Nome:	Número:

Programação Orientada por Objectos 2010/11

1º Exame
8 de Junho de 2011

Instruções (leia com cuidado):

- Escreva de forma CLARA o seu nome e número em todas as folhas.
- O exame contém 8 páginas dividido em 4 partes. Confirme que tem um exame completo.
- A cotação de cada pergunta é indicada junto à questão e encontra-se resumida no quadro em baixo.
- Tem 2 horas para responder ao exame.
- Para planear melhor o seu tempo leia todos os problemas antes de começar.
- Este exame NÃO permite consulta. Deverá responder às questões no espaço disponível, usando a parte de trás das folhas, se necessário.
- **Sobre a mesa deverá encontrar-se APENAS este exame, uma caneta e o seu cartão de identificação.**
- Desligue o telemóvel. O seu uso anula o exame.

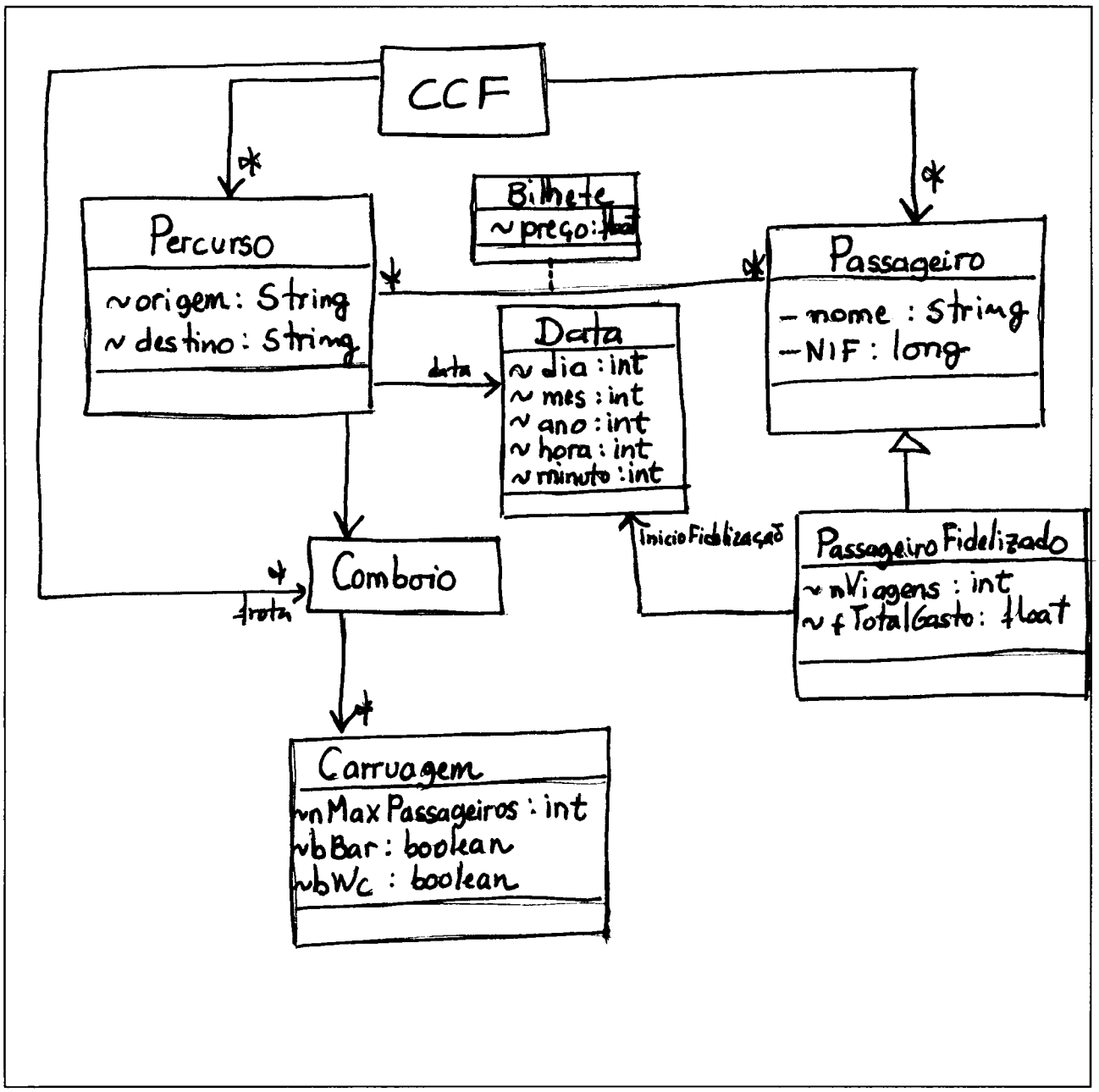
Parte	Problema	Descrição	Pág.	Valores
I	1 a) b)	UML	2	1.5
II	2 a) b) c) d)	Java: desenvolvimento	3	3.0
	3 a) b) c) d) e)	Java: escolha múltipla	6	2.5
	4	Java: miscellaneous	7	1.0
III	5	XML	8	1.0
IV	6	C++	8	1.0
Total				**10.0**

Nome:	Número:

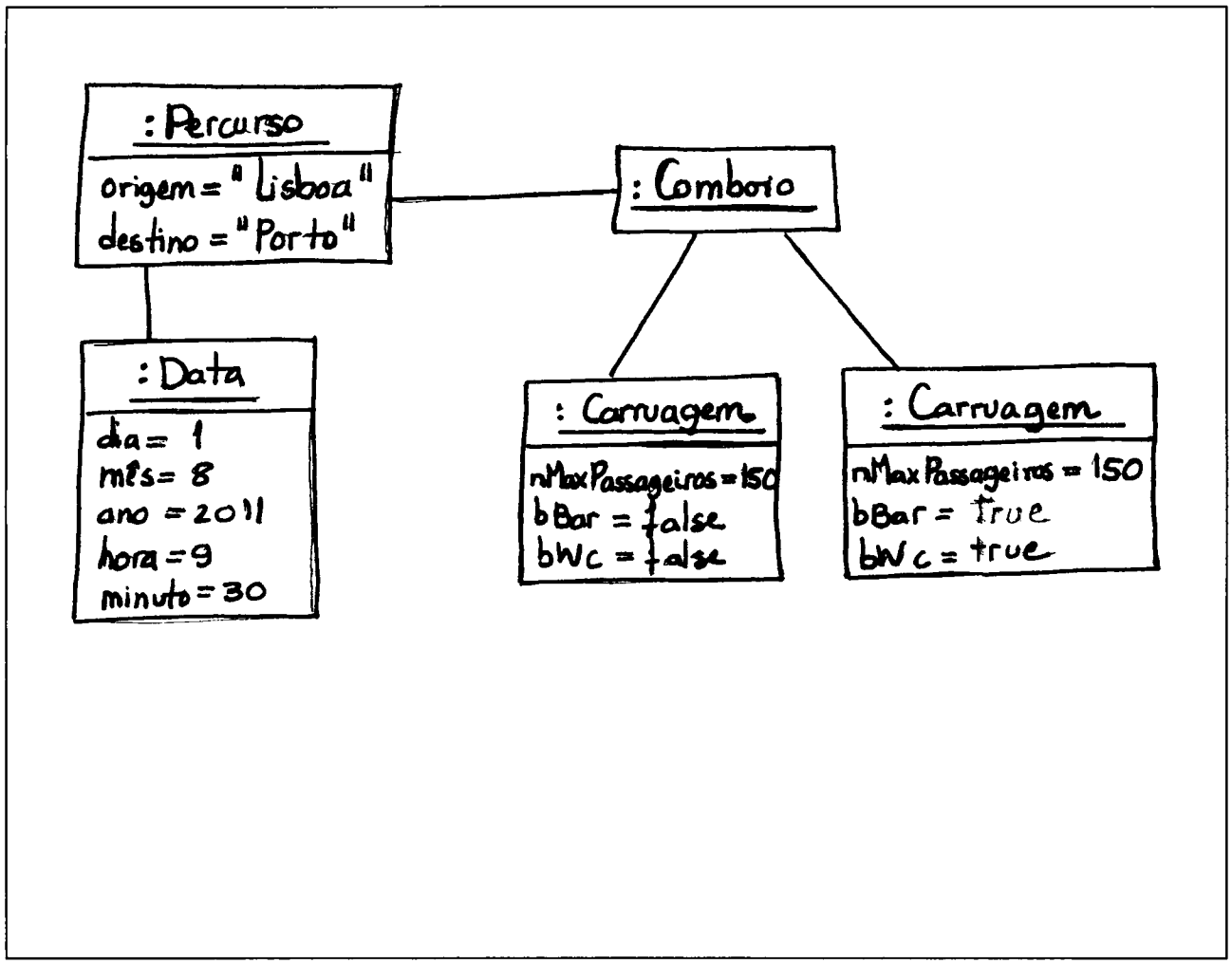
Parte I -- UML (1.5 valores)

1 – Considere que uma companhia de caminhos de ferro (CCF) pretende oferecer um sistema de gestão de bilhetes para passageiros. Assuma que a companhia oferece percursos com diversas origens e diversos destinos. Cada percurso é realizado por um determinado comboio da frota da CCF numa certa data (incluindo ano, mês, dia, hora e minutos). Cada comboio tem um conjunto variável de carruagens. Para cada carruagem deve estar disponível informação sobre o número máximo de passageiros, existência de bar e existência de WC. A CCF pretende manter um registo de todos os passageiros, guardando para o efeito o seu nome e NIF. Um passageiro pode fidelizar-se para ter desconto no preço dos bilhetes. Para tal tem-se em consideração o número de viagens, o total gasto em viagens e a data de início da fidelização do passageiro. Um bilhete diz respeito a um percurso e pertence a um e um só passageiro.

a) **[1.0 valores]** Defina o diagrama de classes em UML para o problema apresentado (ignore os métodos, a menos de selectores/modificadores que ache necessários).



b) [0.5 valores] Defina um diagrama de objectos em UML de um percurso realizado por um comboio com apenas 2 carruagens numa determinada data. Preencha os atributos com valores *dummy*.



Parte II -- Java (6.5 valores)

2 – Pretende-se oferecer um controlo remoto programável (CRP) com cinco botões. Dos cinco botões, apenas quatro são programáveis. O quinto botão serve para fazer uma operação global de *undo*. O objectivo é programar os primeiros quatro botões do CRP com operações de ligar e desligar luz (2 botões) e de abrir e fechar porta da garagem (2 botões). O quinto botão do CRP deverá reverter a última operação realizada. Assuma que a posição do quinto botão do CRP botão é diferenciada das restantes dada que o mesma não é programável.

a) **[0.5 valores]** Considere uma interface `IComando` que oferece 2 métodos, `executar` e `reverter`. Os métodos não recebem qualquer parâmetro e devolvem `void`. Defina a interface `IComando`.

```
interface IComando {
    void executar();
    void reverter();
}
```

Nome:	Número:

b) **[0.5 valores]** Considere que existe uma classe `Luz` com dois métodos, `ligar` e `desligar`, que não recebem qualquer parâmetro e devolvem `void`, e com um construtor por omissão. Defina duas concretizações de `IComando`, chamadas `LigarLuz` e `DesligarLuz`.

```
public class LigarLuz implements IComando {

      private Luz l;
      public LigarLuz(Luz l) { this.l=l; }

      public void executar(){ l.ligar(); }
      public void reverter(){ l.desligar(); }
}
```

```
public class DesligarLuz implements IComando {

      private Luz l;
      public DesligarLuz(Luz l) { this.l=l; }

      public void executar(){ l.desligar(); }
      public void reverter(){ l.ligar(); }
}
```

c) **[0.5 valores]** Considere que existe uma classe `PortaGaragem` com dois métodos, `abrir` e `fechar`, que não recebem qualquer parâmetro e devolvem `void`, e com um construtor por omissão. Defina duas concretizações de `IComando`, chamadas `AbrirPortaGaragem` e `FecharPortaGaragem`.

```
public class AbrirPortaGaragem implements IComando {

      private PortaGaragem pg;
      public FecharPortaGaragem(PortaGaragem pg) { this.pg=pg; }

      public void executar(){ pg.fechar(); }
      public void reverter(){ pg.abrir(); }
}
```

```
public class FecharPortaGaragem implements IComando {

      private PortaGaragem pg;
      public FecharPortaGaragem(PortaGaragem pg) { this.pg=pg; }

      public void executar(){ pg.abrir(); }
      public void reverter(){ pg.fechar(); }
}
```

d) **[0.75 marks]** Defina a classe `ControloRemoto` que implementa o CRP com os 5 botões. Esta deve oferecer 3 métodos: (i) `atribuir`, que recebe um inteiro que identifica a posição do botão e o respectivo comando, e que deve gravar o respectivo comando na posição pretendida; (ii) `carregar`, que recebe a posição do botão, e que deve executar a acção associada a esse botão; (iii) `reverter`, que deve realizar a operação global de *undo*.

```
public class ControloRemoto {
	private IComando[] commandos = new IComando[4];
	private IComando rcomando = null; //comando de reverter

	public void atribuir(int i, IComando c) {
		comandos[i]=c;
	}

	public void carregar(int i) {
		if (comandos[i]!=null)
			comandos[i].executa();
		rcomando=comandos[i];
	}

	public void reverter(){
		if (rcomando!=null)
			rcomando.reverte();
	}
}
```

e) **[0.75 marks]** Exemplifique a utilização de um objecto de tipo `ControloRemoto` num método `main` dentro duma classe `Main`. Para esse efeito deve conseguir realização a seguinte sequência de acções: (i) abrir a porta da garagem; (ii) fechar a porta da garagem; (iii) ligar a luz; (iv) fazer undo (para desligar a luz).

```
public class Main {
	public static void main(String[] args){
		ControloRemoto cr = new ControloRemoto();

		Luz l = new Luz();
		LigarLuz ll= new LigarLuz(l);
		DesligarLuz dl=new DesligarLuz(l);
		cr.atribuir(0,ll); //0->ligar luz
		cr.atribuir(1,dl); //1->desligar luz

		PortaGaragem pg;
		AbrirPortaGaragem apg = new AbrirPortaGaragem(pg);
		FecharPOrtaGaragem fpg = new FecharPortaGaragem(pg);
		cr.atribuir(2,apg); //2->abrir porta garagem
		cr.atribuir(3,fpg); //3->fechar porta garagem

		cr.carregar(2); //abrir porta da garagem
		cr.carregar(3); //fechar porta da garagem
		cr.carregar(1); //ligar a luz
		cr.reverter(); //reverter
	}
}
```

Nome:	Número:

3 – Preencha as respostas às perguntas de escolha múltipla na seguinte tabela (use apenas maiúsculas). Se quiser corrigir alguma resposta risque a incorrecta e escreva ao lado a resposta correcta. Cada resposta correcta vale 0.5 valores. Uma questão não respondida vale 0 valores, enquanto que uma resposta incorrecta desconta 0.2 valores.

Pergunta	a)	b)	c)	d)	e)
Resposta	D	C	D	A	B

a) **[0.5 valores]** Qual o valor do atributo de classe `x` e do atributo de instância `y` após a construçao de um segundo objecto de tipo `X` através da chamada a `X(10)`?

```
class X {
      static int x=0;
      static { x+=1; }
      int y=-1;
      { y+=2; }
      public X() { x=x+y; }
      public X(int i) { this(); }
}
```

A. `x=2 e y=1`
B. `x=2 e y=3`
C. `x=3 e y=3`
D. `x=3 e y=1`
E. Nenhuma das anteriores

b) **[0.5 valores]** Considere as seguintes classes e identifique, método a método, quais são sobreposições, redefinições ou erros de compilação. A ordem dos métodos nas alternativas (A, B, C e D) é a seguinte: `q`; `r`; `v`; `t`.

```
class X {
      X q(){…}
      void r(Number x) {…}
      String v() {…}
      <T extends Number> T t(){…}
}
```

```
class Y extends X {
      Y q(){…}
      void r(Number y) {…}
      Object v() {…}
      Integer t() {…}
}
```

A. sobreposição; sobreposição; sobreposição; erro de compilação
B. redefinição; redefinição; sobreposição; erro de compilação
C. redefinição; redefinição; erro de compilação; redefinição
D. redefinição; sobreposição; sobreposição; erro de compilação
E. Nenhuma das anteriores

c) **[0.5 valores]** O que é imprimido para o terminal?

```
class X{
      public int xpto(){return 5;}
      public static int foo() {return 15;}
}
class Y extends X{
      public int xpto(){return 10;}
      public static int foo() {return 20;}
      public static void main(String[] args){
            X x = new Y();
            Y y = (Y)x;
            System.out.print(x.foo());
            System.out.print(x.xpto());
            System.out.print(((X)y).foo());
            System.out.print(((X)y).xpto());
      }
}
```

A. `20 10 20 5`
B. `20 10 15 10`
C. `15 10 20 10`
D. `15 10 15 10`
E. Nenhuma das anteriores

Nome:	Número:

d) **[0.5 valores]** Uma cláusula *finally* é executada:

A. Sempre.
B. Apenas se o respective bloco de *try* terminar com sucesso e se não houver um *return*/*break*.
C. Apenas se o respective bloco de *try* terminar com successo.
D. Apenas se for lançada uma expecção no respective bloco de *try*.
E. Nenhuma das anteriores.

e) **[0.5 valores]** Assuma que o programador definiu uma excepção denominada `SimpleError`. De que tipo de excepção se trata?

A. `Error` B. `Checked exception` C. `Unchecked exception` D. Nenhuma das anteriores

4 – [1.0 valores] Considere o seguinte programa.

```
public class Pilha {
     private Object[] elementos;
     private int comprimento = 0;
     public static final int INITIAL_CAPACITY=16;

     public Pilha(){
          elementos = new Object[INITIAL_CAPACITY];
     }
     public void insere(Object e) {
          asseguraCapacidade();
          elementos[comprimento++]=e;
     }
     public Object retira(){
          if (comprimento==0) return null;
          return elementos[--comprimento];
     }
     private void asseguraCapacidade() {
          if (elementos.length==comprimento)
          elementos=Arrays.copyOf(elementos,2*comprimento+1);
     }
}
```

Explique o que é um *memory leak*. Diga onde se encontra um potencial *memory leak* no programa anterior e exemplifique uma possível resolução (concordante com o código oferecido).

Um *memory leak* corresponde à afectação de memória sem posterior libertação quando a mesma já não é necessária. No exemplo em cima existe um potencial *memory leak* no método retira.

Possível resolução:

```
public Object retira() {
      if (comprimento==0) return null;
      Object result = elementos[--comprimento];
      elementos[comprimento]=null;
      return result;
}
```

Nome: Número:

Parte III -- XML (1 valor)

5 – [1.0 valores] Considere um elemento Ypto, cujo seu conteúdo é uma sequência de caracteres, que tem um atributo y com valor fixo "ypto". Apresente o respectivo DTD e dois documentos XML, um válido e outro inválido.

```
<!ELEMENT Ypto (#PCDATA)>
<!ATTLIST Ypto y CDATA #FIXED "ypto">

XML válido:
<ypto y="ypto">Hello</ypto>

XML inválido:
<ypto y="hello">Hello</ypto>
```

Parte IV -- C++ (1 valor)

6 – [1.0 valores] O C++ oferece argumentos por omissão. Diga qual o objectivo do mesmo e exemplifique a sua utilização, justificando.

```
O mecanismo de argumentos por omissão serve para evitar a proliferação de
construtores/métodos com diferente número de argumentos.

class Date{
            int d,m,y;
      public:
            Date(int dd=0,mm=0,int yy=0);
      //...
}
Justificação: recorrendo aos argumentos por omissão, é possível oferecer
apenas um construtor para a classe Date; equivalentemente, e sem argumentos
por omissão, teríamos de oferecer 4 construtores:
      Date();
      Date(int yy);
      Date(int yy, int mm);
      Date(int yy, int mm, int dd);
```