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 Recurso
Data: 17/7/2017 Duração: 2h00 sem tolerância

Classificação das questões:

Questões	1	2			3			4				
Alíneas		1	2	3	1.1	1.2	2	1	2	3	4	5
Classificação	10x0,5 (-0,1 errada)	1	1,3	1,2	2	1,5	2	1	1	1,5	1	1,5

Nome:_____Número:_____

Curso:_____

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

1.1. Uma variável static:

- ❑ só pode ser acedida por métodos static.
- ❑ pode ser acedida por qualquer método da classe.
- ❑ não pode mudar de valor, isto é, é uma constante.
- ❑ só pode ser declarada nas superclasses.

1.2. Considere o seguinte código:

```
public class ClasseA {

   private int x;
   public int y;

   void mA( int x )           { /* método (1) */ }
   void mA( int x, String s ) { /* método (2) */ }
   void mA( )                 { /* método (3) */ }
   void mA( String s )        { /* método (4) */ }
   void mA( int y )           { /* método (5) */ }

   private class ClasseB {
      private int x;
      private int y;
   }
}
```

1.2.1. A ClasseB em relação à classe ClasseA é uma:

- ❑ subclasse.
- ❑ classe interna.
- ❑ classe externa.
- ❑ classe agrupada.

1.2.2. A ClasseB pode aceder à variável x da ClasseA?

- ❑ Não, porque x é privada de ClasseA.
- ❑ Não, porque ClasseB também tem uma variável x.
- ❑ Não, porque ClasseB é uma classe privada, logo não tem acesso às variáveis de ClasseA.
- ❑ Sim.

1.2.3. Para a ClasseB aceder à variável y da ClasseA pode-se fazer:

- ❑ Nada, porque simplesmente não pode aceder.
- ❑ `this.y = 10;`
- ❑ `ClasseA.this.y = 10;`
- ❑ `this.ClasseA.y = 10;`

1.2.4. Um dos métodos da ClasseA provoca erros de compilação. Qual?

❑ O método (2). ❑ O método (3). ❑ O método (4). ❑ O método (5)

1.3. Separando uma classe em interface (métodos) e implementação, pode-se dizer que:
- ❑ a interface deve seguir o princípio da abstração e a implementação o princípio do encapsulamento.
- ❑ a interface deve seguir o princípio do encapsulamento e a implementação o princípio da abstração.
- ❑ quer a interface quer a implementação devem seguir o princípio da abstração.
- ❑ a interface deve seguir o princípio da abstração e a implementação o princípio do polimorfismo.

1.4. Uma das vantagens da hierarquia "faz parte de " é:
- ❑ poder usar o polimorfismo.
- ❑ poder construir classes mais complexas com base em classes mais simples.
- ❑ poder conhecer outros elementos da hierarquia a partir de um só elemento.
- ❑ ser mais fácil implementar as subclasses.

1.5. A relação que melhor representa a ligação entre Espada, Pistola e Granada é:
- ❑ Tancos.
- ❑ Herança.
- ❑ Associação.
- ❑ Agregação.

1.6. Qual a diferença entre throw e throws?
- ❑ O throw atira uma exceção e o throws informa que um método pode atirar uma exceção.
- ❑ O throws atira uma exceção e o throw informa que um método pode atirar uma exceção.
- ❑ O throw atira uma exceção e o throws reatira essa exceção depois de apanhada.
- ❑ O throws atira uma exceção e o throw reatira essa exceção depois de apanhada.

1.7. Um bloco try:
- ❑ tem de ter um catch.
- ❑ tem de ter um finally.
- ❑ tem de ter um catch ou um finally.
- ❑ pode ter mais de um finally.

2. Considere a informação sobre uma estação de rádio, nomeadamente o seu nome e frequência. A frequência tem de ser um valor entre 87,5 e 108.0 Mega Hertz (MHz).

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

2.2. Implemente os getters e setters da classe EstacaoRadio. Nos casos de erro, deve ser usada a exceção InvalidArgumentException que é subclasse de RuntimeException.

2.3. Implemente dois construtores para a classe EstacaoRadio: um que inicialize todas as variáveis e um por defeito que inicializa o nome a "Rádio" e a frequência a 87,5, usando o menos código possível.

3. Pretende-se implementar uma aplicação de rádio para dispositivos móveis e que suporte RDS.

3.1. Começou-se por atualizou a classe EstacaoRadio de modo a que esta usa-se um Sintonizador e armazena-se um conjunto de frequências alternativas. Para sintonizar uma estação deve-se começar por sintonizar a frequência base. Se a força do sinal for menor que 80, deve-se então tentar as frequências alternativas até que uma tenha um sinal com força maior que 80. O esboço da classe ficou:

```
class EstacaoRadio {
   RadioTuner tuner;
   // ...

   void sintonizar( ){
      tuner.setFrequencia( getFrequenciaBase() );
      if( tuner.getForcaSinal() < 80 ){
         /*
            código a completar
         */
      }
   }
}
```

3.1.1. Altere a classe de modo a suportar as frequências alternativas, declarando as variáveis necessárias e os métodos addFrequenciaAlternativa e removeFrequenciaAlternativa.

3.1.2. Complete o método sintonizar.

3.2. A aplicação, RadioApp, deve ter uma lista de estações de rádio, e o esboço dela ficou o seguinte:

```
class RadioApp {
   Vector<EstacaoRadio> radios;
   int radioAtual = 0;

   void addEstacaoRadio( EstacaoRadio r ) { ... }
   void removeEstacaoRadio( EstacaoRadio r ) { ... }

   void poeTocar( int index ){
      EstacaoRadio est = radios.get( index );
      est.sintonizar();
      radioAtual = índex;
   }

   String[] getNomesEstacoes(){
      String nomes[] = new String[ radios.size() ];
      for( int i=0; i < radios.size(); i++ )
         nomes[i] = radios.get(i).getNome();
      return nomes;
   }
}
```

Implemente os métodos `proximaRadio()` e `radioAnterior()` que sintonizam a rádio seguinte e a anterior da lista de rádios, respetivamente. Quando chega a uma das extremidades deve recomeçar da outra (a próxima da última é a primeira).

4. Para melhorar a aplicação resolveu-se dar suporte também a rádios web. Dentro destas, existem as rádios livres e as que é necessário terem subscrição. O excerto delas é o seguinte:

```
class WebRadio {
   String nome;
   String url;
   WebService web;

   void sintonizar( ){
      web.connect( url );
      web.startDecodingSound();
   }
}

class AuthWebRadio {
   String nome;
   String url;
   String username;
   String pass;
   WebService web;

   void sintonizar( ){
      web.connect( url );
      web.authenticate( username, pass );
      web.startDecodingSound();
   }
}
```

Ao analisar o código, disse que deveria ser usada a herança de modo a otimizar a aplicação e evitar a repetição de código.

4.1. Elabore o diagrama de classes da sua solução não esquecendo a classe RadioApp e uma interface no topo da hierarquia de herança.

4.2. Implemente a interface.

4.3. Implemente a superclasse.

4.4. Altere a classe AuthWebRadio de acordo com a sua solução.

4.5. Altere a classe RadioApp de acordo com a sua solução.