Lista 2

Wesley Sant'Anna

1 de setembro de 2025

a Exercícios de desenvolvimento de código

1 Atributos Privados

```java
public class Dados {
    private String nome;
    private int idade;

    public String get_nome() {
        return nome;
    }

    public int get_idade() {
        return idade;
    }

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

    public void setIdade(int idade ){
        this.idade = idade;
    }
}
```

2 Modificadores de acesso

Pacote 1

```java
package pacote1;

public class Acesso {

    private String privado = "Privado";
    String padrao = "Padrao (package-private)";
    protected String protegido = "Protegido";
    public String publico = "Publico";

    private String metodoPrivado() {
        return "Metodo Privado";
    }

    String metodoPadrao() {
        return "Metodo Padrao";
    }

    protected String metodoProtegido() {
        return "Metodo Protegido";
    }

    public String metodoPublico() {
        return "Metodo Publico";
    }
}
```

Pacote 2

```java
package pacote2;
import pacote1.Acesso;
public class Main {
    public static void main(String[] args) {
        Acesso obj = new Acesso();

        // System.out.println(obj.privado); Erro: private
        // System.out.println(obj.padrao); Erro: package-
           private
        // System.out.println(obj.protegido); Erro: protected
           (fora do pacote e sem heranca)
        System.out.println(obj.publico);

        // System.out.println(obj.metodoPrivado()); Erro:
           private
        // System.out.println(obj.metodoPadrao()); Erro:
           package-private
        // System.out.println(obj.metodoProtegido Erro:
           protected
        System.out.println(obj.metodoPublico());// OK
    }
}
```

3 Encapsulamento do Construtor

```java
public class Conta {
    private int num_conta;
    private float saldo;

    public Conta(float saldoInicial) {
        this.saldo = saldoInicial;
    }

    public void setConta(int num_conta) {
        this.num_conta = num_conta;
    }

    public void depositar(float valor) {
        this.saldo += valor;
    }

    public boolean sacar(float valor) {
        if (valor <= saldo) {
            saldo -= valor;
            return true;
        } else {
            return false;
        }
    }

    public float getSaldo() {
        return saldo;
    }

    public int getNumConta() {
        return num_conta;
    }
}


public class Main {
    public static void main(String[] args) {
        Conta dados = new Conta(7.7f);
        dados.setConta(4023);

        dados.depositar(100.0f);
        boolean saque = dados.sacar(50.0f);

        System.out.println("Numero da conta: " + dados.
           getNumConta());
        System.out.println("Saldo atual: " + dados.getSaldo())
           ;
        System.out.println("Saque realizado: " + (saque ? "Sim
           " : "Nao"));
    }
}
```

4 Lógica Getter e Setter

```
class Temperatura {
    private float temp_kelvin;

    public Temperatura(float kelvin) {
        if (kelvin >= 0) {
            this.temp_kelvin = kelvin;
        } else {
            throw new IllegalArgumentException("Temperatura abaixo do zero absoluto!"); }}

    public void setKelvin(float kelvin) {
        if (kelvin >= 0) {
            this.temp_kelvin = kelvin;
        } else {
            System.out.println("Valor abaixo do zero absoluto!");}}

    public void setCelsius(float celsius) {
        float kelvin = celsius + 273.15f;
        setKelvin(kelvin);}

    public void setFahrenheit(float fahrenheit) {
        float kelvin = (fahrenheit - 32) / 1.8f + 273.15f;
        setKelvin(kelvin);
    }

    public float getKelvin() {
        return temp_kelvin;
    }

    public float getCelsius() {
        return temp_kelvin - 273.15f;
    }

    public float getFahrenheit() {
        return (temp_kelvin - 273.15f) * 1.8f + 32; }}

public class Main {
    public static void main(String[] args) {
        Temperatura t = new Temperatura(300f);
        System.out.println("Kelvin: " + t.getKelvin());
        System.out.println("Celsius: " + t.getCelsius());
        System.out.println("Fahrenheit: " + t.getFahrenheit());

        t.setCelsius(25f);
        System.out.println("Nova em Kelvin: " + t.getKelvin());
    }
}
```

5 Classe Imutáve

Ponto2D.java

```java
public final class Ponto2D {
    private final float x;
    private final float y;

    public Ponto2D(float x, float y) {
        this.x = x;
        this.y = y;
    }

    public float getX() {
        return x;
    }

    public float getY() {
        return y;
    }
}
```

Main.java

```java
public class Main {
    public static void main(String[] args) {
        Ponto2D ponto = new Ponto2D(2.5f, 3.5f);

        System.out.println("As coordenadas sao:" + "(" + (
            ponto.getX()) + ";" + (ponto.getY()) + ")");
    }
}
```

6 Acesso privado ao pacote

pacoteinterno/ClassePrivada.java

```java
// arquivo: pacoteinterno/ClassePrivada.java
package pacoteinterno;

class ClassePrivada {
    public void mostrar() {
        System.out.println("ClassePrivada: so pode ser acessada dentro do pacote.");
    }
}
```

pacoteinterno/ClassePublica.java

```java
// arquivo: pacoteinterno/ClassePublica.java
package pacoteinterno;

public class ClassePublica {
    public void mostrar() {
        System.out.println("ClassePublica: pode ser acessada de qualquer lugar.");
    }
}
```

pacoteexterno/TesteAcesso.java

```java
// arquivo: pacoteexterno/TesteAcesso.java
package pacoteexterno;

import pacoteinterno.ClassePublica;
// import pacoteinterno.ClassePrivada; // Isso causara erro de compilacao

public class TesteAcesso {
    public static void main(String[] args) {
        ClassePublica publica = new ClassePublica();
        publica.mostrar();

        // ClassePrivada privada = new ClassePrivada(); // Erro: ClassePrivada nao e publica
    }
}
```

b Exercícios Teóricos:

1 **Definição de encapsulamento:** Encapsulamento é um dos pilares da programação orientada a objetos (POO). Em Java, ele consiste em restringir o acesso direto aos atributos de uma classe, permitindo que sejam acessados apenas por meio de métodos públicos (como getters e setters). Isso protege os dados internos da classe contra alterações indevidas e promove uma interface controlada para interação com o objeto. O encapsulamento é essencial para garantir modularidade, segurança e clareza no desenvolvimento de sistemas.

2 **Explicação dos modificadores de acesso:** Java possui quatro modificadores de acesso que controlam a visibilidade de classes, atributos e métodos:

- `private`: O membro só pode ser acessado dentro da própria classe. Ideal para atributos que não devem ser expostos diretamente.
- `default` (sem modificador): O membro é acessível apenas dentro do mesmo pacote. Útil para classes e métodos que fazem parte de uma implementação interna.
- `protected`: O membro é acessível dentro do mesmo pacote e por subclasses, mesmo que estejam em pacotes diferentes. Usado em herança para permitir extensão controlada.
- `public`: O membro é acessível de qualquer lugar. Usado para APIs públicas e métodos que devem ser amplamente utilizados.

3 **Benefícios do encapsulamento:** Encapsular dados e comportamentos traz diversos benefícios para o desenvolvimento de software:

- **Manutenção**: Alterações internas na classe não afetam outras partes do sistema, desde que a interface pública permaneça estável.
- **Reutilização**: Classes bem encapsuladas são mais fáceis de reaproveitar em diferentes contextos, pois possuem responsabilidades bem definidas.
- **Segurança**: Impede acesso direto a dados sensíveis, evitando manipulações indevidas e garantindo integridade.
- **Controle**: Permite aplicar regras e validações ao acessar ou modificar atributos, mantendo consistência no estado do objeto.

4 **Encapsulamento vs. Abstração:** Embora relacionados, encapsulamento e abstração têm propósitos distintos:

- **Encapsulamento** trata de esconder os detalhes internos de implementação e proteger os dados de acesso externo.
- **Abstração** foca em destacar apenas os aspectos essenciais de um objeto, ignorando detalhes irrelevantes para o usuário.

Encapsulamento é uma técnica para implementar abstração. Juntos, esses conceitos permitem construir sistemas modulares, seguros e fáceis de entender, promovendo a separação de responsabilidades e a redução de complexidade.

5 **Objetivo do getter e do setter:** Os métodos `getter` e `setter` são usados para acessar e modificar atributos privados de uma classe:

- `getAtributo()`: retorna o valor de um atributo.
- `setAtributo(valor)`: altera o valor de um atributo, podendo incluir validações.

Esses métodos são úteis para manter o encapsulamento, permitindo controle sobre como os dados são acessados e modificados. Em cenários como objetos imutáveis ou quando o atributo não deve ser alterado após a criação, os `setters` podem ser evitados. Já os `getters` podem ser omitidos quando o valor não precisa ser exposto.

Referências

[1] H. M. Deitel and P. J. Deitel. *Java How To Program: Early Objects*. Prentice Hall, Boston, 11 edition, 2020.

[2] H. M. Deitel and P. J. Deitel. *Java How To Program: Late Objects*. Prentice Hall, Boston, 11 edition, 2020.

[3] Cay Horstmann and Gary Cornell. *Core Java, Volume I: Fundamentals*. Pearson, New York, 11 edition, 2018.

[4] Cay Horstmann and Gary Cornell. *Core Java, Volume II: Advanced Features*. Pearson, New York, 11 edition, 2018.

[5] Ricardo Santos. *Introdução à Programação Orientada a Objetos usando Java*. Campus/Elsevier, Rio de Janeiro, 2 edition, 2013.

[6] Herbert Schildt. *Java: The Complete Reference*. Oracle Press, New York, 9 edition, 2014.