Laboratório 04 — Objetos de Domínio, Encapsulamento e Repositórios em Memória (2h)

Unidade Curricular: Programação Orientada aos Objetos — Java
Modalidade: Laboratório em computador pessoal
Duração sugerida: 2h

> **Estrutura deste enunciado:** A **alínea 2)** contém o **enunciado** em tarefas **a)**–**i)**, incluindo quais **classes** da biblioteca Java (e **métodos**) deverão ser usados.
> A **alínea 3)** apresenta a **resolução guiada** a→i com código e justificações.
> No **Apêndice**, encontra uma explicação *detalhada* das classes/métodos usados.

---

1) Objetivos de aprendizagem

Ao concluir este laboratório, será capaz de:

- Projetar uma **classe de domínio** simples com **encapsulamento**, **invariantes**, **construtores**, `getters`, `toString`, `equals`/`hashCode`.
    - Os **invariantes** em programação orientada a objetos são regras ou condições que devem ser sempre verdadeiras para um objeto durante toda a sua existência, exceto, possivelmente, durante a execução de métodos internos que alteram o estado do objeto de forma controlada. Ou seja: (1) são condições que definem um estado válido para o objeto e (2) devem ser mantidas por todos os métodos públicos da classe. Os invariantes representam **restrições** sobre o estado dos atributos do objeto. Servem para garantir a **consistência** e a **validez** de cada instância da classe.
      Exemplo de invariante: Numa classe ContaBancaria, um possível invariante seria que o saldo nunca pode ser negativo (saldo≥0saldo≥0).
      Aplicação dos invariantes: (1) São verificados e mantidos pelos métodos públicos da classe (por exemplo, o método `sacar` deve garantir que não seja possível sacar mais do que o saldo disponível); (2) Devem ser validados no construtor e em quaisquer métodos que modifiquem o estado do objeto (por exemplo, o construtor deve garantir que o saldo inicial seja não negativo); (3) A violação de um invariante geralmente resulta em uma exceção ou erro, indicando que o objeto está em um estado inválido.(3) Fazem parte do **encapsulamento**, pois apenas métodos autorizados ajustam os atributos de maneira que o invariante não seja quebrado.
      Resumo: Os invariantes são condições essenciais para que o objeto esteja sempre em um estado válido, protegendo dados de corrupção e facilitando manutenção e testes.
- Implementar um **repositório em memória** de objetos com operações **CRUD** e **busca**.
- Separar camadas: **Interface com o utilizador - I/O (CLI)** vs. **lógica de domínio/repositório**.
- Aplicar **validação** e **tratamento de erros** coerente (pré-condições e mensagens de utilizador).

---

2) Enunciado — Tarefas a realizar (a→h) + Biblioteca Java a usar

Estrutura de diretórios a usar (igual aos labs anteriores)

Crie a pasta **`lab04/`** com a mesma convenção usada nos Labs anteriores:

```
lab04/
  ├─ src/
```

```
│   └─ main/
│       └─ java/
│           └─ pt/
│               └─ escnaval/
│                   └─ exercicios/
│                       ├─ MenuAlunos.java
│                       ├─ Aluno.java
│                       ├─ AlunoRepo.java
│                       └─ UtilsIO.java
├─ README.md
├─ .gitignore
└─ out/
```

`.gitignore` recomendado

```
out/
bin/
target/
build/
.vscode/
*.class
*.log
.DS_Store
Thumbs.db
```

Biblioteca Java a usar nesta resolução (classes + métodos relevantes)

- **`java.util.Scanner`** — leitura do terminal (sempre `nextLine()` + `parse`).
- **`java.lang.String`** — `trim()`, `isEmpty()`, `toLowerCase()`, `contains(...)`.

  Regra: continuar a ler **linhas** com `Scanner.nextLine()` e converter manualmente (`Integer.parseInt`).

---

a) Preparar ambiente e validar ferramentas

Objetivo: Confirmar JDK/VS Code ok.
Tarefas: Registar `javac -version`, `java -version` no `README.md`.
Aceitação: evidências no README.

b) Criar a estrutura e `.gitignore`

Objetivo: Criar a estrutura padrão (acima) e `.gitignore`.
Aceitação: árvore criada + commit inicial.

c) Iniciar Git e primeiro commit

Objetivo: Histórico atómico.
Tarefas: `git init`, configurar autor, `git add .`, `git commit -m "Lab04: estrutura inicial"`.
Aceitação: commit criado.

d) Implementar `Aluno` (classe de domínio, encapsulamento)

Objetivo: Representar um aluno com invariantes.
Campos obrigatórios: `id:int` (>0, único no repositório), `nome:String` (não vazio).
Requisitos:

- Construtor com validação (`id>0`, `nome` não nulo/vazio).
- `getId()`, `getNome()`, `setNome(String)` (valida não vazio).
- `toString()` legível.
- `equals/hashCode` **baseados em `id`** (identidade no domínio).

e) Implementar `AlunoRepo` (repositório em memória)

Objetivo: CRUD + busca num tabela de `Aluno`.
Requisitos (métodos públicos):

- `listarPorId()` e `listarPorNome()` → impressão ordenada.
- `adicionar(Aluno)` → falha se `id` já existe.
- `removerPorId(int id)` → `true`/`false`.
- `findById(int id): Aluno`
- `buscarPorNome(String termo)` → imprime alunos cujo nome contém nome (*case-insensitive*).

f) Implementar `MenuAlunos` (CLI com sentinela)

Objetivo: CLI com: `1) Listar por ID 2) Listar por Nome 3) Adicionar 4) Remover por ID 5) Buscar 6) Renomear por ID 0) Sair`.
Requisitos:

- `do/while` + `switch` (setas) + fluxos auxiliares (`adicionarFluxo`, `removerFluxo`, `buscarFluxo`, `renomearFluxo`).
- Validação robusta (`UtilsIO.lerInt`, `UtilsIO.lerOpcao`).

g) Qualidade e mensagens

Objetivo: UX (User Experience - Experiência do Utilizador) consistente.
Requisitos: mensagens claras, `printf` para tabelas, linha em branco entre operações.

h) README + Tabela de testes + comandos

Objetivo: Evidenciar verificação.
Requisitos: tabela com **10** casos (exemplos na alínea 3), comandos `javac/java`, notas de design (encapsulamento, equals/hashCode).

---

3) Resolução guiada — comentários e justificação (a→i)

a–c) Ambiente, Estrutura e Git

```
cd lab04
git init
git config user.name  "meu.nome"
git config user.email "meu.email@example.com"
echo "# Lab 04 — Objetos de Domínio e Repositórios" > README.md
git add .
git commit -m "Lab04: estrutura inicial, README e .gitignore"
```

d) Classe de domínio `Aluno`

`src/main/java/pt/escnaval/exercicios/Aluno.java`:

```java
package pt.escnaval.exercicios;

public class Aluno {
    private final int id;
    private String nome;
    private String mensagemErro;

    public Aluno(int id, String nome) {
        this.mensagemErro = "";

        if (id <= 0) {
            this.mensagemErro = "Erro: id deve ser > 0";
            this.id = -1;
            this.nome = "";
            return;
        }
        this.id = id;
        setNome(nome); // reutiliza validação
    }

    public int getId() { return id; }

    public String getNome() { return nome; }

    public String getMensagemErro() { return mensagemErro; }

    public void setNome(String nome) {
        this.mensagemErro = "";

        if (nome == null) {
            this.mensagemErro = "Erro: nome não pode ser null";
            return;
        }

        String n = nome.trim();
        if (n.isEmpty()) {
            this.mensagemErro = "Erro: nome não pode ser vazio";
            return;
        }
        this.nome = n;
    }

    @Override public String toString() {
        return String.format("%d\t%s", id, nome);
    }

    @Override public boolean equals(Object o) {
        if (this == o) return true;
        if (!(o instanceof Aluno)) return false;
        Aluno other = (Aluno) o;
        return id == other.id; // identidade pelo id
    }

    @Override public int hashCode() {
        return Integer.hashCode(id);
    }
}
```

Notas: `id` é **final** (imutável após criação). Invariantes centralizadas. `equals/hashCode` por `id` permitem procurar/remover eficientemente.

e) Repositório `AlunoRepo` (ordenação)

`src/main/java/pt/escnaval/exercicios/AlunoRepo.java`:

```java
package pt.escnaval.exercicios;

public class AlunoRepo {
    private Aluno[] dados;
    private int tamanho;
    private static final int CAPACIDADE_INICIAL = 10;

    public AlunoRepo() {
        this.dados = new Aluno[CAPACIDADE_INICIAL];
        this.tamanho = 0;
    }

    private void redimensionar() {
        Aluno[] novoArray = new Aluno[dados.length * 2];
        for (int i = 0; i < tamanho; i++) {
            novoArray[i] = dados[i];
        }
        dados = novoArray;
    }

    public boolean adicionar(Aluno a) {
        if (findById(a.getId()) != null) return false;

        if (tamanho == dados.length) {
            redimensionar();
        }
        dados[tamanho] = a;
        tamanho++;
        return true;
    }

    public boolean removerPorId(int id) {
        int indice = -1;
        for (int i = 0; i < tamanho; i++) {
            if (dados[i].getId() == id) {
                indice = i;
                break;
            }
        }

        if (indice == -1) return false;

        for (int i = indice; i < tamanho - 1; i++) {
            dados[i] = dados[i + 1];
        }
        tamanho--;
        return true;
    }

    public Aluno findById(int id) {
        for (int i = 0; i < tamanho; i++) {
            if (dados[i].getId() == id) {
                return dados[i];
            }
        }
        return null;
    }

    public void listarPorId() {
        Aluno[] copia = new Aluno[tamanho];
        for (int i = 0; i < tamanho; i++) {
            copia[i] = dados[i];
        }
        ordenarPorId(copia, tamanho);
        for (int i = 0; i < tamanho; i++) {
            System.out.println(copia[i]);
        }
```

```
    }

    public void listarPorNome() {
        Aluno[] copia = new Aluno[tamanho];
        for (int i = 0; i < tamanho; i++) {
            copia[i] = dados[i];
        }
        ordenarPorNome(copia, tamanho);
        for (int i = 0; i < tamanho; i++) {
            System.out.println(copia[i]);
        }
    }

    private void ordenarPorId(Aluno[] arr, int n) {
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j].getId() > arr[j + 1].getId()) {
                    Aluno temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    private void ordenarPorNome(Aluno[] arr, int n) {
        for (int i = 0; i < n - 1; i++) {
            for (int j = 0; j < n - i - 1; j++) {
                if (arr[j].getNome().compareTo(arr[j + 1].getNome()) > 0) {
                    Aluno temp = arr[j];
                    arr[j] = arr[j + 1];
                    arr[j + 1] = temp;
                }
            }
        }
    }

    public void buscarPorNome(String termo) {
        String t = termo == null ? "" : termo.toLowerCase();
        boolean algum = false;
        for (int i = 0; i < tamanho; i++) {
            if (dados[i].getNome().toLowerCase().contains(t)) {
                System.out.println(dados[i]);
                algum = true;
            }
        }
        if (!algum) System.out.println("(nenhum resultado)");
    }
}
```

`@Override` é uma **anotação** (annotation) que serve para:

1. **Documentar intenção**: Indica explicitamente que um método está a **sobrescrever** (overriding) um método de uma superclasse ou interface.

2. **Detecção de erros em tempo de compilação**: O compilador verifica se:

    - O método na classe pai/interface realmente existe
    - A assinatura (nome, parâmetros, tipo de retorno) está **exatamente correta**
    - Se houver erro, o compilador **avisa** imediatamente

3. **Melhora legibilidade**: Deixa claro para quem lê o código que esse método é uma redefinição de um método herdado.

Aqui, @Override indica que estes métodos estão sobrescrevendo os da classe Object.
NOTA: NÃO é obrigatório, mas é altamente recomendado:

Aspecto	Sem `@Override`	Com `@Override`
Compilação	Funciona normalmente	Funciona normalmente
---	---	---
Erro de digitação	❌ Passa despercebido	✅ Compilador avisa
Legibilidade	Menos clara	Mais clara
Boas práticas	Não segue padrão	Segue padrão Java

Decisões: cópia defensiva antes de ordenar (não alteramos ordem interna).

f) CLI `MenuAlunos`

`src/main/java/pt/escnaval/exercicios/MenuAlunos.java`:

```
package pt.escnaval.exercicios;

import java.util.Scanner;

public class MenuAlunos {
    private static final AlunoRepo repo = new AlunoRepo();

    public static void main(String[] args) {
        try (Scanner sc = new Scanner(System.in)) {
            int op;
            do {
                mostrarMenu();
                op = UtilsIO.lerOpcao(sc, 0, 6);
                switch (op) {
                    case 1 -> repo.listarPorId();
                    case 2 -> repo.listarPorNome();
                    case 3 -> adicionarFluxo(sc);
                    case 4 -> removerFluxo(sc);
                    case 5 -> buscarFluxo(sc);
                    case 6 -> renomearFluxo(sc);
                    case 0 -> System.out.println("A terminar...");
                    default -> System.out.println("Opção inválida.");
                }
                System.out.println();
            } while (op != 0);
        }
    }

    static void mostrarMenu() {
        System.out.println("=== MENU ALUNOS ===");
        System.out.println("1) Listar por ID");
        System.out.println("2) Listar por Nome");
        System.out.println("3) Adicionar");
        System.out.println("4) Remover por ID");
        System.out.println("5) Buscar por termo");
        System.out.println("6) Renomear por ID");
        System.out.println("0) Sair");
        System.out.print("Opção (0..6) → ");
    }

    static void adicionarFluxo(Scanner sc) {
        int id = UtilsIO.lerInt(sc, "ID (inteiro > 0): ");
        System.out.print("Nome: ");
        String nome = sc.nextLine();
        try {
```

```
            boolean ok = repo.adicionar(new Aluno(id, nome));
            System.out.println(ok ? "Adicionado." : "Falha: ID já existente.");
        } catch (IllegalArgumentException e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }

    static void removerFluxo(Scanner sc) {
        int id = UtilsIO.lerInt(sc, "ID a remover: ");
        boolean ok = repo.removerPorId(id);
        System.out.println(ok ? "Removido." : "ID não encontrado.");
    }

    static void buscarFluxo(Scanner sc) {
        System.out.print("Termo (parte do nome): ");
        String termo = sc.nextLine();
        repo.buscarPorTermo(termo);
    }

    static void renomearFluxo(Scanner sc) {
        int id = UtilsIO.lerInt(sc, "ID a renomear: ");
        var opt = repo.findById(id);
        if (opt.isEmpty()) {
            System.out.println("ID não encontrado.");
            return;
        }
        System.out.print("Novo nome: ");
        String novo = sc.nextLine();
        try {
            opt.get().setNome(novo);
            System.out.println("Atualizado.");
        } catch (IllegalArgumentException e) {
            System.out.println("Erro: " + e.getMessage());
        }
    }
}
```

f.1) `UtilsIO` (reutilizável da série)

`src/main/java/pt/escnaval/exercicios/UtilsIO.java`:

```
package pt.escnaval.exercicios;

import java.util.Scanner;

public final class UtilsIO {
    private UtilsIO() {}

    private static boolean ehInteiro(String s) {
        if (s == null || s.trim().isEmpty()) {
            return false;
        }
        String trimmed = s.trim();
        if (trimmed.startsWith("-")) {
            trimmed = trimmed.substring(1);
        }
        for (char c : trimmed.toCharArray()) {
            if (!Character.isDigit(c)) {
                return false;
            }
        }
        return true;
    }

    public static int lerInt(Scanner sc, String prompt) {
```

```
            System.out.print(prompt);
            while (true) {
                String s = sc.nextLine();
                if (ehInteiro(s)) {
                    return Integer.parseInt(s.trim());
                }
                System.out.print("Inteiro inválido. Tente novamente: ");
            }
        }

        public static int lerOpcao(Scanner sc, int min, int max) {
            while (true) {
                String s = sc.nextLine();
                if (ehInteiro(s)) {
                    int op = Integer.parseInt(s.trim());
                    if (op < min || op > max) {
                        System.out.printf("Opção fora do intervalo. (%d..%d) → ", min, max)
                    } else {
                        return op;
                    }
                } else {
                    System.out.print("Opção inválida. Introduza um número: ");
                }
            }
        }
    }
```

g) UX e formatação

- Saída tabular (`id\t nome`) e linhas em branco entre operações → **legibilidade**.
- Mensagens de erro originadas do domínio (ex.: "id deve ser > 0") são mostradas como **feedback** ao utilizador.

h) README + Tabela de testes + comandos

Tabela exemplo (preencher com saídas reais):

#	Caso	Passos/Entrada	Saída esperada
1	Listar vazio	`1`	`(nenhum resultado)` ou nada
2	Adição válida	`3 → id=1, nome=Ana`	`Adicionado.`
3	Duplicado	`3 → id=1, nome=Ana`	`Falha: ID já existente.`
4	Remover existente	`4 → id=1`	`Removido.`
5	Remover inexistente	`4 → id=2`	`ID não encontrado.`
6	Buscar existente	`5 → "na"`	mostra `1 Ana`
7	Buscar inexistente	`5 → "zzz"`	`(nenhum resultado)`
8	Renomear existente	`6 → id=1, novo=Ana Maria`	`Atualizado.`
9	Renomear inexistente	`6 → id=9`	`ID não encontrado.`
10	Nome inválido	`3 → id=2, nome=" "`	`Erro: nome não pode ser vazio`

Compilar/Executar (Linux/macOS)

```
# a partir de lab04/
find src -name "*.java" -print0 | xargs -0 javac -d out
java -cp out pt.escnaval.exercicios.MenuAlunos
```

PowerShell (Windows)

```
Get-ChildItem -Recurse src -Filter *.java | % { $_.FullName } | % { & javac -d out $_ }
java -cp out pt.escnaval.exercicios.MenuAlunos
```

---

4) Critérios de avaliação (formativa)

- **Classe de domínio** correta (encapsulamento, invariantes, `equals`/`hashCode`, `toString`).
- **Repositório** funcional (CRUD, busca, ordenações) e **separação** I/O vs. lógica.
- **Validação** e mensagens de erro claras; sem exceções visíveis ao utilizador.
- README com **tabela de 10 testes** e instruções de execução; código formatado e legível.

---

5) Resolução de problemas (FAQ)

- **IDs duplicados** → verifique `findById` antes de adicionar.
- **Ordenação muda ordem interna** → use **cópias** antes de ordenar.
- **`Could not find or load main class`** → confirme `package` e `-cp out` (`pt.escnaval.exercicios.MenuAlunos`).

---

6) Cronograma sugerido (120 min)

- **0–10** a) Ambiente
- **10–20** b) Estrutura + `.gitignore`
- **20–25** c) Git init
- **25–45** d) Classe `Aluno`
- **45–80** e) Repositório `AlunoRepo`
- **80–100** f–g) CLI + UX
- **115–120** i) README + testes

---

7) Apêndice — Coleções e classes (detalhado)

`Scanner` e `String`

- **Scanner:** `nextLine` + `parse` (`Integer.parseInt`) .
- **String:** `trim`, `isEmpty`, `toLowerCase`, `contains` para validações e buscas.
