasses em qualquer altura.
❑ Só em superclasses e quando a variável for um contentor de elementos (Vetor, ArrayList, etc) .
❑ Nunca.

1.6. Considere o seguinte código

```
1  static int metodoUm( ){
2     int x = 10;
3     try {
4        rebenta();
5     }
6     catch( NullPointerException e){
7        x = 30;
8     }
9     catch( InvalidArgumentException e ){
10       x = 40;
11    }
12    return x;
13 }
14 public static void main( String []args ){
15    int res = metodoDois();
16    System.out.println("res = " + res );
17 }
```

```
18 static int metodoDois(){
19    int x = 100;
20    try {
21       x = metodoUm();
22    }
23    catch( NullPointerException e){
24       x = 120;
25    }
26    catch( NumberFormatException e ){
27       x = 130;
28    }
29    return x;
30 }
31
```

1.6.1. Considerando que o método rebenta() atira a exceção NumberFormatException o código imprime:
- ❑ res = 10.
- ❑ res = 130.
- ❑ res = 100.
- ❑ uma mensagem de erro.

1.6.2. Considerando que o método rebenta() atira a exceção NullPointerException o código imprime:
- ❑ res = 100.
- ❑ res = 120.
- ❑ res = 30.
- ❑ uma mensagem de erro.

1.6.3. Considerando que o método rebenta() atira a exceção MateriaPorEstudarException o código imprime:
- ❑ res = 100.
- ❑ res = 10.
- ❑ res = 40.
- ❑ uma mensagem de erro.

1.6.4. Considerando que o método rebenta() atira a exceção InvalidArgumentException o código imprime:
- ❑ res = 100.
- ❑ res = 10.
- ❑ res = 40.
- ❑ uma mensagem de erro.

2. Implemente os setters e getters da classe Coordenada, tendo em conta que a longitude tem de ser um valor entre -180 e 180 e a latitude um valor entre -90 e 90. Deve usar a exceção IllegalArgumentException.

3. Implemente os construtores da classe PointOfInterest, tendo em conta que, por defeito, o ponto de interesse não deve ser catalogado como favorito. Reutilize o máximo de código possível.

4. Escreva um excerto de código em que cria e inicializa os seguintes pontos de interesse:

Nome	latitude	longitude	favorito	Categorias
EST	50.234	20.542	sim	Escola, Monumento
Estádio do Dragão	50.124	21.005	sim	Desporto, Concertos
Festival Paredes de Coura	50.115	21.948	sim	Paisagem, Campismo, Concertos

5. Considere as seguintes classes, as quais fazem parte de um sistema de GPS, bem com as já mencionadas classes Coordenada e PointOfInterest. Este sistema representa um sistema de GPS com capacidade de navegação. O sistema é constituído por um GPSReader, responsável por ler os sinais de gps e um mapa de estradas. O mapa, além de conter as estradas também tem uma lista de pontos de interesse. Cada ponto de interesse deverá estar associado a, pelo menos, uma categoria (Hotel, Restaurante, Farmácia, Centro Comercial, ect). Como já se viu (pergunta 4) um POI pode ser classificado em várias categorias. Para melhor pesquisar os POI pelas categorias, o mapa tem um hashmap que armazena os pontos de interesse de acordo com as categorias possíveis. Para cada categoria indexada, o hashmap armazena um vetor com todos os POI que tenham essa categoria.

```
class Navegador {
   GPSReader gps = new GPSReader();
   Mapa mapa;

   public void irPara( Coordenada c ){ }
   public void setMapa(Mapa mapa) { ... }
   public Mapa getMapa() { ... }
}

class GPSReader {

   public Coordenada getLocalizacaoAtual(){ return null; }
}
```

```
class Mapa {
   String nome;
   Vector<Estrada> estradas = new Vector<Estrada>();
   Map<String,Vector<PointOfInterest>> pois = new HashMap<String,Vector<PointOfInterest>>();

   Estrada getEstradaMaisProxima( Coordenada c ){ ... }

   ArrayList<PointOfInterest> getPOIsRaio( Coordenada c, double raio ){ ... }
   ArrayList<PointOfInterest> getPOIsRaio( Coordenada c, String categoria, double raio ){...}
   ArrayList<PointOfInterest> getPOIsRaio( Coordenada c, String categorias[], double raio ){
       ...
   }
}

class Estrada {
   // ...
}
```

5.1. Elabore o diagrama de classes deste sistema, sem esquecer as classes Coordenada e PointOfInterest. Não coloque variáveis nem métodos.

5.2. Complete o seguinte método, que ilustra como se adiciona um POI ao mapa. Para isso deve preencher cada espaço com **uma única** palavra. RESPONDA NO ENUNCIADO.

```
public void addPOI(_____________________ poi ){
    for( _____________________ cat : poi.getCategorias() ){
        // ver qual o vetor de pois associado à categoria
        Vector<PointOfInterest> v = pois.get( cat );
        // se o vetor ainda não existir é porque
        // a categoria ainda não foi usada
        if( v == _____________________ ){
            v = new Vector<PointOfInterest>();
            pois.put( cat,  v );
        }
        v._____________________( poi );
    }
}
```

5.3. Implemente o método `ArrayList<PointOfInterest> getPOIsRaio( Coordenada c, String categoria, double raio )` da classe Mapa que retorna um ArrayList dos POI que pertençam a uma dada categoria e que estejam a menos de `raio` kilometros da coordenada `c`.

6. Considere as seguintes classes.

```
class UsaEstruturas {

    private Estrutura estruts[];

    void criaEstruturas( ){
       estruts = new Estrutura[ 5 ];
       estruts[0] = new Estrutura( Estrutura.ESTRUT_SOLIDA, 200, 3 );
       estruts[1] = new Estrutura( Estrutura.ESTRUT_SOLIDA, 300, 4 );
       estruts[2] = new Estrutura( Estrutura.ESTRUT_MOVEL, 10, 5 );
       estruts[3] = new Estrutura( Estrutura.ESTRUT_LEVE, 2, 6 );
       estruts[4] = new Estrutura( Estrutura.ESTRUT_FRAGIL, 100, 4 );
    }

    void processaEstruturas( int x ){
       for( Estrutura est : estruts )
          est.fazerIsto( x );
    }
}
```

```
class Estrutura {

    private int fator;
    private int fixante;
    private int gama;
    private int tipo;

    public Estrutura(int tipo, int fator, int gama) {
       this.fator = fator;
       this.tipo = tipo;
       this.gama = gama;
       // se for do tipo ESTRUT_SOLIDA também tem
       // de inicializar o campo fixante (as outras não precisam deste campo)
       if( tipo == ESTRUT_SOLIDA )
          fixante = gama*fator;
    }

    void fazerIsto( int x ){
       switch( tipo ){
       case ESTRUT_SOLIDA:
          fator += x;
          fixante = fator / x;
          break;
       case ESTRUT_FRAGIL:
          fator -= x;
          break;
       case ESTRUT_LEVE:
          fator /= x;
          break;
       case ESTRUT_MOVEL:
          fator += (x+3);
          break;
       }
    }

    void fazerAquilo( int x ){
       if( tipo == ESTRUT_SOLIDA ){
          fixante = x;
          fator = x < fator? x: fator+x;
       }
       else {
          fator = x + 12;
       }
    }

    void setGama( int g ){
       gama = g > gama? g-fator: g;
    }
}
```

Os programadores deste código estavam particularmente satisfeitos pela solução arranjada, pois poderiam acrescentar mais tipos de estruturas sem nunca alterarem a classe UsaEstruturas (exceto o método criaEstruturas, claro).
Claro que, quando viu este código, achou que podia fazer muito melhor, pois este encontra-se cheio de ifs e switchs. Para isso apresentou a proposta de usar a herança e polimorfismo. O seu argumento é que nunca teria de mexer na classe UsaEstruturas (exceto, claro, o método criaEstruturas) e melhor, também não precisava de alterar a classe Estrutura.

6.1. Como todas as boas hierarquias de herança a sua também começa com uma interface. Implemente-a.

6.2. Implemente a superclasse da sua hierarquia.

6.3. Implemente as subclasses da sua hierarquia que lidariam com as estruturas sólidas e móveis. Para simplificar assuma que as variáveis da superclasse são protected.

6.4. Altere o método criaEstruturas da classe UsaEstruturas para que ele use a sua hierarquia.