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 Exame
Data: 4/7/2016 Duração: 2h00 sem tolerância

Classificação das questões:

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

Nome:_____Número:_____

Curso:_____

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

1.1. O nível de acesso por defeito, permite o acesso:
- ❑ só às subclasses do mesmo package.
- ❑ só às subclasses, seja qual for o package.
- ❑ a todas as classes do mesmo package.
- ❑ a todas as classes do mesmo package e a subclasses de outros packages.

1.2. Em Java, as variáveis de uma classe:
- ❑ podem ser inicializadas na declaração.
- ❑ devem ter, no seu nome, o nome da classe a que pertencem.
- ❑ só podem ser inicializadas no construtor.
- ❑ têm de ser inicializadas nos setters.

1.3. Os getters:
- ❑ permitem o acesso de escrita às variáveis de uma classe.
- ❑ permitem o acesso de leitura às variáveis de uma classe.
- ❑ devem validar sempre os dados.
- ❑ por norma têm como tipo de retorno o void.

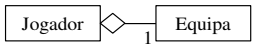
1.4. Um método static:
- ❑ só pode aceder a variáveis static.
- ❑ tem de ser público.
- ❑ não pode atirar exceções.
- ❑ não pode ser redefinido.

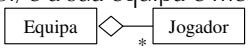
1.5. Em Java, quando se passa um objeto como parâmetro:
- ❑ usa-se a passagem por cópia.
- ❑ usa-se a passagem por referência.
- ❑ usa-se a passagem por pointer.
- ❑ usa-se a passagem por clone.

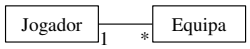
1.6. Ao distribuir as classes pelos packages, deve-se ter em conta o princípio:
- ❑ da abstração.
- ❑ do encapsulamento.
- ❑ da modularidade.
- ❑ da hierarquia.

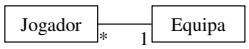
1.7. A composição é um caso especial:
- ❑ da associação.
- ❑ da dependência.
- ❑ da agregação.
- ❑ da herança.

1.8. A relação entre um jogador (de futebol) e a sua equipa é melhor representada por:

❑ 

❑ 

❑ 

❑ 

1.9. O polimorfismo é mais útil:

- ❑ para os clientes de uma hierarquia de herança.
- ❑ para quem cria novas classes de uma hierarquia de herança.
- ❑ quando se usam enumerações.
- ❑ para complicar o sistema.

1.10. Um método final:

- ❑ só pode aceder a variáveis final.
- ❑ tem de ser público.
- ❑ não pode atirar exceções.
- ❑ não pode ser redefinido.

1.11. 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       rebenta();
23    }
24    catch( NullPointerException e){
25       x = 120;
26    }
27    catch( NumberFormatException e ){
28       x = 130;
29    }
30    return x;
31 }
```

1.11.1. Considerando que o método rebenta() atira a exceção NumberFormatException o código imprime:

- ❑ res = 10.
- ❑ res = 130.
- ❑ res = 100.
- ❑ uma mensagem de erro.

1.11.2. Considerando que o método rebenta() atira a exceção NullPointerException o código imprime:

- ❑ res = 100.
- ❑ res = 120.
- ❑ res = 30.
- ❑ uma mensagem de erro.

1.11.3. Considerando que o método rebenta() atira a exceção InvalidArgumentException o código imprime:

- ❑ res = 100.
- ❑ res = 10.
- ❑ res = 40.
- ❑ uma mensagem de erro.

2. Considere uma cor no formato RGB. Essa cor tem 3 campos: vermelho (Red), verde (Green) e azul (Blue). Exemplo de uma cor: (100,90,110), onde Red = 100, Green = 90 e Blue = 110. Todos os valores são inteiros na gama [0, 255].

2.1. Implemente a classe CorRgb, declarando as variáveis e os getters e setters. Deve usar a exceção IllegalArgumentException.

2.2. Implemente dois construtores, um que inicializa todos os valores a 0 e outro com valores definidos pelo cliente. Utilize o menos código possível.

2.3. Crie o método eEscalaCinzento, que indica se a cor é um cinzento (todas as componentes iguais).

3. Considere as seguintes classes, as quais fazem parte de uma agenda eletrónica que armazena informação sobre eventos. A agenda também suporta lembretes (avisos de que um evento está próximo).

```
class Evento {
   String descricao;
   Tempo quando;
   String onde;
   Vector<Lembrete> lembretes = new Vector<Lembrete>();

   public String getDescricao() {
      return descricao;
   }
   public Tempo getQuando() {
      return quando;
   }
   public String getOnde() {
      return onde;
   }
}

class Lembrete {
   Evento evento;
   Tempo quando;
   boolean ativo;
   int tipo;

   public Tempo getQuando() {
      return quando;
   }
   public boolean estaAtivo() {
      return ativo;
   }
   public void setAtivo(boolean ativo) {
      this.ativo = ativo;
   }

   public void despoletar() { ... }
   private String comporMensagem(){ ... }
}

class Agenda {
   ArrayList<Evento> eventos = new ArrayList<Evento>();
   ArrayList<Lembrete> lembretes = new ArrayList<Lembrete>();

   void addEvento( Evento e ){ ...}
   void despoletarLembretes( Tempo agora ){ ... }
}
```

3.1. Elabore o diagrama de classes deste sistema, sem esquecer a classe Tempo. Não coloque variáveis nem métodos.

3.2. Complete o seguinte método, que ilustra como se adiciona um evento à agenda, mantendo-os ordenados por tempos. Para isso deve preencher cada espaço com **uma única** palavra. RESPONDA NO ENUNCIADO.

```
void addEvento( ____________________ e ){
  int idx = 0;
  for( ; idx < ____________________.size(); idx++ ){
     Evento ei = eventos.get( idx );
     if( ei.____________________.compareTo( e.____________________) >= 0 )
       break;
  }
  eventos.add( idx, ________________ );
}
```

3.3. Implemente o método `void despoletarLembretes( Tempo agora )` da classe Agenda que percorre todos os lembretes e despoleta todos cujo tempo já tenha passado e que ainda estejam ativos. Depois de despoletar um lembrete deve desativar esse lembrete.

4. Considere a classe Lembrete, agora mais pormenorizada.

```
class Lembrete {
   private Evento evento;
   private Tempo quando;
   private boolean ativo;
   private int tipo;

   private String telefone;
   private String email;

   public Tempo getQuando() {
      return quando;
   }
   public boolean estaAtivo() {
      return ativo;
   }
   public void setAtivo(boolean ativo) {
      this.ativo = ativo;
   }
   public void despoletar() {
      switch( tipo ){
      case LEMB_EMAIL:
         enviarEmail();
         break;
      case LEMB_NOTIFICACAO:
         enviarNotificao();
         break;
      case LEMB_ALARME:
         tocarAlarme();
         break;
      case LEMB_CHAMADA:
         fazerChamada();
         break;
      }
   }
   private String comporMensagem(){
      return "Não esquecer " + evento.getDescricao() + " no " + evento.getOnde() +
             " em " + evento.getQuando();
   }
   private void fazerChamada() {
      makeCall( telefone );
      speak( comporMensagemn() );
      disconnect( );
   }
   private void tocarAlarme() {
      playAlarm( );
   }
   private void enviarNotificao() {
      sendNotification( comporMensagem() );
   }
   private void enviarEmail() {
      sendEmail( email, "Lembrete", comporMensagem() );
   }
   private void sendEmail(String email, String assunto, String texto) { ... }
   private void disconnect() { ... }
   private void speak(String msg) { ... }
   private void makeCall(String telefone) { ... }
   private void playAlarm() { ... }
   private void sendNotification( String msg ) { ... }
}
```

Os programadores deste código estavam particularmente satisfeitos pela solução arranjada, pois poderiam acrescentar mais tipos de lembretes bastando alterar a classe Lembrete.
Claro que, quando viu este código, achou que podia fazer muito melhor, usando a herança.

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

4.2. Implemente a superclasse da sua hierarquia. Especifique todos os níveis de acesso.

4.3. Implemente as subclasses da sua hierarquia que lidariam com os lembretes que usam email e notificações. Especifique todos os níveis de acesso.