Seção 08: Introdução a POO

Classe

- É um tipo estruturado que contém membros como atributos (dados) e métodos (funções).
- Pode incluir recursos como construtores, sobrecarga, encapsulamento, herança e polimorfismo.
- Exemplos:
  - **Entidades**: Produto, Cliente, Triangulo.
  - **Serviços**: ProdutoService, ClienteService, EmailService.
  - **Controladores**: ProdutoController, ClienteController.
  - **Utilitários**: Calculadora, Compactador.
  - **Outros:** views, repositórios, gerenciadores, etc.

Exemplo:

```java
public class Pessoa {
    // Atributos
    private String nome;
    private int idade;
    private String endereco;

    // Construtor
    public Pessoa(String nome, int idade, String endereco) {
        this.nome = nome;
        this.idade = idade;
        this.endereco = endereco;
    }

    // Métodos
    public String getNome() {
        return nome;
    }
    public void setNome(String nome) {
        this.nome = nome;
    }
    public int getIdade() {
        return idade;
    }
    public void setIdade(int idade) {
        this.idade = idade;
    }
    public String getEndereco() {
        return endereco;
    }
    public void setEndereco(String endereco) {
        this.endereco = endereco;
    }
}
```

Instanciação

Uma instância é um objeto criado a partir de uma classe. Quando você cria uma instância de uma classe, o sistema aloca memória para armazenar os dados e métodos daquele objeto.

Alocação de Memória (Stack e Heap):

- **Stack**: é usada para armazenar variáveis locais e referências a objetos. A memória da stack é gerenciada automaticamente e é liberada quando o método termina. É mais rápida e tem um tamanho limitado.
- **Heap**: é usada para armazenar os objetos e suas instâncias. A memória na heap é alocada dinamicamente e é gerenciada pelo garbage collector no Java. Objetos permanecem na heap enquanto forem referenciados.

Ao criar uma instância de uma classe:

1. A referência ao objeto (endereço de memória) é armazenada na **stack,** essa referência aponta para o endereço da memória na **heap.**
2. O próprio objet (dados) é alocado na **heap,** quando o **new** é chamado.

Object e toString

- Em Java, todas as classes são implicitamente subclasses da classe `Object`. Isso significa que, por padrão, elas herdam os métodos definidos em `Object`. Esses métodos podem ser sobrescritos (overridden) para personalizar o comportamento, mas se não forem, as implementações padrão de `Object` serão usadas.
- Métodos da classe Object:
  - `getClass()`: Retorna o tipo do objeto.
  - `equals()`: Compara se dois objetos são iguais.
  - `hashCode()`: Retorna o código hash do objeto.
  - `toString()`: Retorna uma representação em string do objeto.

```
class Pessoa {
    private String nome;
    private int idade;
    private String endereco;

    public Pessoa(String nome, int idade, String endereco) {
        this.nome = nome;
        this.idade = idade;
        this.endereco = endereco;
    }

public class Main {
    public static void main(String[] args) {
        Pessoa pessoa1 = new Pessoa("Leandro", 30, "Rua A");
        Pessoa pessoa2 = new Pessoa("Leandro", 30, "Rua A");

        System.out.println("Classe de pessoa1: " + pessoa1.getClass())
        // Saída: class Pessoa

        System.out.println("pessoa1 é igual a pessoa2? " + pessoa1.equals(pessoa2));
        // Saída: false (porque equals() não foi sobrescrito)

        System.out.println("hashCode de pessoa1: " + pessoa1.hashCode());
        System.out.println("hashCode de pessoa2: " + pessoa2.hashCode());
        // Retorna o hash de pessoa 1 e pessoa2

        System.out.println("pessoa1: " + pessoa1.toString());
        System.out.println("pessoa2: " + pessoa2.toString());
        // Saída: Pessoa@[código hexadecimal] para ambos
        // Para uma saída formatada seria necessário sobrescrita a função
    }
}
```

toString sobrecrito:

```
@Override
    public String toString() {
        return "Pessoa{nome='" + nome + "', idade=" + idade + ", endereco='" + endereco + "'}";
    }
```

Obervação: A chamada da função “toString” não é necessária, basta chamar o objeto na hora de imprimir.

Exemplo:

```
System.out.println("pessoa1: " + pessoa2.toString());
// ou
System.out.println("pessoa2: " + pessoa2);
```

Formatando a saída para 2 casas decimais:

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

        Produto produto = new Produto("Camiseta", 49.99);

        System.out.println(produto.toString());
    }
}

class Produto {
    private String nome;
    private double preco;

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

    @Override
    public String toString() {
        // Formatando o preço para exibir com 2 casas decimais
        return "Produto: " +
        nome + ", Preço: R$" +
        String.format("%.2f", preco);
    }
}
```

Membros Estáticos

- Membros estáticos são métodos que podem ser chamados por outra classe sem que haja a necessidade de instancia-los, sendo chamados pelo próprio nome da classe. Por exemplo, a classe Math, onde seus métodos são estáticos. Ex: "Math.sqrt(double)".
- É frequentemente utilizada em classe utilitárias e em declaração de constantes.

Exemplo:

```
package anotacoes.util;

public class Circulo {
    private static final double PI = 3.14159;
```

```
    public static double getPi() {
        return PI;
    }

    public static double circumference(double raio) {
        return 2*PI*raio;
    }

    public static double volume(double raio) {
        return 4.0 / 3.0 * PI * (raio * raio * raio);
    }
}
```

```
package anotacoes.entities;

import java.util.Locale;
import java.util.Scanner;

public class Main {

    public static void main(String[] args) {
        Locale.setDefault(Locale.US);
        Scanner sc = new Scanner(System.in);

        double raio, circunferencia, volume;

        raio = sc.nextDouble();

        // Chamada das funções estáticas:
        circunferencia = Circulo.circumference(raio);
        volume = Circulo.volume(raio);

        System.out.printf("Circunferencia: %.2f\n", circunferencia);
        System.out.printf("Volume: %.2f\n", volume);
        System.out.println("Valor do PI " + Circulo.getPi());

        sc.close();
    }
```

Para criar uma constante:

```
public static final double PI = 3.14;
```

Observação: Por padrão, os nomes das constantes devem ser escritos em letras maiúsculas.