INSTITUTO SUPERIOR TÉCNICO

Versão: 1234114234

Teste de Programação com Objectos
LCI, LEIC, LERCI
Prof. responsável: David Martins de Matos (Alameda)
Prof. responsável: João Dias Pereira (TagusPark)
Ano lectivo 2005/2006 – 1º Semestre
20 de Dezembro de 2005

Escreva já o seu número em **todas** as folhas da prova.

O teste está dividido em duas partes. A primeira parte tem a cotação máxima de 7,5 valores e contém 15 perguntas de resposta múltipla (cinco respostas possíveis), devendo assinalar na tabela da segunda folha deste teste a resposta correcta (só há uma resposta correcta) para cada pergunta. Para efeitos classificativos só se considerarão as respostas assinaladas nesta tabela.

Se se enganar, risque a resposta dada e escreva à frente a nova resposta. Se deixar duas respostas, mesmo que uma delas seja a correcta considerar-se-á uma pergunta não respondida (ausência de resposta).

A segunda parte tem a cotação máxima de 12,5 valores e contém 6 perguntas abertas, sendo aconselhável limitar o tamanho das respostas ao espaço disponível. O teste tem a duração de 2:15 horas.

Durante esta prova não há esclarecimento de dúvidas. Se detectar algum erro, que tenha influência na resolução de um exercício, assinale-o. Se tiver razão ser-lhe-á atribuída a totalidade da cotação dessa pergunta.

Boa sorte!

Aluno Número: : | | | | | |

Nome: _____

Licenciatura: _____

Respostas certas: : | |

Respostas erradas: : | |

Perguntas. Abertas: : | |

Classificação final:

1ª PARTE

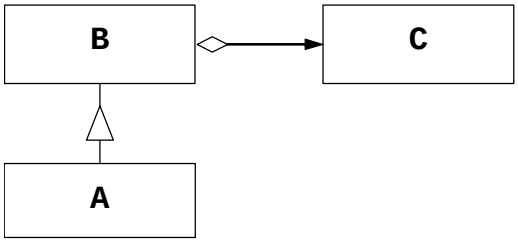
(cada resposta certa=+0,5; cada resposta errada=-0,125; ausência de resposta=0,0)

RESPOSTAS:

PERGUNTA	Resposta escolhida
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	
14	
15	

PERGUNTAS:

1. Qual das seguintes afirmações está correcta, tendo em consideração o diagrama UML:



A C é uma implementação de B

B A é uma implementação de B

C C é uma classe derivada de B

D B possui (pelo menos 1) instâncias de C

E C é uma agregação de B

2. Em Java, qual das seguintes afirmações é verdadeira?

A existem classes sem construtores

B todas as classes têm, pelo menos, um construtor

C todas as classes têm, pelo menos, o construtor sem argumentos definido

D cada classe tem de ter, pelo menos, um construtor público

E todas as anteriores

3. Em Java, e acerca de classes abstractas e interfaces, qual é a afirmação verdadeira?

A não há diferenças

B as classes abstractas não podem ter atributos “final”

C as interfaces não podem ter atributos “final”

D pode-se herdar de várias classes abstractas, mas só se pode concretizar uma interface

E todas são falsas

4. Considerando a linguagem de programação Java, qual das seguintes frases está correcta?

A uma classe abstracta não pode ter atributos

B uma classe abstracta não pode ter construtores “public”

C uma classe abstracta não pode ter construtores “protected”

D uma classe abstracta não pode ter métodos “final”

E uma classe abstracta não pode ser “final”

5. Qual das seguintes afirmações é verdadeira em Java?

A é possível modificar o tamanho de um vector

B vectores de tipos primitivos não são reciclados automaticamente

C pode-se alterar a dimensão de um vector de objectos

D não é possível aceder a um vector fora dos seus limites

E atributos de objectos que sejam vectores de objectos não podem ser “public”

6. Em Java, qual das seguintes frases é verdadeira, relativamente ao uso da palavra reservada “final”:

A indica que um método não pode ser sobrecarregado (overloaded)

B indica que um método não pode ser substituído (overriden)

C indica que um método não pode ser “protected” nem *friendly*

D só podem existir dois métodos “final” por classe

E esta palavra reservada só se pode aplicar a atributos e a métodos

7. Qual das seguintes opções completa de forma correcta a seguinte frase: Em Java, os construtores...

A não podem lançar excepções

B não podem alterar os valores dos atributos do objecto a criar

C podem chamar outros construtores da mesma classe

D não têm acesso aos atributos da classe base

E nunca podem ser privados

8. Qual das seguintes afirmações é verdadeira em Java?

A as classes não são representáveis como objectos

B dado um objecto de uma classe não especificada, não é possível saber que métodos são invocáveis sobre ele

C existe uma instância da classe Class por cada classe carregada na aplicação

D todas as classes são carregadas no início da execução

E os métodos não podem ser representados como objectos

9. Qual das seguintes opções completa de forma correcta a seguinte frase: Em Java, a palavra reservada "import"...

A pode ser utilizada para herdar código de uma superclasse

B tem a função inversa à da palavra reservada "export"

C tem de ser a primeira instrução do ficheiro

D pode ser utilizada para importar campos não "static" das classes

E permite declarar uma ou mais classes

10. Em Java, e considerando o método genérico `fromArrayToCollection`, e o seguinte fragmento de código, qual ou quais das linhas contêm instruções ilegais:

```
static <T> void fromArrayToCollection(T[] a, Collection<T> c) {
  for (T o : a) {
    c.add(o);
  }
}
```

```
1. Object[] oa = new Object[100];
2. Collection<Object> co = new ArrayList<Object>();
3. fromArrayToCollection(oa, co);
4.
5. String[] sa = new String[100];
6. Collection<String> cs = new ArrayList<String>();
7. fromArrayToCollection(sa, cs);
8. fromArrayToCollection(sa, co);
9.
10.Integer[] ia = new Integer[100];
11.Collection<Integer> ci = new ArrayList<Integer>();
12.fromArrayToCollection(ia, ci);
13.fromArrayToCollection(ia, cs);
```

A linha 8

B linhas 3, 7, 8, 12 e 13

C linhas 12 e 13

D linha 12

E linha 13

11. Sobre os padrões *Adapter* e *Decorator*, qual é a afirmação verdadeira?

A são equivalentes

B o padrão *Adapter*, ao contrário do padrão *Decorator*, permite acrescentar novas funcionalidades ao objecto adaptado sem alterar a sua interface

C o padrão *Decorator* permite redefinir completamente a interface do objecto decorado

D o objecto decorador não pode ter estado

E o padrão *Adapter* permite redefinir completamente a interface do objecto adaptado

12. Em Java, qual ou quais das linhas do seguinte programa contêm instruções ilegais:

Ficheiro A.java:

```
1. package Uma;
2. public class A {
3.             void x() { System.out.println("x"); }
4.   protected void y() { System.out.println("y"); }
5.   public    void z() { System.out.println("z"); }
6. }
```

Ficheiro B.java:

```
7. package Outra;
8. import Uma.*;
9. class B extends A {
10.  void metodo(A a, B b) {
11.    a.x();
12.    a.z();
13.    b.x();
14.    b.y();
15.    b.z();
16.  }
17.}
```

A linhas 11 e 12

B linhas 11e 13

C linha 14

D linhas 12 e 15

E linhas 13 e 14

13. Indique a frase verdadeira acerca do padrão *Decorator*:

A dificulta a transição para a tecnologia OO

B diminui o número de objectos da aplicação

C permite alterar a interface dos objectos decorados

D permite adicionar novas funcionalidades aos objectos decorados

E nenhum dos anteriores

14. Qual das seguintes frases se aplica ao padrão de desenho *Command* :

A permite encapsular um algoritmo

B permite especificar os passos de um algoritmo

C delega a criação de objectos nas classes derivadas

D permite acrescentar comportamento a um objecto sem alterar a sua interface

E .ermite encapsular um pedido como sendo um objecto

15. Qual dos seguintes inconvenientes é um dos inconvenientes do padrão de desenho *Factory Method*:

A aumenta o número de objectos da aplicação

B nalguns casos diminui a flexibilidade da solução aplicada

C pode aumentar o número de classes da aplicação

D aumenta sempre o número de classes da aplicação

E leva a uma estrutura de controlo invertida do tipo "*Princípio de Hollywood*"

2ª PARTE

(6 perguntas com a cotação máxima de 12,5 valores
a cotação de cada pergunta está indicada entre parêntesis)

1. (1,5 valores) Descreva o padrão de desenho *Observer*, indicando as suas vantagens e inconvenientes.

2. (1,5 valores) Considere a modelação, em Java, do seguinte problema: um construtor de aviões constrói aparelhos que podem ser equipados com motores de vários fabricantes, desde que obedeçam às especificações do produtor da aeronave, que toma a decisão com base nas características do avião e nas preferências das companhias aéreas suas clientes. É crítico que, para além das especificações do produtor, não haja dependência na construção da aeronave face aos pormenores de construção dos motores, nem vice-versa: o produtor da aeronave deve pode mudar facilmente de um fabricante de motores para outro e assim melhorar o serviço ao cliente. Que padrão(ões) de desenho poderia aplicar? Descreva sucintamente o modelo resultante.

3. (1,5 valores) Considere o seguinte programa:

```
abstract class A {
  public char getChar() {
    return 'X'; }

  protected abstract void computeChar();

  protected abstract void destroyChar();

  public final void doIt() {
    computeChar();
    System.out.print(getChar());
    destroyChar();
  }
}

class B extends A {
  protected void computeChar() {
    System.out.print(“ computeChar B “); }

  protected void destroyChar() {
    System.out.print(“ destroyChar B “); }
}

class C extends A {
  public char getChar() {
    return 'Y'; }

  protected void computeChar() {
    System.out.print(“ computeChar C “); }

  protected void destroyChar() {
    System.out.println(“ destroyChar C “); }
}

public class Misterio {
  public static void main(String[] args) {
    A[] array = {new B(), new C()};
    for (A a: array)
      a.doIt();
    System.out.println(“”);
  }
}
```

(0,75 valor) Qual o resultado que se obtém quando se executa o programa?

_____

_____

(0,75 valor) Que padrão de desenho é usado no programa?

_____

4. (2,5 valores) Desenhe o "diagrama de classes" UML correspondente ao seguinte problema:

Uma empresa de caminhos de ferros é constituída por um conjunto de comboios e por vários empregados. Cada empregado é identificado por um dado número dentro da empresa e tem um dado salário. Existem dois tipos de empregados, os revisores e os condutores. Os primeiros são responsáveis por validar os bilhetes dos passageiros enquanto que os segundos conduzem comboios.

Um comboio é constituído por uma locomotiva e por uma ou mais carruagens. As locomotivas, por seu turno, podem ser eléctricas ou a diesel; e as carruagens podem ser de passageiros (são caracterizadas pelo número de lugares sentados) ou de mercadorias (são caracterizadas pela capacidade de carga). As carruagens de passageiros podem ainda dividir-se em classe executiva ou de turística. Nas carruagens de classe executiva apenas há lugares sentados, enquanto que nas de classe turística há lugares sentados e em pé. Todos os tipos de carruagem são ainda caracterizados pelo peso em vazio. As locomotivas são caracterizadas pela potência de tracção e pelo peso. Cada comboio tem um dado percurso a realizar que é representado pelas estações nas quais o comboio tem que parar.

Cada passageiro tem um nome e uma morada e possui pelo menos um bilhete para uma viagem de comboio. Cada bilhete indica, além da classe (executiva ou turística) e do preço correspondente, a estação de partida e a estação de chegada.

Represente as classes pelos seus nomes, métodos e atributos. Indique também as relações de herança, associação e agregação.

5. (1,5 valores) Desenhe o "diagrama de sequência" UML correspondente à execução do seguinte programa Java (este programa corresponde à pergunta 3 da segunda parte) . O diagrama de sequência deve conter os nomes das mensagens trocadas, mas não é necessário representar os argumentos dessas mensagens nem as correspondentes ao retorno. Não se esqueça de representar a criação de novos objectos.

```
abstract class A {
  public char getChar() {
    return 'X'; }

  protected abstract void computeChar();

  protected abstract void destroyChar();

  public final void doIt() {
    computeChar();
    System.out.print(getChar());
    destroyChar();
  }
}

class B extends A {
  protected void computeChar() {
    System.out.print(" computeChar B "); }

  protected void destroyChar() {
    System.out.print(" destroyChar B "); }
}

class C extends A {
  public char getChar() {
    return 'Y'; }

  protected void computeChar() {
    System.out.print(" computeChar C "); }

  protected void destroyChar() {
    System.out.println(" destroyChar C "); }
}

public class Misterio {
  public static void main(String[] args) {
    A[] array = {new B(), new C()};
    for (A a: array)
      a.doIt();
    System.out.println("");
  }
}
```

6. (4,0 valores) No jogo da tesoura, papel e pedra existem dois jogadores em que cada um pode jogar uma das seguintes três peças: tesoura, papel e pedra. A tesoura ganha ao papel, o papel ganha à pedra e a pedra ganha à tesoura. Jogadas com pedras iguais dão resultado a um empate. Primeiro defina, em Java, a classe ou as classes necessárias para representar os três tipos de peças deste jogo. Segundo, defina uma classe em Java que permite guardar as jogadas realizadas por cada um dos dois jogadores. Não é obrigatório introduzir as jogadas dos jogadores de forma alternada, pode-se indicar duas do primeiro jogador, três do segundo e outrs do primeiro, por exemplo. Adicionalmente, esta classe tem a funcionalidade que em qualquer momento pode-se saber quantas jogos foram ganhos pelo primeiro jogador e pelo segundo jogador e quantos terminaram empatados. Se no momento da execução desta funcionalidade o número de jogadas do primeiro jogador não for igual ao número de jogadas do segundo então deve ser gerado uma excepção do tipo *NumeroJogadasInvalido* (classe a concretizar também). Pretende-se ter uma solução em que seja possível adicionar novos tipos de peças sem que seja necessário alterar nenhum dos métodos já realizados.

a) Complete o seguinte código no que diz respeito à classe Jogo e especifique também a classe *NumeroJogadasInvalido*. Deve completar os métodos que não estão preenchidos e deve definir os atributos que ache necessário para guardar a informação relativa a um jogo. Assuma que a determinação do resultado de uma jogada é da responsabilidade da classe ou classes que representam as peças do jogo (representada(s) pelo tipo *Peca* no código a completar).

```
import java.util.*;
public class Jogo {


  public Jogo() {



  }

  /**
   * Adiciona uma jogada para o jogador 1.
   * @param peca a peça jogada pelo jogador 1
   **/
  public void adicionaJogada1(Peca peca) {


  }

  /**
   * Adiciona uma jogada para o jogador 2.
   * @param peca a peça jogada pelo jogador 2
   **/
  public void adicionaJogada2(Peca peca) {


  }

  /**
   * Escreve o resultado no stdout (VIT, EMP, DER relativamente ao jogador 1) para cada
   * jogada realizada pelos dois jogadores.
   *
   * @throws NumeroJogadasInvalido caso os dois jogadores tenham jogado um número distinto
   *         de jogadas.
   **/
  public void escreveResultado() {









  }
}
```

b) Defina a classe ou classes que concretizam o comportamento das várias peças do Jogo. A solução deve suportar a inserção de novos tipos de peças com um mínimo de alterações ao código já realizado.