UNIVERSIDADE DE BRASÍLIA
Faculdade de Ciencias e Tecnologias de Engenharia – Campus Gama
Bacharelado em Engenharia de Software

PROGRAMAÇÃO ORIENTADA A OBJETOS (POO): CONCEITOS FUNDAMENTAIS

Gustavo Antonio Rodrigues e Silva
Matrícula: 242015380
Disciplina: Programação Orientada a Objetos
Professor: Lucas Boaventura

Brasília – DF
Abril de 2025

Programação Orientada a Objetos (POO): Conceitos Fundamentais

1. Definição de Orientação a Objetos (OO)

A **Orientação a Objetos (OO)** é um **paradigma de programação** fundamental para o desenvolvimento de software moderno. Ela se baseia na criação de **objetos**, que são instâncias de classes e encapsulam **atributos** (dados) e **métodos** (comportamentos). Os objetos representam entidades do mundo real ou abstrações, possuindo nome, atributos e comportamento. Este paradigma promove a **reutilização de código**, simplifica a **manutenção** e facilita a **modelagem** de sistemas complexos de forma mais intuitiva, utilizando conceitos como classes, objetos, atributos e métodos para representar abstrações do mundo real.

2. Conceitos Básicos

- **Classe:** Funciona como um "molde" ou "planta" que define os atributos e métodos comuns a um determinado tipo de objeto.
  - *Exemplo:* Uma classe `Carro` pode definir atributos como `marca`, `modelo` e `ano`.
- **Objeto:** É a concretização (instância) de uma classe. Cada objeto possui seus próprios valores para os atributos definidos pela classe.
  - *Exemplo:* Um objeto específico da classe `Carro` poderia ter `marca = "Ferrari"`, `modelo = "F40"` e `ano = 1990`.
- **Atributos:** São as características, propriedades ou dados que descrevem o estado de um objeto. São definidos na classe.
  - *Exemplo:* No caso da classe `Carro`, `marca`, `modelo` e `ano` são atributos.
- **Métodos:** Representam as ações, operações ou comportamentos que um objeto pode realizar. São definidos na classe e operam sobre os atributos do objeto.
  - *Exemplo:* Para a classe `Carro`, métodos poderiam ser `acelerar()`, `frear()` ou `estacionar()`.

3. Exemplo de Classe e Objeto

A seguir, um exemplo simples em `Java` demonstrando a definição de uma classe `Carro` e a criação de um objeto (`meuCarro`) a partir dela:

```java
// Definição da Classe Carro
class Carro {
    String modelo;
    String marca;
    int ano;
}

// Criação e utilização de um Objeto da classe Carro
public class ExemploCarro {
    public static void main(String[] args) {
        Carro meuCarro = new Carro(); // Instanciação do objeto
        meuCarro.modelo = "Fusca";
```

```
        meuCarro.marca = "Volkswagen";
        meuCarro.ano = 1970;         // Atribuições

        System.out.println("Meu carro: " + meuCarro.marca + " " +
meuCarro.modelo + " (" + meuCarro.ano + ")");
    }
}
```

4. Associação entre classes:

Basicamente é um relacionamento entre duas ou mais classes, onde uma `classe` utiliza objetos de outra `classe` para realizar suas operações, onde uma classe pode possuir a outra. Exemplo: `livro` e `professor`:

```
class Livro {
    String titulo;
    String autor;

    Livro(String titulo, String autor) {
        this.titulo = titulo;
        this.autor = autor;
    }
}

class Professor {
    String nome;
    Livro livroDidatico; // Associação: Professor TEM um Livro

    Professor(String nome, Livro livroDidatico) {
        this.nome = nome;
        this.livroDidatico = livroDidatico;
    }

    void ensinar() {
        System.out.println(nome + " está ensinando com o livro \"" +
livroDidatico.titulo + "\" de " + livroDidatico.autor + ".");
    }
}

// uso
public class Main {
    public static void main(String[] args) {
        Livro poo = new Livro("Programação Orientada a Objetos", "Tim
Cook");
        Professor prof = new Professor("Lucas", poo);

        prof.ensinar();
    }
}
```

5. Criação de classe simples:

Classe `Apartamento` em Java, que tem os atributos `area`, `quartos`, `andar`, `valorDeCompra`, `vagasDeGaragem` e `temVaranda`, e um método `exibirInfo()` que imprime esses dados no terminal:

```
public class Main {
    public static void main(String[] args) {
    Apartamento unbAP = new Apartamento();
    unbAP.exibirInfo();
  }
}

public class Apartamento{
    // Atributos
    private double area;
    private int quartos;
    private int andar;
private double valorDeCompra;
private int vagasDeGaragem;
    private boolean temVaranda;

    // Método
public void exibirInfo(){
    System.out.println("Área: " + area + " m²");
    System.out.println("Quartos: " + quartos);
    System.out.println("Andar: " + andar);
    System.out.println("Valor de Compra: R$ " + valorDeCompra);
    System.out.println("Vagas de Garagem: " + vagasDeGaragem);
    System.out.println("Tem Varanda? " + (temVaranda ? "Sim" : "Não"));
  }
}
```

6. Herança:

A **herança** acontece quando uma classe herda atributos/métodos de uma ou mais **superclasses**. No caso do `Java` as classes só podem herdar de uma superclasse, já no `C++` podem existir classes com heranças múltiplas. Porém podem existir múltiplos níveis de heranças entre classes em ambas as linguagens.

a) Herança Simples:

```
// Superclasse
class Heroi {
    String nome;
    void usarPoder() {
```

```
        System.out.println(nome + " usou um poder!");
    }
}

// Subclasse
class HeroiVoador extends Heroi {
    void voar() {
        System.out.println(nome + " está voando!");
    }
}

// Uso
HeroiVoador superman = new HeroiVoador();
superman.nome = "Superman";
superman.usarPoder(); // Herdado de heroi,  saida: “Superman está voando”
superman.voar();
```

b)Herança em múltiplos níveis:

```
    //classe super da super
    class Veiculo {
    String marca;
    int ano;

    void ligar() {
        System.out.println(marca + " está ligado.");
    }
}

//classe super
class Carro extends Veiculo {  // Herda de Veículo
    int portas;

    void abrirPortas() {
        System.out.println("Abrindo " + portas + " portas...");
    }
}

//classe filha
class CarroEsportivo extends Carro {  // Herda de Carro (que herda de
Veículo)
    boolean turbo;

    void ativarTurbo() {
        if (turbo) {
            System.out.println(marca + " turbo ativado! VROOOOM!");
        }
    }
}
```

```
public class Main {
    public static void main(String[] args) {
        CarroEsportivo ferrari = new CarroEsportivo();
        ferrari.marca = "Ferrari";
        ferrari.portas = 2;
        ferrari.turbo = true;
        ferrari.ligar();
        ferrari.abrirPortas();
        ferrari.ativarTurbo();
    }
}
```

7. Polimorfismo - Sobrecarga:

Acontece por meio de métodos de uma classe que **contém o mesmo nome**, mas recebem **parâmetros diferentes**. *Exemplo:*

```
class Calculadora {
    int somar(int a, int b) { return a + b; }  // Recebe int
    double somar(double a, double b) { return a + b; } // Recebe double
}  // Mas no final exercem a mesma função de soma
```

8. Polimorfismo - Sobrescrita:

Acontece quando o `@Override` é utilizado na **subclasse** para **sobrescrever** um método da **superclasse**.

```
class Inimigo {
    void atacar() { System.out.println("Ataque básico!"); }
}
class Chefe extends Inimigo {
    @Override
    void atacar() { System.out.println("Ataque devastador!"); } //
Sobrescrita
```

9. Encapsulamento:

Encapsulamento é o princípio de proteger os dados internos de uma classe, expondo apenas o que é necessário através de métodos controlados `(getters/setters)`. Código da atividade:

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

    Apartamento unbAP = new Apartamento();

    unbAP.setArea(75.5);
    unbAP.setArea(10);
    unbAP.setQuartos(2);
    unbAP.setQuartos(1);


    // exibindo todas as informações
    unbAP.exibirInfo();
  }
}

public class Apartamento {
    double area;
    int quartos;
    int andar;
    double valorDeCompra;
    int vagasDeGaragem;
    boolean temVaranda;

    void exibirInfo() {
        System.out.println("Área: " + area + "m²");
        System.out.println("Quartos: " + quartos);
        System.out.println("Andar: " + andar);
        System.out.println("Valor de Compra: R$ " + valorDeCompra);
        System.out.println("Vagas de Garagem: " + vagasDeGaragem);
        System.out.println("Tem Varanda? " + (temVaranda ? "Sim" :
"Não"));
  }

  //getters
   public double getArea() {
        return area;
    }

    public int getQuartos() {
        return quartos;
    }

    public int getAndar() {
        return andar;
    }

    public double getValorDeCompra() {
        return valorDeCompra;
    }

    public int getVagasDeGaragem() {
        return vagasDeGaragem;
```

```
    }

    public boolean getTemVaranda() {
        return temVaranda;
    }

    //setters
    public void setArea(double area) {
    this.area = area;
    }

    public void setQuartos(int quartos) {
        this.quartos = quartos;
    }

    public void setAndar(int andar) {
        this.andar = andar;
    }

    public void setValorDeCompra(double valorDeCompra) {
        this.valorDeCompra = valorDeCompra;
    }

    public void setVagasDeGaragem(int vagasDeGaragem) {
        this.vagasDeGaragem = vagasDeGaragem;
    }

    public void setTemVaranda(boolean temVaranda) {
        this.temVaranda = temVaranda;
}

}
```

10. Agregação:

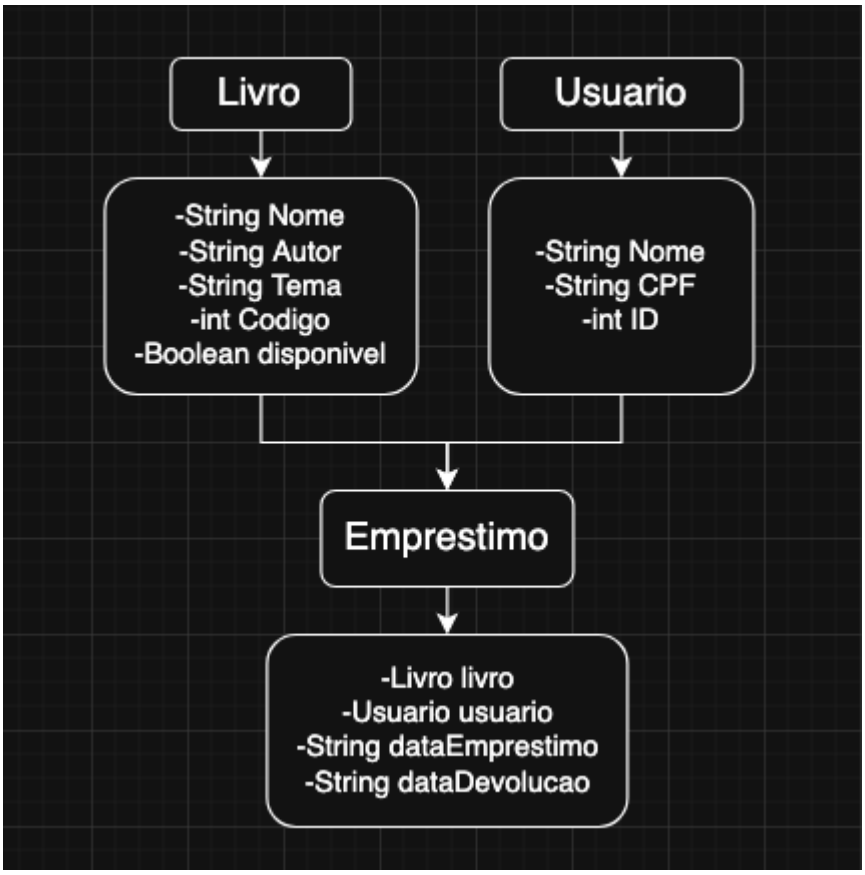
Nesse relacionamento uma classe (todo) contém outra classe (parte), mas a parte pode existir independentemente, sem o todo. Exemplo: Time e Jogador.

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

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

    Departamento fisica = new Departamento("Física");
    Departamento matematica = new Departamento("Matemática");

    Universidade ufmg = new Universidade("UnB");
```

```
        ufmg.addDepartamento(fisica);
        ufmg.addDepartamento(matematica);
        ufmg.listarDepartamentos();

        // Departamento continua existindo sem a universidade
        Departamento fisicaIndependente = fisica;


    }
}

class Departamento {
    private String nome;

    public Departamento(String nome) {
        this.nome = nome;
    }

    public String getNome() {
        return nome;
    }
}

class Universidade {
    private String nome;
    private List<Departamento> departamentos; // Agregação

    public Universidade(String nome) {
        this.nome = nome;
        this.departamentos = new ArrayList<>();
    }

    public void addDepartamento(Departamento depto) {
        departamentos.add(depto);
    }

    public void listarDepartamentos() {
        System.out.println("Departamentos da " + nome + ":");
        for (Departamento depto : departamentos) {
            System.out.println("- " + depto.getNome());
        }
    }
}
```

11. Projeto orientado a objetos:

Aqui está um diagrama representando as classes `Livro`, `Usuario` e `Emprestimo`, feito no site draw.io, sendo que `Emprestimo` tem uma associação com as classes Livro e Usuario:



12. Projeto orientado a objetos 2:

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

    //criando pelo menos um objeto de cada classe
    Livro livro = new Livro( "Machado de Assis", "Dom Casmurro","Nem eu
sei", 123);
    Usuario usuario = new Usuario("João Silva", "000-000-000-00", 1);
    Emprestimo emprestimo = new Emprestimo(livro, usuario, "20/04/2025");

    emprestimo.infoEmprestimo();

    emprestimo.registrarDevolucao("30/04/2025");

    emprestimo.infoEmprestimo();

  }
}

public class Livro {
    private String titulo;
    private String autor;
    private String tema;
    private int codigo;
    private Boolean disponivel;
```

```
    // construtor
    public Livro(String titulo, String autor, String tema, int codigo) {
        this.titulo = titulo;
        this.autor = autor;
        this.tema = tema;
        this.codigo = codigo;
        this.disponivel = true;
    }

    //metodo para emprestar
    public void emprestar() {
        if (disponivel) {
            disponivel = false;
            System.out.println("Livro '" + titulo + "' emprestado com
sucesso!");
        } else {
            System.out.println("Livro '" + titulo + "' já está
emprestado!");
        }
    }

    //metodo para devolver
    public void devolver() {
        if (!disponivel) {
            disponivel = true;
            System.out.println("Livro '" + titulo + "' devolvido com
sucesso!");
        } else {
            System.out.println("Livro '" + titulo + "' já está
disponível!");
        }
    }

    // getters e setters

    public String getTitulo() {
    return titulo;
    }

    public void setTitulo(String titulo) {
        this.titulo = titulo;
    }

    public String getAutor() {
        return autor;
    }

    public void setAutor(String autor) {
        this.autor = autor;
    }

    public String getTema() {
        return tema;
```

```
    }

    public void setTema(String tema) {
        this.tema = tema;
    }

    public int getCodigo() {
        return codigo;
    }

    public void setCodigo(int codigo) {
        this.codigo = codigo;
    }

    public Boolean getDisponivel() {
    return disponivel;
    }

    public void setDisponivel(Boolean disponivel) {
        this.disponivel = disponivel;
    }
}

public class Usuario {
    private String nome;
    private String CPF;
    private int ID;


    // construtor
    public Usuario(String nome, String CPF, int ID) {
        this.nome = nome;
        this.CPF = CPF;
        this.ID = ID;

    }

    // getters e setters
    public String getNome() {
    return nome;
    }

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

    public String getCPF() {
        return CPF;
    }

    public void setCPF(String CPF) {
        this.CPF = CPF;
    }
```

```
    public int getID() {
        return ID;
    }

    public void setID(int ID) {
        this.ID = ID;
    }
}

public class Emprestimo {
    private Livro livro;
    private Usuario usuario;
    private String dataEmprestimo;
    private String dataDevolucao;

    // construtor
    public Emprestimo(Livro livro, Usuario usuario, String dataEmprestimo)
{
        this.livro = livro;
        this.usuario = usuario;
        this.dataEmprestimo = dataEmprestimo;
        this.livro.setDisponivel(false);
    }


    //devolucao
    public void registrarDevolucao(String dataDevolucao) {
        this.dataDevolucao = dataDevolucao;
        System.out.println("Data de devolução: " + dataDevolucao);
        this.livro.setDisponivel(true);
        System.out.println();

    }


    // getters e setters
    public Livro getLivro() {
    return livro;
    }

    public void setLivro(Livro livro) {
        this.livro = livro;
    }

    public Usuario getUsuario() {
        return usuario;
    }

    public void setUsuario(Usuario usuario) {
        this.usuario = usuario;
    }

    public String getDataEmprestimo() {
        return dataEmprestimo;
```

```
    }

    public void setDataEmprestimo(String dataEmprestimo) {
        this.dataEmprestimo = dataEmprestimo;
    }

    public String getDataDevolucao() {
        return dataDevolucao;
    }

    public void setDataDevolucao(String dataDevolucao) {
        this.dataDevolucao = dataDevolucao;
    }

    public void infoEmprestimo(){
        System.out.println("Empréstimo/Devolucão Registrado:");
        System.out.println("Livro: " + this.getLivro().getTitulo());
        System.out.println("Usuário: " + this.getUsuario().getNome());
        System.out.println("Data: " + this.getDataEmprestimo());
    System.out.println("Status: " + (this.getLivro().getDisponivel() ?
"Disponível" : "Emprestado"));
        System.out.println();
    }
}
```