Seção 09: Pilares POO

Construtor

- Um construtor é executado automaticamente quando um objeto é instanciado.
- Ele é frequentemente utilizado para inicializar os atributos do objeto e pode ter sobrecarga, permitindo diferentes maneiras de instanciar a classe.

Exemplo:

```
class Produto {
    private String nome;
    private double preco;

    // Construtor com parâmetros
    public Produto(String nome, double preco) {
        this.nome = nome;
        this.preco = preco;
    }
}
```

- Se nenhum construtor for definido, a classe tem um construtor padrão sem parâmetros.

Exemplo:

```
public class Produto {
    // Construtor padrão (opcional)
    public Produto() {
    }
}

Produto p = new Produto();  // utilizando o construtor padrão
```

Palavra this

- É uma referência para o próprio objeto
- Usos comuns:
  - Diferenciar atributos de variáveis locais
  - Passar o próprio objeto como argumento na chamada de um método ou construtor

```
class Produto {
    private String nome;
    private double preco;

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

    public void setPreco(double preco) {
        this.preco = preco;
    }
}
```

Sobrecarga

- Uma classe pode ter múltiplos construtores, cada um com diferentes parâmetros. Isso permite flexibilidade na criação de objetos, fornecendo diferentes maneiras de instanciá-los, dependendo dos dados disponíveis no momento da criação.

Exemplo:

```java
public class Produto {
    private String nome;
    private double preco;

    // Construtor com parâmetros
    public Produto(String nome, double preco) {
        this.nome = nome;
        this.preco = preco;
    }

    // Construtor com apenas o nome do produto
    public Produto(String nome) {
        this.nome = nome;
    }

    // Construtor sem parâmetros
    public Produto() {
        this.nome = "Desconhecido";
        this.preco = 0.0;
    }
}
```

Encapsulamento

- É um princípio que consiste em esconder detalhes de implementação de uma classe, expondo apenas operações seguras e que mantenham os objetos em um estado consistente.
- O **encapsulamento** visa proteger os dados de uma classe, tornando seus atributos privados e acessíveis apenas por métodos específicos **(getters e setters)**. Isso mantém o estado do objeto consistente, evitando modificações indevidas. O encapsulamento esconde detalhes de implementação e fornece uma interface pública para interagir com o objeto de forma segura.

Exemplo:

```java
public class Produto {
    private String nome;
    private double preco;

    public Produto(String nome, double preco) {
        this.nome = nome;
        this.preco = preco;
    }

    public String getNome() {
        return nome;
    }

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

```java
    public double getPreco() {
        return preco;
    }

    public void setPreco(double preco) {
        if (preco > 0) {
            this.preco = preco;
        } else {
            System.out.println("Preço inválido.");
        }
    }

    @Override
    public String toString() {
        return nome + " - R$ " + String.format("%.2f", preco);
    }
}
```

Modificadores de Acesso:

Os **modificadores de acesso** em Java controlam a visibilidade e o nível de acesso dos membros (atributos e métodos) de uma classe. Eles definem quais partes do código podem acessar ou modificar esses membros. Aqui estão os principais modificadores:

1. `private`: O membro (atributo ou método) só pode ser acessado dentro da própria classe. É o nível mais restrito de acesso.
2. **Sem modificador (padrão)**: Quando nenhum modificador é especificado, o membro é acessível apenas por classes no mesmo **pacote**. Esse nível é chamado de *package-private*.
3. `protected`: O membro pode ser acessado por classes no mesmo pacote e também por subclasses, mesmo que estejam em pacotes diferentes.
4. `public`: O membro pode ser acessado por qualquer classe, desde que o pacote onde ele está seja exportado em um módulo (se o projeto estiver modularizado). É o nível mais aberto de acesso.

Tabela de Visibilidade

Modificador	Mesma Classe	Mesmo Pacote	Subclasse (mesmo pacote)	Subclasse (outro pacote)	Qualquer Classe
`private`	Sim	Não	Não	Não	Não
Sem Modificador (Padrão)	Sim	Sim	Sim	Não	Não
`protected`	Sim	Sim	Sim	Sim	Não
`public`	Sim	Sim	Sim	Sim	Sim