

1º Trabalho

Modelação e Programação

REVISÕES, CLASSES E OBJECTOS

DATA DE ENTREGA: 16/03/2025

Data-Limite da Discussão: 30/03/2025



O aluno deverá colocar as várias resoluções relativas a este trabalho no package tp1.

Avaliação: Este trabalho é para ser **realizado individualmente**. A execução deste trabalho é obrigatória. Cada questão deste trabalho está associada a uma pontuação específica, indicada na própria questão. Para alcançar tal pontuação, a aplicação desenvolvida pelo aluno deverá satisfazer integralmente todas as exigências propostas no enunciado, e o aluno deverá ser capaz de apresentar e justificar o trabalho efetuado na respetiva questão. Na eventualidade de não se cumprir esta condição, a pontuação atribuída nessa questão será de zero. A apresentação do trabalho terá de ocorrer nas aulas práticas da unidade curricular, até à data-limite divulgada na capa deste trabalho. A falta de participação na discussão resultará na reprovação à unidade curricular. Não é permitido o uso de ferramentas de geração automática de código (ex.: ChatGPT e similares). A utilização destas ferramentas resultará numa classificação de zero no trabalho.

Relatório: Este trabalho está isento de relatório se o código entregue estiver devidamente comentado seguindo as regras semânticas da ferramenta Javadoc (os outputs desta ferramenta deverão ser entregues conjuntamente com o código fonte do trabalho).

Nota acerca da metodologia de execução do trabalho: Encoraja-se enfaticamente que, durante a elaboração deste trabalho, os estudantes procedam à pesquisa na internet sobre o funcionamento dos algoritmos e operações matemáticas solicitadas, evitando a tentação de utilizar (ou procurar compreender) códigos previamente desenvolvidos, e que **comecem o seu desenvolvimento a partir de uma página vazia**.



Parte A – Revisões

Coloque as alíneas deste grupo no package tp1.pack1Revisoes

A1 (2v) Desenvolva o programa **NotasISEL** que, no seu método `main`, execute o seguinte:

1. Deve solicitar ao utilizador o número de alunos (ou de notas) que pretende inserir.
2. Criar uma estrutura de dados (e.g. um array) para receber as notas inseridas.
3. Ordenar essa estrutura de dados por ordem crescente (implementar e usar, por exemplo, o método **bubblesort**).
4. Calcular e exibir na consola as seguintes informações:
 - a. A nota mínima inserida
 - b. A nota máxima inserida
 - c. A média das notas
 - d. O número de notas que estão acima ou abaixo da média.

Nota: Em caso de introdução de valores inválidos (por exemplo, valores não numéricos, ou notas negativas), deve ser apresentada uma mensagem de erro e solicitado novamente ao utilizador que introduza a sequência correta de números.

A2 (2v) Crie o programa **PMatrixOperations** que, no seu método `main`, deve solicitar ao utilizador o tamanho de uma matriz quadrada ($n \times n$) e, em seguida, pedir ao utilizador para preencher todos os elementos dessa matriz. Após a introdução da matriz, o programa deve oferecer ao utilizador as seguintes operações a serem realizadas com a matriz:

1. Calcular e exibir a transposta da matriz;
2. Calcular e exibir a soma de todos os elementos da matriz;
3. Verificar e indicar se a matriz é simétrica (uma matriz é simétrica se for igual à sua transposta).

Para cada uma dessas operações, o programa deve exibir o resultado na consola. Em caso de introdução de valores inválidos (por exemplo, caracteres não numéricos onde se esperam números), deve ser apresentada uma mensagem de erro e solicitado novamente ao utilizador que introduza os dados corretos. Para esta questão, é obrigatório utilizar `arrays` bidimensionais para a representação da matriz.

A3 (2v) Este trabalho visa exercitar a recursividade, a manipulação de arrays bidimensionais e a formatação de saída em Java. Pretende-se desenvolver um programa que, em Java, gere na consola uma representação fractal utilizando caracteres ASCII. O fractal escolhido para este exercício é o Triângulo de Sierpinski. As especificações são as seguintes:

Estrutura de Dados:

Utilize uma matriz bidimensional de caracteres (por exemplo, `char[][] canvas`) para armazenar a forma do fractal. Inicialmente, cada posição da matriz deve ser preenchida com um caractere de espaço (' ').

Geração do Fractal:




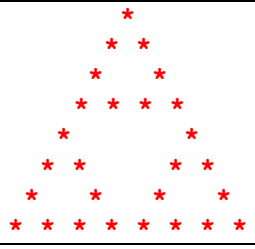
Crie a função recursiva

```
private static void drawSierpinski(char[][] canvas, int row, int col, int size)
```

para desenhar o Triângulo de Sierpinski, tendo em conta o seguinte:

- Quando o tamanho do triângulo é 1, o caractere "*" é colocado na posição do array correspondente às variáveis `row` e `col`, que no início da recursividade são definidas como zero na função principal.
- No passo recursivo, a função desenha três triângulos menores (topo, canto inferior esquerdo e canto inferior direito) nas posições adequadas dentro da matriz (usando `row`, `col`, `size`).
- O tamanho do fractal é controlado através da constante inteira `size` (potência de 2), que representa tanto a altura do triângulo quanto a metade do comprimento total da base (ou seja, a matriz terá uma dimensão de `size` por $(2 * size)$).

Saída:

			
Size = 1	2	4	8

Parte B – O Jardim Zoológico

Coloque as classes deste cenário no package tp1.zoo



B1 (7v) Considere o seguinte enunciado, criado para um trabalho de programação em Java, com o objetivo de promover a aprendizagem da programação orientada a objetos. Pretende-se desenvolver uma aplicação informática para gerir um jardim zoológico. Nesta aplicação, deverá ser modelado um conjunto de entidades (classes) que representem os diferentes animais, a sua alimentação e também os funcionários (tratadores, administração, atendimento, etc.). Cada uma destas entidades terá métodos e atributos que deverão ser definidos, discutidos com o professor, implementados e posteriormente defendidos pelo aluno na discussão. Devem ser criadas pelo menos duas classes (e no máximo quatro) por cada entidade (por exemplo, *Elefante*, *Leão*, *Tratador*, *Administradora*, *Palha*, *Carne*, etc.). Cabe ao aluno refletir, desenhar em UML, debater com o docente e implementar as classes e as associações entre as várias entidades.

Para cada entidade criada, deverá incluir o seguinte:

- Construtor(es);
- Métodos acessores, tendo em conta que certos atributos não podem ser alterados depois de instanciados. Por exemplo, no caso de um funcionário, não se deve permitir a alteração do nome, do Número de Funcionário, do BI, do Número de Contribuinte (NC) e da data de nascimento após a sua criação;
- `toString`;
- `print`;
- `equals`.

Nota: Antes de começar a programar, comece por desenhar o UML das classes utilizando uma das muitas ferramentas disponíveis para o efeito como, por exemplo:

- <https://online.visual-paradigm.com/diagrams/solutions/free-class-diagram-tool/>
- Usando o package TikZ-UML do latex (se quiser desenhar o diagrama programaticamente).
- Editor UML do IDE que o aluno esteja a utilizar (e.g. o `uml-designer` do eclipse).

Implemente os métodos que executam as funcionalidades supra descritas e, no método **main** da classe **TesteZoo**, instancie vários objetos de cada uma das classes, e mostre o output (`print`) na consola de cada entidade. Exemplo:

TesteZoo.java

```
public class TesteZoo {

    public static void main(String[] args) {

        // Instanciar alguns animais (exemplo)
        Elefante elefante1 = new Elefante("Dumbo", 10, 1200);
        Leao leao1 = new Leao("Simba", 5, 180);

        // Instanciar alguns funcionários (exemplo)
        Tratador tratador1 = new Tratador("João", "Secção de Elefantes");
        Administracao admin1 = new Administracao("Maria", "Financeiro");

        // Instanciar alguns tipos de alimentação (exemplo)
        Carne carne1 = new Carne("Carne de Vaca");
        Palha palha1 = new Palha("Palha de trigo");

        // Apresentar na consola as instâncias criadas
        System.out.println("=== Animais ===");
        System.out.println(elefante1);
        System.out.println(leao1);

        System.out.println("\n=== Funcionários ===");
        System.out.println(tratador1);
        System.out.println(admin1);

        System.out.println("\n=== Tipos de Alimentação ===");
        System.out.println(carne1);
        System.out.println(palha1);
    }
}
```

Consola:

```
=== Animais ===
Elefante {nome=Dumbo, idade=10, peso=1200}
Leao {nome=Simba, idade=5, peso=180}

=== Funcionários ===
Tratador {nome=João, secção=Secção de Elefantes}
Administracao {nome=Maria, departamento=Financeiro}

=== Tipos de Alimentação ===
Carne {tipo=Carne de Vaca}
Palha {tipo=Palha de trigo}
```

Parte C – A Classe Zoo

C1 (7v) Adicione ao projeto a classe **Zoo**. Esta classe deverá conter um conjunto de Funcionários, Animais e outras entidades desenvolvidas na alínea anterior. Atualize o diagrama UML, tendo em conta o seguinte:

O Zoo possui um nome, um preçário e a capacidade de adicionar novos animais, funcionários e outras entidades relevantes definidas na alínea B. Deve igualmente permitir visualizar na consola e obter todos os animais e funcionários existentes no jardim zoológico. Além disso, deve facultar a referência para um tratador, desde que seja fornecido o BI (ou CC). Por fim, deve oferecer uma estrutura de dados onde constem todos os animais que estão ao cuidado de um determinado tratador.

Nota: O aluno tem igualmente a possibilidade de adicionar outras funcionalidades. Tanto a classificação desta alínea, como a da alínea B (majorada em 7 valores), está parcialmente dependente da originalidade e qualidade do programa apresentado na discussão.

O código relativo a este trabalho deverá ser submetido até ao dia **16 de março**. A discussão do mesmo terá lugar nas aulas práticas até ao dia **30 de março de 2024**.

Bom trabalho, Jorge Branco, José Múrias, Andreia Artífice, Pedro Fazenda

