Instituto Superior de Engenharia
Politécnico de Coimbra

Departamento de Engenharia Informática e de Sistemas

Licenciatura em Engenharia Informática (regime pós-laboral)
Licenciatura em Engenharia Informática (curso europeu)
Licenciatura em Engenharia Informática

Programação Avançada

Exame da época normal de 2021/22

Consulta limitada Duração: 2 horas

Notas:

- Leia o enunciado do exame até ao fim antes de iniciar a resolução
- A consulta está limitada a uma página A4 manuscrita e identificada com o nome e número de aluno
- As respostas às perguntas deverão ser distribuídas pelas folhas de prova de acordo com o seguinte:
  - Folha 1: perguntas 1, 2, 3 e 4
  - Folha 2: pergunta 5
  - Folha 3: pergunta 6

1. [15%] Indique o *output* da execução do seguinte programa:

```
class X1 {
    String s;
    public X1(String s) {
        this.s = s;
        System.out.println("X1:Constructor");
    }

    public void setS(String s) { this.s = s; }

    @Override
    public String toString() {
       return "X1=" + s + ";";
    }
}
```

```
class X2 extends X1 {
    public X2(String s) {
        super(s);
        System.out.println("X2:Constructor");
    }

    @Override
    public String toString() {
        return "X2="+s+";"+super.toString();
    }
}
```

```
public class Main {
    public static void main(
                            String[] args) {
        System.out.println("Begin");
        X1 obj1 = new X1("a");
        System.out.println(obj1);
        X2 obj2 = new X2("b");
        System.out.println(obj2);
        X1 obj3 = obj2;
        System.out.println(obj3);
        obj1.setS("k");
        if (Math.random() < 10.0)
            obj2.setS("l");
        if (obj3 instanceof X2)
            obj3.setS("m");
        System.out.println(obj1);
        System.out.println(obj2);
        System.out.println(obj3);
        System.out.println("End");
    }
}
```

2. [10%] O programa seguinte escreve nove valores booleanos, correspondentes ao resultado das operações "*add*" realizadas, e um valor inteiro no final. Indique o *output* do programa e apresente uma justificação (máx. 3 linhas) para o comportamento do mesmo.

```
class ObjectBase {
  protected double d;

  public ObjectBase(double d) { this.d = d; }
  @Override
  public boolean equals(Object o) {
    if (this == o) return true;
    if (o instanceof ObjectBase other)
      return d == other.d;
    return false;
  }
  @Override
  public int hashCode() { return 0; }
}
```

```
class Object1 extends ObjectBase{
  static int count = 0;

  public Object1(double d) { super(d); }

  @Override
  public int hashCode() {
    return count++ / 2;
  }
}
```

```
class Object2 extends ObjectBase {
  public Object2(double d) { super(d); }
}
```

```
class Object3 extends ObjectBase {
  static int count = 1;

  public Object3(double d) {
    super(d);
  }

  @Override
  public int hashCode() {
    return count++ % 2;
  }
}
```

```
public class Main {
  public static void main(
                          String[] args) {
    Set<Object> set = new HashSet<>();
    for (int i = 0; i < 3; i++) {
      System.out.println(
               set.add(new Object1(1.0)));
      System.out.println(
               set.add(new Object2(2.0)));
      System.out.println(
               set.add(new Object3(3.0)));
    }
    System.out.println(set.size());
  }
}
```

3. [10%] Defina as classes `ClassA`, `ClassB` e `ClassC` de modo que o seguinte código possa ser executado sem qualquer problema. Deverá ter em conta o(s) comentário(s) que assinalam possíveis erros. Garanta que apenas a `ClassC` pode derivar da `ClassA` (garantir que não é possível derivar outra classe a partir de `ClassA`, para além da `ClassC`).

```
interface InterfA {
    void func1();
    void func2();
}
interface InterfB extends InterfA {
    default void func1() { }
    void func3();
}
sealed abstract class BaseClass implements InterfA permits ClassA, ClassB {  }

public class Main {
    public static void main(String[] args) {
        ClassA objA = new ClassA();
        BaseClass objA1 = objA;
        InterfB objA2 = objA;
        ClassB objB = new ClassB();
        BaseClass objB1 = objB;
        //Error: InterfB objB2 = objB;
        ClassC objC = new ClassC();
        ClassA objC1 = objC;
    }
}
```

4. [10%] Tendo em consideração o tratamento de excepções em Java e respetivas consequências no comportamento dos programas, indique qual o *output* esperado para o seguinte programa.

```
class MyException extends InvalidObjectException {
    public MyException(String reason) {
        super(reason);
    }

    @Override
    public String toString() {
        return "Error: "+super.toString();
    }
}

class Project {
    String name;
    String teacher;
    String student;

    public Project(String name, String teacher, String student) throws Exception {
        this.name = name;
        this.teacher = teacher;
        this.student = student;
        if (teacher.equals(student))
            throw new MyException("Teacher and Student have the same name");
    }

    @Override
    public String toString() {
        return String.format("%s / %s / %s",name,teacher,student);
    }
}

public class Main {
    public static Project createProject(String name, String teacher, String student) {
        try {
            return new Project(name, teacher, student);
        } catch (MyException e) {
            System.out.println("Msg A");
        } catch (Exception e) {
            System.out.println("Msg B");
        } finally {
            System.out.println("Msg C");
        }
        System.out.println("Msg D");
        return null;
    }
    public static void main(String[] args) {
        System.out.println("Begin");
        Project a = createProject("A","B","C");
        Project b = createProject("A","B","B");
        Project c = createProject("A",null,"C");
        System.out.println(a+"\n"+b+"\n"+c);
        System.out.println("End");
    }
}
```

5. [30%] Assuma que o comportamento de uma melga obedece aos seguintes princípios:
    - A melga pode estar **em voo**;
    - Quando está em voo, pode mexer-se;
    - Quando se **mexe** e atinge uma superfície que não é um animal, passa a estar **pousada**;
    - Quando se **mexe** e atinge um animal (pousa no animal), passa a estar **pronta** para picar;
    - Quando está **pronta** para picar, pode **picar**, passando a estar apenas **pousada**;
    - Quando está **pousada** ou **pronta** para picar, pode **descolar**, passando a estar **em voo**;
    - Em qualquer uma das situações referidas anteriormente (**em voo**, **pousada** e **pronta** para picar), pode ser alvo de uma **tentativa de esmagamento** por parte de uma pessoa;
    - Quando uma **tentativa de esmagamento** falha (a pessoa não acerta), passa a estar **em voo** ou mantém-se nessa situação se já for esse o caso;
    - Quando a **tentativa de esmagamento** é bem-sucedida (a pessoa acerta), passa a estar **morta**, ficando nessa situação para sempre.

    Considere que se pretende implementar, em linguagem Java, o modelo de uma melga sob a forma de uma máquina de estados orientada a objetos. Recorra, para o efeito, ao padrão estudado nas aulas e aplicado ao trabalho prático para simular a evolução de estado de uma melga que respeite os princípios descritos.
    a. [10%] Idealize um diagrama que represente a máquina de estados de forma adequada, com atribuição de nomes aos estados e às transições relacionados com os termos usados na descrição do comportamento (situações em que se encontra a melga, eventos e condições).
    b. [15%] Para o diagrama desenvolvido, defina as seguintes componentes da máquina de estados em Java:
        i. Enumeração `MelgaState` que representa os estados pelos quais pode passar uma melga;
        ii. Interface `IStates`. Inclua na interface um método `public MelgaState getState()` que permita retornar uma das constantes definidas através da enumeração `MelgaState`;
        iii. Classe `MelgaAdapter` que implementa a interface `IStates`, com exceção do método `getState()`, e a partir da qual derivam todos os estados da máquina de estados. Esta classe deve ter acesso a um objeto do tipo `MelgaData` (dados que representam uma melga – *Listagem 1*) e a outro do tipo `MelgaContext` (contexto da máquina de estados – *Listagem 2*).

    Atenção: o código em falta nas classes `MelgaData` e `MelgaContext` não deve ser completado.

```
public class MelgaData {
    //...

    public boolean tentaEsmagar() {...}
    public void mexe() {...}
    public void picar() {...}
    public void descolar() {...}

    public boolean isSobreAnimal() {...}
    public boolean isSobreObjeto() {...}

    public long getNumMexidas(){...}
    public long getNumTentativasEsmagamento() {...}
    public long getNumPicadas() {...}

    //...
}
```

Listagem 1 - Estrutura parcial da classe `MelgaData`

```
public class MelgaContext {
    private MelgaData data;
    private IStates state;

    //...

    void setState(IStates state){this.state=state;}

    void mexe() {...}
    void picar() {...}
    void descolar() {...}
    boolean tentaEsmagar() {...}

    public MelgaState getState() {...}

    public long getNumMexidas() {...}
    public long getNumTentativasEsmagamento() {...}
    public long getNumPicadas() {...}
    public boolean isSobreAnimal() {...}
    public boolean isSobreObjeto() {...}

    //...
}
```

Listagem 2 - Estrutura parcial da classe `MelgaContext`

c. [5%] Complemente a enumeração `MelgaState` que definiu na alínea c) para que esta passe a ser também uma fábrica de estados. Para o feito, acrescente um (e apenas um) dos seguintes métodos à sua escolha:

```
public IStates criaMelgaState(MelgaContext context,MelgaData data)

public static IStates criaMelgaState(MelgaState tipo,MelgaContext context,MelgaData data)
```

Atenção: não desenvolva as classes correspondentes aos vários estados que são criados pela fábrica de objetos. Assuma apenas que já existem. Tenha, no entanto, o cuidado de atribuir nomes relacionados com os termos usados no enunciado.

d. [5%] Implemente uma classe que derive de `MelgaAdapter` e represente o estado em que uma melga se encontra em voo. Nessa implementação tenha em atenção os métodos indicados na *Listagem 1* e *Listagem 2*. Para desencadear transições de estado nos métodos da classe `MelgaAdapter` deve usar a fábrica de objetos desenvolvida na alínea anterior (enumeração `MelgaState`).

6. Considere que se pretende desenvolver uma interface com o utilizador em modo gráfico (GUI) elementar que:
   - Indique quantas vezes a melga se mexeu;
   - Indique quantos vezes a melga foi vítima de uma tentativa de ataque;
   - Indique a situação em que se encontra a melga;
   - Permita dar ordens para a melga se mexer;
   - Permita desencadear uma tentativa de esmagamento da melga.

   Esta GUI deve ser desenvolvida recorrendo a JavaFX e ser comparável aos exemplos da Figura 1, Figura 2 e Figura 3. Assuma, igualmente, uma abordagem de notificações assíncronas semelhante à aplicada durante as aulas e no trabalho prático, ou seja, baseada na utilização de:
   - classe `java.beans.PropertyChangeSupport`
   - interface `java.beans.PropertyChangeListener`.

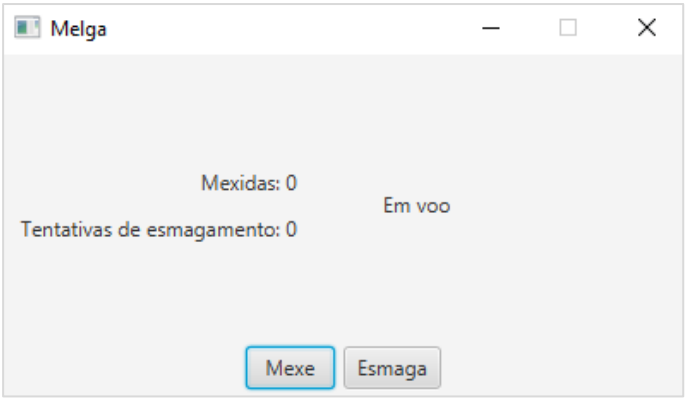

Figura 1 – GUI: 1º exemplo

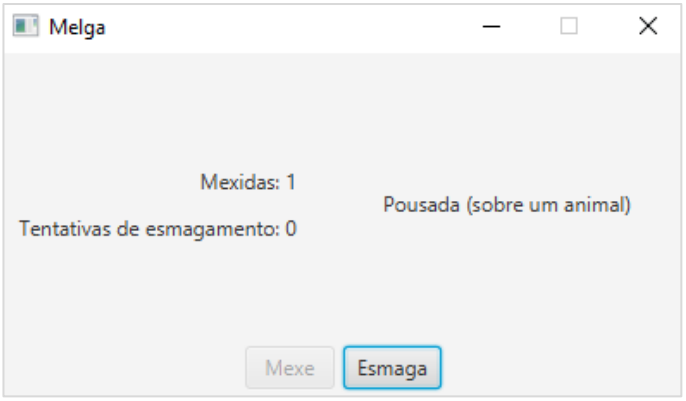

Figura 2 – GUI: 2º exemplo

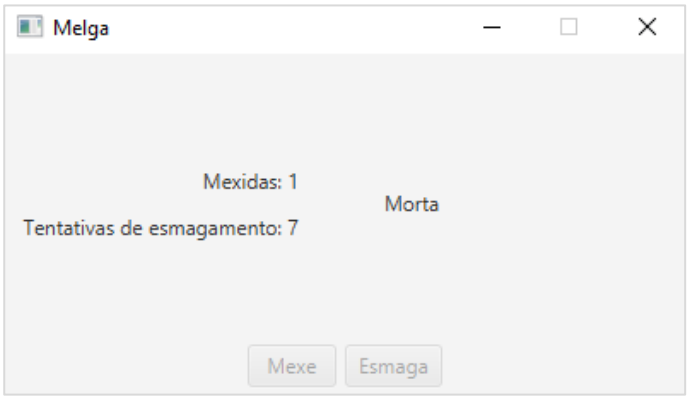

Figura 3 – GUI: 3º exemplo

Atenção: para feitos de simplificação, foi intencional especificar uma GUI incompleta neste enunciado, ao não prever a possibilidade de a melga poder "picar" ou "descolar".

a. [5%] Modifique o código da classe MelgaManager, referenciada pela GUI, para que esta atue efetivamente como modelo observável (escreva na folha de prova apenas os métodos que sofrem alguma alteração ou que possam ter que ser acrescentados).

```
public class MelgaManager {

    private MelgaContext melgaFsm;
    private PropertyChangeSupport pcs;

    public MelgaManager() {
        melgaFsm = new MelgaContext();
        pcs = new PropertyChangeSupport(this);
    }

    public void addPropertyChangeListener(PropertyChangeListener listener) {
        pcs.addPropertyChangeListener(listener);
    }

    public void mexe() { melgaFsm.mexe(); }
    public boolean tentaEsmagar() { return melgaFsm.tentaEsmagar(); }
    public MelgaState getState() { return melgaFsm.getState(); }
    public long getNumMexidas() { return melgaFsm.getNumMexidas(); }
    public long getNumTentativasEsmagamento() {
        return melgaFsm.getNumTentativasEsmagamento();
    }
    public long getNumPicadas() { return melgaFsm.getNumPicadas(); }
    public boolean isSobreAnimal() { return melgaFsm.isSobreAnimal(); }
    public boolean isSobreObjeto() { return melgaFsm.isSobreObjeto(); }
}
```

b. [15%] Complete o código da classe RootPane, para que seja obtido um resultado o mais próximo possível dos exemplos apesentados na Figura 1, Figura 2 e Figura 3. Tenha em atenção o facto de os botões deverem ficar "*disable*" quando as ações que estes desencadeiam não têm qualquer efeito no modelo da melga.

```
public class RootPane extends BorderPane {
    MelgaManager melgaManager;

    Button btTentaEsmagar, btMexe;
    Label lblMexidas, lblTentativasEsmagamento, lblSituacao;

    public RootPane(MelgaManager melgaManager){
        this.melgaManager = melgaManager;

        createViews();
        registerHandlers();
        update();
    }

    private void createViews() { /* TODO */ }

    private void registerHandlers() { /* TODO */ }

    private void update() { /* TODO */ }
}
```