

Prática 3 Estruturas de dados e persistência

Objetivos

- Resolução de pequenos problemas recorrendo a instruções de controlo de fluxo e estruturas de dados (antes da introdução de modelos de classes e objetos)
- Decomposição de complexidade por meio de métodos/funções
- Manipulação e formatação de texto – Strings (objetos do tipo *java.lang.String*)
- Persistência da informação para além da execução do programa

Tópicos

- Instruções de decisão (*if*, *switch*)
- Ciclos (*for*, *while*, *do .. while*)
- Funções (métodos estáticos)
- Java Collections
- Ficheiros

Exercício 3.1

Escreva um programa que leia do teclado um número inteiro positivo e devolva a soma de todos os números primos até esse valor (inclusive). Repare que deve validar o valor de entrada repetindo a leitura se o valor for inválido (positivo).

Deve implementar uma função que devolva se um número é um número primo. Um número natural é um número primo quando tem exatamente dois divisores naturais distintos: o número 1 e ele mesmo.

Exercício 3.2

O jogo *AltoBaixo* consiste em tentar adivinhar um número (inteiro) entre 1 e 100. O programa escolhe um número aleatoriamente. Depois, o utilizador insere uma tentativa e o programa indica se é demasiado alta, ou demasiado baixa. Isto é repetido até o utilizador acertar no número. O jogo deve indicar quantas tentativas foram feitas e a seguir perguntar: “Pretende continuar? Prima (S)im”. O programa termina caso a resposta seja diferente de “S” ou “Sim”.

Sugestão: para ler uma palavra utilize o método next: String resposta = sc.next();

Exercício 3.3

Usando como base o código seguinte (*CollectionTester.java*) compare o desempenho de algumas das coleções em Java, tais como *ArrayList*, *LinkedList*, *HashSet* e *TreeSet*.

```
public class CollectionTester {

    public static void main(String[] args) {
        int dim = 5000;
        Collection<Integer> col = new ArrayList<>();
        Map<String, Double> res = checkPerformance(col, dim);
        System.out.println(col.getClass().getSimpleName());
        System.out.println(res);
    }

    private static Map<String, Double> checkPerformance(Collection<Integer> col,
final int DIM) {
        double start, stop, delta;
        Map<String, Double> out = new HashMap<>();

        // Add
        start = System.nanoTime(); // clock snapshot before
        for(int i=0; i<DIM; i++ )
            col.add( i );
        stop = System.nanoTime(); // clock snapshot after
        delta = (stop-start)/1e6; // convert to milliseconds
        out.put("add", delta);

        // Search
        start = System.nanoTime(); // clock snapshot before
        for(int i=0; i<DIM; i++ ) {
            int n = (int) (Math.random()*DIM);
            if (!col.contains(n))
                System.out.println("Not found???" + n);
        }
        stop = System.nanoTime(); // clock snapshot after
        delta = (stop-start)/1e6; // convert nanoseconds to milliseconds
        out.put("search", delta);

        // Remove
        start = System.nanoTime(); // clock snapshot before
        Iterator<Integer> iterator = col.iterator();
        while (iterator.hasNext()) {
            iterator.next();
            iterator.remove();
        }
        stop = System.nanoTime(); // clock snapshot after
        delta = (stop-start)/1e6; // convert nanoseconds to milliseconds
        out.put("remove", delta);

        return out;
    }
}
```

- a) Execute o código acima para avaliar a performance nas operações de adicionar um elemento, procurar um elemento, e remover um elemento num *ArrayList* com 5000 objetos de *Integer*.

- b) Adapte o programa de modo a medir os resultados para várias dimensões da coleção (i.e., em vez de serem sempre 5000 objetos, meça a performance com diferentes números). Substitua o ArrayList por outras estruturas de dados. Analise os resultados comparando estruturas e operações. Crie uma tabela de saída similar à seguinte:

Collection	1000	5000	10000	20000	40000	100000
ArrayList						
add	0,5	...				
search	11,5					
remove	1,2					
LinkedList						
...						

Quais são as conclusões que tira face às características de cada estrutura de dados?

Exercício 3.4

Comece por implementar um programa que lê do teclado as notas da componente prática (*notaP*) e teórica (*notaT*) de uma turma.

Defina uma função para calcular a nota final de um aluno a partir das notas das componentes prática e teórica, da seguinte forma:

- 66 (reprovado por nota mínima), se tiver obtido menos do que 7.0 em pelo menos uma das componentes;
- $0.4 * notaT + 0.6 * notaP$ (arredondada a unidades), nos restantes casos.

Informado pelos resultados do exercício 3.3., crie uma estrutura de dados para armazenar as notas das componentes teórica e prática de todos os alunos. Processe as notas e imprima os resultados com o formato seguinte:

NotaT	NotaP	Pauta
11.3	9.3	10
16.7	5.1	66
7.8	18.9	14
10.6	15.9	14
16.9	5.9	66
1.9	12.7	66
17.6	4.8	66
0.7	12.1	66
8.7	8.6	9
19.2	1.4	66
17.5	3.4	66
11.6	11.4	11
7.2	8.5	8
1.9	1.4	66
19.3	14.9	17
0	12.1	12

- a) Altere o programa para ler estes valores a partir de um ficheiro em que cada linha tem (separado por “\t”) nome, notaT e notaP – por esta ordem
- b) Imprima no ecrã quais os alunos que tiveram aprovação nessa disciplina

Exercício 3.5 (exploração de LLMs)

Utilizando um motor público de AI (pex., ChatGPT, Deepseek, outro) pergunte:

“Give me a sample java code that reads a tab separated file containing name, noteP and noteT in each line and in the end calculates the average per parameter. Should store also present the names of the persons with noteT above the overall average without using Records. “

- a) Execute o programa
- a) Pergunte ao motor de AI “can you use records in the solution?”
- b) Analise a resposta e execute o programa
- c) Tente alterar o programa sem usar AI para apresentar os alunos com classificação final acima da média da turma

Nota: crie o seu próprio ficheiro de teste ou peça ao motor público de AI.

Exercício 3.6

Escreva um programa que lê do teclado uma data no formato *mm/yyy* (validando os valores) e o dia da semana em que começa esse mês (1 = Segunda, 2 = Terça, 3 = Quarta, 4 = Quinta, 5 = Sexta, 6 = Sábado, 7 = Domingo). O programa deve depois apresentar no monitor o calendário desse mês com o formato que a seguir se apresenta. Implemente a funcionalidade pretendida com três funções: leitura de valores, com validação, cálculo de dias no mês, considerando os anos bissextos, e impressão de resultados.

```
February 2025
Su Mo Tu We Th Fr Sa
                1
 2  3  4  5  6  7  8
 9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28
```

- Altere o programa para assumir a data de hoje no dia da semana caso seja introduzido um número negativo

How to Determine Day of Week by Passing Specific Date in Java?
<https://www.baeldung.com/java-get-day-of-week>

Get the day of week for a particular date in Java
<https://www.tutorialspoint.com/get-the-day-of-week-for-a-particular-date-in-java>

Exercício 3.7

O seguinte excerto de código permite ler todas as palavras (palavra a palavra) de um ficheiro de texto, que terá de estar localizado na pasta do projeto. Pode criar este ficheiro com um editor de texto ou usar um qualquer ficheiro de código java.

```
public static void main(String[] args) throws IOException{
    Scanner input = new Scanner(new FileReader("words.txt"));
    while (input.hasNext()) {
        String word = input.next();
        System.out.println(word);
    }
}
```

- a) Teste o excerto de código para listar o conteúdo do ficheiro.
- b) Guarde numa estrutura de dados adequada todas as palavras com mais do que 2 caracteres.
- c) Liste todas as palavras terminadas em 's'.
- d) Remova da estrutura todas as palavras que contenham outros caracteres que não letras.