Exame de Paradigmas da Programação IV
2ª chamada

2003.07.09
Duração: **2h30min**

Leia o teste com muita atenção antes de começar. Deve tentar resolver os exercícios procurando, o mais possível, reutilizar definições já existentes.
RESPONDA A CADA PARTE EM FOLHAS SEPARADAS.

PARTE I

1. Considere que na distribuição de Java com que está a trabalhar não existe a classe `Stack`. No entanto precisa de a implementar com o seguinte conjunto de métodos:

   - `public Stack()`
   - `public void push(Object item)`
   - `public Object pop() throws EmptyStackException`
   - `public Object peek() throws EmptyStackException`
   - `public boolean empty()`
   - `public int search(Object o)`, que devolve o índice do objecto na stack, ou `-1` caso ele não exista.

   (a) Implemente a classe `Stack`

   (b) Relativamente ao método `search`, vê necessidade de garantir que algum método tenha que estar implementado? Justifique.

2. Considere que a classe `PropostaCompra` permite modelar ofertas de compra de um determinado bem. A sua definição (incompleta) é:

```
public class PropostaCompra {
  private String nomeProponente; // quem faz a proposta
  private double qtProposta;     // quantia oferecida

  public PropostaCompra(String nomeProp, double qt) {
    this.nomeProponente = nomeProp;
    this.qtProposta = qt;
  }
  ....
}
```

Considere agora que para uma hasta pública queremos implementar uma classe para modelar stacks de propostas de compras, a que chamamos `StackPropostas`, e que permite guardar as propostas de compras oferecidas. No entanto, esta nova stack não aceita todas as propostas emitidas pelos possíveis compradores, mas apenas aquelas que não excedem o valor máximo permitido[1] (é necessário que o construtor desta classe tenha este aspecto em atenção) e que não são inferiores ao valor da proposta actualmente mais elevada.

(a) Recorrendo à classe `Stack` implemente a classe `StackPropostas` tendo em conta que a assinatura dos métodos deve ser a mesma de `Stack` (a verificação de que se trata de um objecto do tipo `PropostaCompra` deve ser feita nos métodos)

(b) Acrescente agora à classe `StackPropostas` o método `public String proponenteVencedor()` que permite saber o nome do vencedor da hasta pública.

3. Relembre o trabalho prático. Considere que foi desenvolvida a classe `Parque` que implementa a interface `IParque`:

```
public interface IParque {
  public void entra(Cartao c, Veiculo a) throws SemPermissaoException;
  public void sai(String matricula) throws VeiculoNaoExisteException;
}
```

Desenvolva a classe `ParqueComRelatorio` que, para além de implementar IParque mantém o registo de todas as entrada e saídas, permitindo obter esse registo com o método `String getRelatorio()` (Nota: utilize new GregorianCalendar() para obter a data/hora de entrada e saída).

4. Relembre, mais uma vez, o enunciado do trabalho prático. Considere que estão a ser desenvolvidas as classes: Parque, ParquePago e ParqueLivre. Existem duas propostas para a implementação do método toString:

(a)
```
public abstract class Parque {
  private String cod;
  private int lotacao;
  public String toString() {
    StringBuffer sb = new StringBuffer(this.getClass().getName());
    sb.append("(");
    sb.append(cod); sb.append(",");
    sb.append(lotacao); sb.append(")");
    return sb.toString();
  }
}
public class ParquePago extends Parque {
  ...
  public String toString() {
    StringBuffer sb = new StringBuffer(super.toString());
```

[1] Serve para evitar movimentos especulatórios.

```
    ...
    return sb.toString();
  }
}
public class ParqueLivre extends Parque {
  ...
  public String toString() {
    StringBuffer sb = new StringBuffer(super.toString());
    ...
    return sb.toString();
  }
}
```

(b)

```
public abstract class Parque {
  ...
  public abstract String toString();
}
public class ParquePago extends Parque {
  ...
  public String toString() {
    StringBuffer sb = new StringBuffer("ParquePago(");
    sb.append(getCod()); sb.append(",");
    sb.append(getLotacao()); sb.append(")");
    ...
    return sb.toString();
  }
}
public class ParqueLivre extends Parque {
  ...
  public String toString() {
    StringBuffer sb = new StringBuffer("ParqueLivre(");
    sb.append(getCod()); sb.append(",");
    sb.append(getLotacao()); sb.append(")");
    ...
    return sb.toString();
  }
}
```

Qual é, em sua opinião, a proposta mais vantajosa. **Justifique** a sua escolha.

Parte II

5. Considere que se pretende modelar um Sistema de Gestão de Conferências para suportar os próximos eventos a serem promovidos pelo Departamento. Os recursos necessários para uma conferência são um conjunto de Participantes e um conjunto de Oradores. Sobre cada *Participante* deve ficar registado o seu nome e a empresa a que pertencem. Sobre os *Oradores* além destes dados é necessária informação relativa ao preço que cobram pela sua função.

   A informação necessária a guardar de cada conferência é a seguinte:

   - O tema da conferência;
   - O preço a pagar por participante;
   - O conjunto dos oradores que a realizam, e
   - O conjunto dos participantes que nela estão inscritos.

   No entanto, existem conferências que não faz sentido realizarem-se se não existir um número mínimo de participantes a assistir. A organização teria prejuízo se tais conferências se realizassem. É necessário associar à informação de cada conferência informação que permita implementar este requisito.

   É também objectivo que o Sistema de Gestão de Conferências guarde informação relativa às conferências que estão propostas para decorrer (e já aceitam inscrições) e aquelas que já se efectuaram. É também possível saber, para uma conferência já realizada, qual foi o lucro conseguido com a sua realização.

   Tendo em conta estas premissas, responda às seguintes questões:

   (a) Modele o sistema, providenciando para o efeito um desenho das classes que o integram ("a lá BlueJ") e o "esqueleto"das classes (declaração de variáveis e construtores).

   (b) Escreva os métodos necessários para efectuar as seguintes operações sobre o Sistema de Gestão de Conferências:

      i. Criação do Sistema;
      ii. Criação de uma nova conferência (com ou sem número mínimo de participantes);
      iii. Inscrição de um participante numa conferência,
      iv. Dar por terminada uma conferência;
      v. Para uma conferência terminada calcular o lucro obtido;
      vi. Visualização do sistema sobre a forma de String;
      vii. Gravação do sistema para ficheiro utilizando uma ObjectStream,

   Deve utilizar excepções para tornar a sua solução o mais completa possível. Será valorizada a capacidade de modularidade que as soluções apresentem.