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 de recurso de 2021/22

*Consulta limitada* Duração: 2 horas

**Notas:**

- Leia o enunciado do exame até ao fim antes de iniciar a resolução
- Todas as implementações devem ser realizadas em linguagem Java
- Assuma que em todo o código apresentado são realizados os *imports* necessários à compilação correta dos programas
- 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 e 2**
  - **Folha 2: pergunta 3**
  - **Folha 3: perguntas 4 e 5**

1. Considere o seguinte programa em Java:

```
class Ex1ClassA {
  private static Ex1ClassA ref = null;

  public static Ex1ClassA getRef() {
    if (ref == null) {
      System.out.println("Ex1ClassA Msg1");
      ref = new Ex1ClassA();
    }
    ++count;
    System.out.println("Ex1ClassA Msg2");
    return ref;
  }

  private Ex1ClassA() {}

  private int value = 1;
  private static int count = 0;

  public int getValue() { return value; }
  public void incrementValue() {++value; }

  @Override
  public String toString() {
    return "Ex1ClassA: " + value +
           " , count: "  + count;
  }
}
```

```
class Ex1ClassB {
  private final int value;

  Ex1ClassB(int value) { this.value = value; }

  @Override
  public String toString() {
    return "Ex1ClassB: " + value;
  }
}
```

---

```
public class E1 {
  public static void main(String[] args) {
    System.out.println("Begin");
    Ex1ClassA obj1 = Ex1ClassA.getRef();
    System.out.println(obj1);

    Ex1ClassB obj2 =
                  new Ex1ClassB(obj1.getValue());
    System.out.println(obj2);

    Ex1ClassA.getRef().incrementValue();

    System.out.println(obj2);
    System.out.println(obj1);
    System.out.println(Ex1ClassA.getRef());

    System.out.println("End");
  }
}
```

   a. [10%] Indique o *output* previsto para execução do programa.

   b. [5%] No código fornecido está presente um ou mais padrões de programação lecionados nas aulas. Identifique-o(os) referindo qual o seu objetivo.

2. No contexto do tratamento de excepções em Java , considere o seguinte programa:

```
class BaseException extends Exception {}

class SpecificException extends BaseException {}

public class E3 {
    static class Values {
        private final int[] values;

        public Values() {
            System.out.println("Init");
            values = new int[]{0, 1, 2};
        }

        public int getValuesLength() {
            return values.length;
        }

        public int getValue(int index) {
            try {
                final int value = values[index];
                switch (index) {
                    case 0 -> throw new BaseException();
                    case 1 -> throw new SpecificException();
                    case 999 -> throw new Exception();
                }
                return value;
            } catch (/* TODO A */ e) {
                System.out.println("Exception A");
            } catch (/* TODO B */ e) {
                System.out.println("Exception B");
            } catch (/* TODO C */ e) {
                System.out.println("Exception C");
            } finally {
                System.out.println("Index: " + index);
            }
            return -1;
        }
    }

    public static void main(String[] args) {
        Values values = new Values();
        System.out.println("Begin");
        for (int i = 0; i <= values.getValuesLength(); i++) {
            System.out.println("Value " + values.getValue(i));
        }
        System.out.println("End");
    }
}
```

a. [7.5%] Indique o que deve ser colocado nos *TODO*'s assinalados com A, B e C para viabilizar a compilação e execução do programa.

b. [7.5%] Indique o output da execução do programa.

3. Considere uma máquina de lavar roupa de baixo custo cujo funcionamento é o seguinte:
    - O utilizador abre a porta. Enquanto está com a porta aberta, pode colocar roupa e tirar roupa (a roupa que já não cabe é rejeitada/ignorada pela máquina).
    - Tendo a porta aberta, o utilizador pode fechar a porta, ficando pronta (a lavar).
    - Estando pronta, o utilizador pode voltar a abrir a porta (para meter ou tirar mais alguma roupa, por exemplo). Mas o mais normal é que insira detergente. Se não inserir detergente, a máquina lavará a roupa na mesma, mas esta não ficará lá muito limpa. Se o utilizador tentar inserir detergente a mais, o excedente é rejeitado/ignorado.
    - Se a máquina está pronta, o utilizador pode programar a lavagem, indicando o número de ciclos de lavagem que deseja. A lavagem inicia-se automaticamente se o número de ciclos indicado estiver de acordo com o que o fabricante idealizou (pormenor de fabrico).
    - Enquanto está a lavar, o utilizador é responsável por indicar que deseja que a máquina avance para o ciclo seguinte (há um botão para esse efeito). Nas máquinas habituais isto não é necessário e a máquina avança sozinha até ao fim dos ciclos programados. No entanto, esta máquina é de baixo custo e o utilizador tem que indicar à máquina para executar o ciclo seguinte. Enquanto não chega ao número programado de ciclos a máquina mantém-se a lavar; se atingir o número programado de ciclos a máquina regressa à situação em que está pronta.
    - Ao iniciar o ciclo seguinte, a máquina pode encravar. Se isso acontecer, a máquina fica numa situação de bloqueio e só sai daí se o utilizador efetuar uma abertura manual, ficando a máquina na situação de porta aberta de onde pode por e tirar roupa.

    Pretende-se implementar o software de controlo desta máquina em Java utilizando o padrão de máquina de estados orientado a objetos tal como abordado nas aulas. A figura seguinte representa um diagrama incompleto da máquina de estados envolvida que deve completar.


    a. [10%] Complete o diagrama da máquina de estados usando o vocabulário usado no enunciado.

    b. [5%] Elabore a enumeração `EMaqStates` com um valor para cada estado (no contexto da resposta a esta pergunta não é necessário definir um método *factory*, como realizado nas aulas, mas pode assumir a sua existência nas alíneas seguintes).

    c. [5%] Considere a interface `IMaqLavarState` que representa um estado genérico. Escreva essa interface colocando nela as funções adequadas ao diagrama e ao funcionamento da máquina de lavar descrito neste enunciado. Inclua uma função que permita obter o valor de `EMaqStates` correspondente ao estado atual.

    d. [10%] Considere a classe `MaqLavarData` que contém as funções necessárias ao controlo da máquina de lavar e a classe `MaqLavarStateAdapter` que disponibiliza implementações por omissão para todos os métodos que definiu em `IMaqLavarState` (ver listagem seguinte). Implemente de forma completa o estado "*A Lavar*" associado à situação em que a máquina está a lavar.

```
class MaqLavarData {
  //...
  public bool programa(int numCiclos) { ... } // retorna true se o número de ciclos é válido

  public void insereDetergente(int qty) { ... }  // insere detergente, descartando o excedente

  public bool programaTerminado() { ... }
       // retorna: true  se o número de ciclo de lavagem está completo
       //          false se não terminou os ciclos ou se nem sequer começou

  public bool executaProximoCiclo() { .. } // retorna false caso bloqueie

  public bool insereRoupa(PecaRoupa pr) { ... } // Assuma que PecaRoupa já existe
       // retorna false se já não couber mais roupa dentro da máquina

  public PecaRoupa removeRoupa() { ... }
       // retorna: instância de PecaRoupa em caso de sucesso (remover peca de roupa)
       //          null se a máquina está vazia
}
```

---

```
abstract class MaqLavarStateAdapter implements IMaqLavarState {
  protected MaqLavarData data;
  protected MaqLavarContext context;

  protected MaqLavarStateAdapter(MaqLavarData data, MaqLavarContext contexto) {
    this.data = data;
    this.context = contexto;
  }

  /* ... */
}
```

4. Pretende-se criar um programa em Java para controlar uma televisão. O programa deve ser implementado respeitando o padrão *Command* lecionado nas aulas. Para simplificar o exercício, assuma que as ações possíveis de uma televisão são apenas: ligar, desligar, aumentar volume, diminuir volume, avançar para o canal seguinte e recuar para o canal anterior. Para isso, assuma a existência da classe TV, que encapsula os conceitos relativos a uma televisão, e da classe CommandManager, implementada de acordo com o lecionado nas aulas.

```
class TV {
    /* ... */
    public int getChannel() { /* ... */ }

    public int getVolume() { /* ... */ }

    public boolean getOn() { /* ... */ }

    public void increaseChannel() { /* ... */ }

    public void decreaseChannel() { /* ... */ }

    public void increaseVolume() { /* ... */ }

    public void decreaseVolume() { /* ... */ }

    public void on() { /* ... */ }

    public void off() { /* ... */ }
    /* ... */
}

class CommandManager {
    /* ... */
    public boolean invokeCommand(ICommand cmd) { /* ... */ }

    public boolean undo() { /* ... */ }

    public boolean redo() { /* ... */ }

    public boolean hasUndo() { /* ... */ }
    public boolean hasRedo() { /* ... */ }
}
```

   a. [15%] Respeitando o padrão *Command*:
      i. Construa a interface Java ICommand, que declare o conjunto de ações comuns a todos os comandos
      ii. Implemente os comandos concretos para:
         1. aumentar o volume
         2. desligar a televisão. Deve garantir que ao desligar a televisão não é possível fazer *undo* de operações anteriores, seguindo a lógica de implementação do CommandManager apresentado nas aulas (na resposta a esta questão não se pretende que as classes CommandManager e TV sejam alteradas).

   b. [5%] Explique como poderia incluir uma funcionalidade adicional em que a televisão se desligasse automaticamente caso não fosse realizada qualquer ação num período seguido de 1 hora. Na resposta a esta pergunta não é necessário desenvolver código nem enquadrar a sua resposta no contexto do padrão *Command*.

5. Considere que se pretende desenvolver uma interface com o utilizador elementar e em modo gráfico (GUI) para visualizar o estado da televisão e para permitir a realização das ações sobre a TV previstas no enunciado da pergunta anterior.
   Esta GUI deve ser desenvolvida recorrendo a JavaFX e ser comparável aos exemplos das Figuras 1 a 4. Adopte, igualmente, uma abordagem de notificações assíncronas semelhante à aplicada nos exercícios das aulas e no trabalho prático, ou seja, baseada na utilização de:
   - classe `java.beans.PropertyChangeSupport;`
   - interface `java.beans.PropertyChangeListener`.

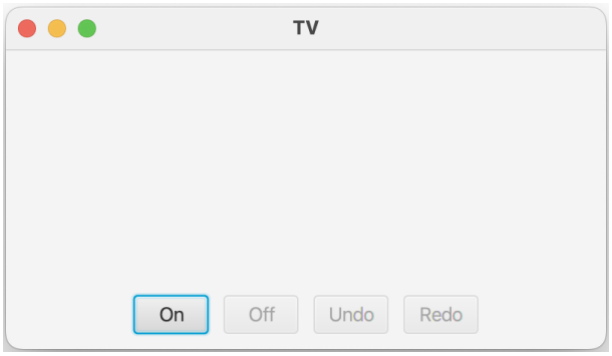


*Figura 1 – GUI: TV desligada*

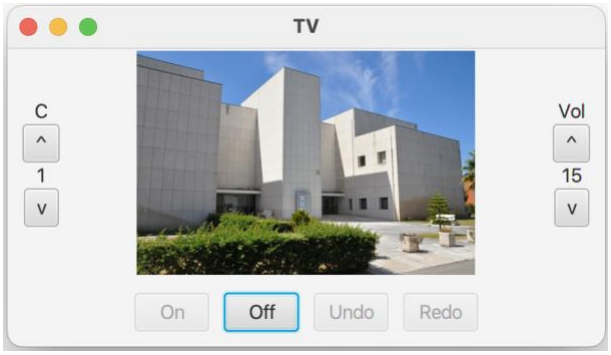


*Figura 2 – GUI: TV depois de ligada*

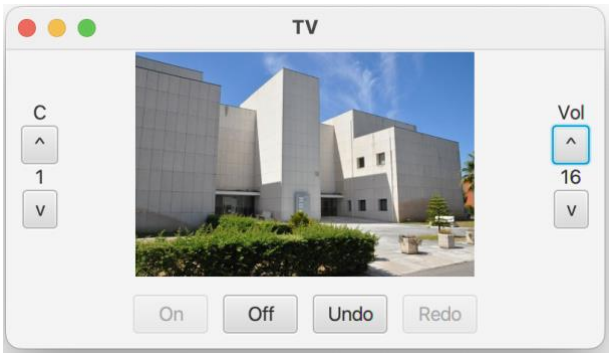


*Figura 3 – GUI: TV depois de aumentar o volume*

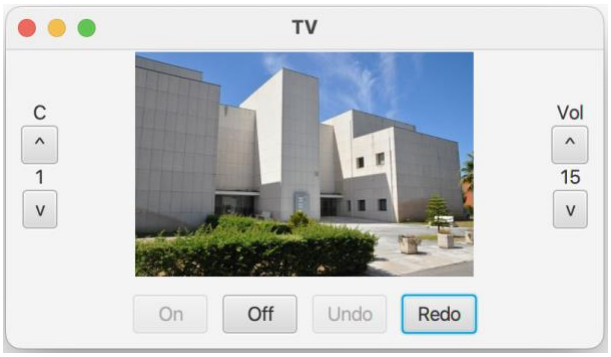


*Figura 4 – GUI: TV depois de fazer undo à ação anterior*

*<u>No contexto deste exercício não se pretende que seja implementada toda a interface mas apenas as funcionalidades especificadas nas alíneas seguintes.</u>*

a. [7.5%] Complete o código assinalado com os comentários *TODO A-D* na classe `TVManager`, referenciada pela GUI e mostrada de seguida, para que esta atue efetivamente como modelo observável (escreva na folha de prova apenas os métodos que sofrem alguma alteração). Deverá ter o cuidado de as diversas ações serem realizadas recorrendo ao padrão *Command*, através do objeto `cmdManager`, de modo a posteriormente possibilitar as ações de *undo*/*redo*.

```
public class TVManager {
    TV tv;
    CommandManager cmdManager;
    PropertyChangeSupport pcs;

    public TVManager() {
        tv = new TV();
        cmdManager = new CommandManager();
        pcs = new PropertyChangeSupport(this);
    }

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

    public void decreaseVolume()  { /* ... */ }
    public void increaseVolume()  { /* TODO A */ }
    public void decreaseChannel() { /* ... */ }
    public void increaseChannel() { /* ... */ }
    public void turnOn()          { /* ... */ }
    public void turnOff()         { /* TODO B */ }
    public void undo()            { /* TODO C */ }
    public void redo()            { /* ... */ }
    public int getVolume()        { /* ... */ }
    public int getChannel()       { /* ... */ }
    public boolean isOn()         { /* ... */ }
    public boolean hasUndo()      { /* TODO D */ }
    public boolean hasRedo()      { /* ... */ }
}
```

b. [12.5%] Assumindo o código parcial da classe `RootPane`, mostrado de seguida, apresente o código para as classes `VolumePane` e `ControlPane` para que seja obtido um resultado o mais próximo possível dos exemplos apresentados nas Figuras 1, 2, 3 e 4. Tenha em atenção o facto de os botões deverem ficar "*disabled*" quando as ações não façam sentido em determinados momentos.

```
class RootPane extends BorderPane {
    private final TVManager tv;
    ImageView ivTV;

    public RootPane(TVManager tv) {
        this.tv = tv;
        createViews();
        registerHandlers();
        update();
    }

    private void createViews() {
        /* ... */
        setCenter(ivTV); // tv image
        setLeft(new ChannelPane(tv));
        setRight(new VolumePane(tv));
        setBottom(new ControlPane(tv));
    }

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

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