Exercicio Collections

📝 Tarefa 01 — Controle de usuários online

(Set + Map)

📂 Estrutura sugerida

```
css

exercicio01_usuarios_online
 ├── UsuarioService.java
 └── Main.java
```

UsuarioService.java

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

public class UsuarioService {

    private Set<String> usuariosOnline = new HashSet<>();
    private Map<String, Integer> acoesPorUsuario = new HashMap<>();

    public void login(String usuario) {
        usuariosOnline.add(usuario);
    }

    public void registrarAcao(String usuario) {
        usuariosOnline.add(usuario);
        acoesPorUsuario.merge(usuario, 1, Integer::sum);
    }

    public Set<String> getUsuariosOnline() {
        return usuariosOnline;
    }

    public Map<String, Integer> getAcoesPorUsuario() {
        return acoesPorUsuario;
    }
}
```

Main.java

```
public class Main {
    public static void main(String[] args) {

        UsuarioService service = new UsuarioService();

        service.login("ana");
        service.registrarAcao("ana");
        service.registrarAcao("ana");

        service.login("bruno");
        service.registrarAcao("bruno");

        System.out.println("Usuários online: " +
service.getUsuariosOnline());
        System.out.println("Ações:");
        service.getAcoesPorUsuario()
                .forEach((u, qtd) -> System.out.println(u + " -> " +
qtd));
    }
}
```

📝 Tarefa 02 — Limpeza de pedidos inválidos

(List + removeIf)

📁 Estrutura

```
css

exercicio02_pedidos
 ├── Pedido.java
 └── Main.java
```

Pedido.java

```
public class Pedido {

    private String id;
    private double valor;
    private String status;

    public Pedido(String id, double valor, String status) {
        this.id = id;
        this.valor = valor;
        this.status = status;
    }

    public double getValor() {
        return valor;
    }

    public String getStatus() {
        return status;
    }

    public String getId() {
        return id;
    }
}
```

Main.java

```
import java.util.ArrayList;
import java.util.List;

public class Main {
    public static void main(String[] args) {

        List<Pedido> pedidos = new ArrayList<>();
        pedidos.add(new Pedido("P1", 100, "OK"));
        pedidos.add(new Pedido("P2", 50, "CANCELADO"));
        pedidos.add(new Pedido("P3", -10, "OK"));

        pedidos.removeIf(p ->
                p.getValor() <= 0 ||
"CANCELADO".equals(p.getStatus())
        );

        System.out.println("Pedidos válidos:");
        for (Pedido p : pedidos) {
            System.out.println(p.getId());
        }
    }
}
```

📝 Tarefa 03 — Funcionários por departamento

(Map + List + computeIfAbsent)

📁 Estrutura

```
exercicio03_departamentos
 ├── Funcionario.java
 └── Main.java
```

Funcionario.java

```
public class Funcionario {

    private String nome;
    private String departamento;

    public Funcionario(String nome, String departamento) {
        this.nome = nome;
        this.departamento = departamento;
    }

    public String getNome() {
        return nome;
    }

    public String getDepartamento() {
        return departamento;
    }
}
```

Main.java

```
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

public class Main {
    public static void main(String[] args) {

        List<Funcionario> funcionarios = List.of(
                new Funcionario("Ana", "TI"),
                new Funcionario("Carlos", "TI"),
                new Funcionario("Bruno", "RH")
        );

        Map<String, List<Funcionario>> porDepartamento = new
HashMap<>();

        for (Funcionario f : funcionarios) {
            porDepartamento
                    .computeIfAbsent(f.getDepartamento(), k -> new
ArrayList<>())
                    .add(f);
        }

        porDepartamento.forEach((dep, lista) -> {
            System.out.print(dep + ": ");
            for (Funcionario f : lista) {
                System.out.print(f.getNome() + " ");
            }
            System.out.println();
        });
    }
}
```

📝 Tarefa 04 — Ranking de vendas

(List + Comparator + sort)

📂 Estrutura

```
exercicio04_ranking_vendas
 ├── Vendedor.java
 └── Main.java
```

Vendedor.java

```
public class Vendedor {

    private String nome;
    private double totalVendas;

    public Vendedor(String nome, double totalVendas) {
        this.nome = nome;
        this.totalVendas = totalVendas;
    }

    public String getNome() {
        return nome;
    }

    public double getTotalVendas() {
        return totalVendas;
    }
}
```

Main.java

```
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

public class Main {
    public static void main(String[] args) {

        List<Vendedor> vendedores = new ArrayList<>();
        vendedores.add(new Vendedor("Ana", 2000));
        vendedores.add(new Vendedor("Carlos", 1500));
        vendedores.add(new Vendedor("Bruno", 900));

        vendedores.sort(

Comparator.comparingDouble(Vendedor::getTotalVendas).reversed()
        );

        int posicao = 1;
        for (Vendedor v : vendedores) {
            System.out.println(posicao + " - " + v.getNome() + " ("
+ v.getTotalVendas() + ")");
            posicao++;
        }
    }
}
```

📝 Tarefa 05 — Cache de configurações imutável

(Map + unmodifiableMap)

📂 Estrutura

```
exercicio05_config_cache
 ├── ConfigService.java
 └── Main.java
```

ConfigService.java

```
import java.util.Collections;
import java.util.HashMap;
import java.util.Map;

public class ConfigService {

    private Map<String, String> configs = new HashMap<>();

    public ConfigService() {
        configs.put("timezone", "America/Sao_Paulo");
        configs.put("lang", "pt-BR");
    }

    public Map<String, String> getConfigs() {
        return Collections.unmodifiableMap(configs);
    }
}
```

Main.java

```
import java.util.Map;

public class Main {
    public static void main(String[] args) {

        ConfigService service = new ConfigService();
        Map<String, String> config = service.getConfigs();

        System.out.println("Configurações carregadas: " + config);

        try {
            config.put("nova", "teste");
        } catch (UnsupportedOperationException e) {
            System.out.println("Erro: mapa não pode ser
modificado");
        }
    }
}
```

✅ Resultado pedagógico

Com essas tarefas o aluno pratica **exatamente o que o mercado exige**:

- Escolha correta de Collection
- Performance básica implícita
- Código limpo, legível e seguro
- Uso real de `removeIf`, `merge`, `computeIfAbsent`, `sort`, `unmodifiableMap`