

POO2

Encontro 1

Apresentação da disciplina; Introdução e revisão de conceitos básicos de POO

Classroom: <https://classroom.google.com/c/Nzc5NzY1OTgzODMx?cjc=6rcsj4qo>

Sala: 6rcsj4qo

Referencias da Biblioteca Virtual

- Fundamentos do desenho orientado a objeto com UML
por Meilir Page-Jones (autor), Celso Roberto Paschoa (tradutor), José Davi Furlan (revisor)
<https://plataforma.bvirtual.com.br/Acervo/Publicacao/33>
- Gerenciando projetos de desenvolvimento de software com PMI, RUP e UML
5ª edição por José Carlos Cordeiro Martins (autor), Fabricio Ramirez (colaborador)
<https://plataforma.bvirtual.com.br/Acervo/Publicacao/216084>
- Sistemas orientados a objetos: teoria e prática com UML e Java
por Pablo Rangel (autor), José Gomes de Carvalho Junior (autor)
<https://plataforma.bvirtual.com.br/Acervo/Publicacao/197367>

Ementa da Disciplina

Modelagem computacional de solução de problema no paradigma de programação orientada a objeto. Uso de padrões de projeto (micro arquitetura) na composição de programas de computador. Modelagem objeto relacional. Laboratório de programação.

Objetivos da Disciplina

Capacitar os alunos a compreender e aplicar os principais conceitos e técnicas de qualidade e teste de software, promovendo o desenvolvimento de produtos de software com alto padrão de qualidade. Preparar os alunos para implementar processos de verificação e validação de software, utilizando técnicas de teste automatizado e desenvolvimento orientado a testes, além de introduzi-los aos principais padrões e modelos de melhoria de processo.

Aulas previstas

Aula	Data
Apresentação da disciplina; Introdução e revisão de conceitos básicos	13/08/25
Herança e polimorfismo; Atividade prática: Implementação de herança e polimorfismo	20/08/25
Encapsulamento e abstração; Atividade prática: Exemplos de encapsulamento e abstração	27/08/25
Interfaces e classes abstratas; Atividade prática: Implementação de interfaces e classes abstratas	03/09/25
Exceções e tratamento de erros; atividade prática: Manipulação de exceções	10/09/25
Revisão Avaliação 1	17/09/25

Aulas previstas

Aula	Data
Avaliação 1	24/09/25
Coleções e generics; Atividade prática: Uso de coleções e generics	01/10/25
Reflexão e introspecção; Atividade prática: Aplicação de reflexão	08/10/25
Padrões de projeto OO: Singleton, Factory, Builder, Decorator, Proxy, Strategy, Observer	15/10/25
Relacionamentos entre classes: agregação, composição e especialização.	22/10/25
Persistência de dados e de objetos.	29/10/25
Revisão Avaliação 2	05/11/25

Aulas previstas

Aula	Data
Avaliação 2	12/11/25
Entrega de Resultados	19/11/25
Reposição	26/11/25
Revisão de dados	03/12/25

Objetivos do Encontro

- **Apresentação da disciplina; Introdução e revisão de conceitos básicos de POO.**
- Discutir exemplos práticos com UML e código.
- Fixar conteúdos com exercícios práticos.

Introdução

Nesta aula abordaremos:

- Definições fundamentais
- Benefícios e aplicações
- Boas práticas

Classe e Objeto

Conteúdo teórico detalhado sobre **Classe**:

- **Definições:** Classe é um modelo ou template que define as características (atributos) e comportamentos (métodos) que os objetos desse tipo terão. É como uma "planta" para criar objetos.
- Exemplos práticos
- Diagramas UML quando aplicável

Exemplo prático – Classe

```
// Definição de uma classe simples
public class Pessoa {
    private String nome;
    private int idade;

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

    // Método
    public void apresentar() {
        System.out.println("Sou " + nome + ", " + idade + " anos");
    }

    // Getter
    public String getNome() { return nome; }
}
```

Objeto

Conteúdo teórico detalhado sobre **Objeto**.

- **Definições:** Objeto é uma instância de uma classe, uma entidade concreta criada a partir do modelo da classe. Cada objeto possui seu próprio estado (valores dos atributos) e pode executar os comportamentos definidos na classe.
- Exemplos práticos
- Diagramas UML quando aplicável

Exemplo prático – Objeto

```
public class ExemploObjeto {  
    public static void main(String[] args) {  
        // Criando objetos (instâncias)  
        Pessoa p1 = new Pessoa("João", 25);  
        Pessoa p2 = new Pessoa("Maria", 30);  
  
        // Usando os objetos  
        p1.apresentar(); // Sou João, 25 anos  
        p2.apresentar(); // Sou Maria, 30 anos  
  
        // Cada objeto tem seu próprio estado  
        System.out.println(p1.getNome()); // João  
        System.out.println(p2.getNome()); // Maria  
    }  
}
```

Atributos

Conteúdo teórico detalhado sobre **Atributos**.

- **Definições:** Atributos são variáveis que representam as características ou propriedades de uma classe/objeto. Podem ser de instância (cada objeto tem sua própria cópia) ou de classe/estáticos (compartilhados por todos os objetos).
- Exemplos práticos
- Diagramas UML quando aplicável

Exemplo prático – Atributos

```
public class Conta {  
    // Atributos de instância  
    private String numero;  
    private double saldo;  
  
    // Atributo de classe (compartilhado)  
    private static int totalContas = 0;  
  
    public Conta(String numero) {  
        this.numero = numero;  
        this.saldo = 0.0;  
        totalContas++; // Incrementa contador  
    }  
  
    public void depositar(double valor) {  
        saldo += valor;  
    }  
  
    public static int getTotalContas() {  
        return totalContas;  
    }  
}
```

Métodos

Conteúdo teórico detalhado sobre **Métodos**.

- **Definições:** Métodos são funções que definem os comportamentos de uma classe/objeto. Representam as ações que o objeto pode realizar, podendo receber parâmetros, retornar valores e modificar o estado do objeto.
- Exemplos práticos
- Diagramas UML quando aplicável

Exemplo prático – Métodos

```
public class Calculadora {  
  
    // Método sem retorno  
    public void ligar() {  
        System.out.println("Calculadora ligada!");  
    }  
  
    // Método com retorno  
    public double somar(double a, double b) {  
        return a + b;  
    }  
  
    // Método estático  
    public static double elevar(double base, int exp) {  
        return Math.pow(base, exp);  
    }  
  
    // Sobrecarga de métodos  
    public double multiplicar(double a, double b) {  
        return a * b;  
    }  
  
    public double multiplicar(double a, double b, double c) {  
        return a * b * c;  
    }  
}
```

Encapsulamento

Conteúdo teórico detalhado sobre **Encapsulamento**.

- **Definições:** Encapsulamento é o princípio de ocultar os detalhes internos de implementação de uma classe, controlando o acesso aos atributos através de métodos públicos (getters/setters), garantindo proteção e integridade dos dados.
- Exemplos práticos
- Diagramas UML quando aplicável

Exemplo prático – Encapsulamento

```
public class ContaCorrente {  
    // Atributos privados (encapsulados)  
    private double saldo;  
    private String senha;  
  
    public ContaCorrente(String senha) {  
        this.senha = senha;  
        this.saldo = 0.0;  
    }  
  
    // Acesso controlado ao saldo  
    public double getSaldo(String senhaInformada) {  
        if (senha.equals(senhaInformada)) {  
            return saldo;  
        }  
        return -1; // Senha incorreta  
    }  
  
    // Depósito com validação  
    public boolean depositar(double valor) {  
        if (valor > 0) {  
            saldo += valor;  
            return true;  
        }  
        return false;  
    }  
}
```

Herança

Conteúdo teórico detalhado sobre **Herança**.

- **Definições:** Herança é o mecanismo que permite criar uma nova classe (filha/subclasse) baseada em uma classe existente (pai/superclasse), herdando seus atributos e métodos, promovendo reutilização de código e hierarquias de classes.
- Exemplos práticos
- Diagramas UML quando aplicável

Exemplo prático – Herança

```
// Classe pai
public class Veiculo {
    protected String marca;
    protected boolean ligado;

    public Veiculo(String marca) {
        this.marca = marca;
        this.ligado = false;
    }

    public void ligar() {
        ligado = true;
        System.out.println(marca + " ligado!");
    }

    public void acelerar() {
        System.out.println("Acelerando...");
    }
}

// Classe filha
public class Carro extends Veiculo {
    private int portas;

    public Carro(String marca, int portas) {
        super(marca); // Chama construtor pai
        this.portas = portas;
    }

    @Override
    public void acelerar() {
        System.out.println("Carro acelerando suavemente");
    }
}
```

Polimorfismo

Conteúdo teórico detalhado sobre **Polimorfismo**.

- **Definições:** Polimorfismo é a capacidade de objetos de diferentes classes responderem de forma específica a uma mesma mensagem/método. Permite que um mesmo código trabalhe com diferentes tipos de objetos de forma transparente.
- Exemplos práticos
- Diagramas UML quando aplicável

Exemplo prático – Polimorfismo

```
// Classe abstrata
abstract class Animal {
    protected String nome;

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

    public abstract void emitirSom(); // Método abstrato
}

// Subclasses
class Cachorro extends Animal {
    public Cachorro(String nome) { super(nome); }

    @Override
    public void emitirSom() {
        System.out.println(nome + ": Au au!");
    }
}

class Gato extends Animal {
    public Gato(String nome) { super(nome); }

    @Override
    public void emitirSom() {
        System.out.println(nome + ": Miau!");
    }
}

// Polimorfismo em ação
Animal[] animais = {new Cachorro("Rex"), new Gato("Mimi")};
for (Animal a : animais) {
    a.emitirSom(); // Comportamento específico
}
```

Abstração

Conteúdo teórico detalhado sobre **Abstração**.

- **Definições:** Abstração é o processo de identificar características essenciais de um objeto, ignorando detalhes irrelevantes. Permite criar modelos simplificados através de classes abstratas e interfaces, focando no "o que" fazer ao invés de "como" fazer.
- Exemplos práticos
- Diagramas UML quando aplicável

Exemplo prático – Abstração

```
// Interface (abstração pura)
interface Forma {
    double calcularArea();
    void desenhar();
}

// Classe abstrata (abstração parcial)
abstract class FormaGeometrica implements Forma {
    protected String cor;

    public FormaGeometrica(String cor) {
        this.cor = cor;
    }

    public abstract void redimensionar(double fator);
}

// Implementação concreta
class Retangulo extends FormaGeometrica {
    private double largura, altura;

    public Retangulo(String cor, double l, double a) {
        super(cor);
        this.largura = l; this.altura = a;
    }

    @Override
    public double calcularArea() { return largura * altura; }

    @Override
    public void desenhar() { System.out.println("Desenhando retângulo " + cor); }

    @Override
    public void redimensionar(double f) { largura *= f; altura *= f; }
}
```

Mensagens e Interação

Conteúdo teórico detalhado sobre **Mensagens e Interação**.

- **Definições:** Mensagens são chamadas de métodos entre objetos que permitem a comunicação e colaboração. A interação representa como objetos trabalham juntos para realizar tarefas complexas, trocando informações e coordenando ações.
- Exemplos práticos
- Diagramas UML quando aplicável

Exemplo prático – Mensagens e Interação

```
// Classe Cliente
class Cliente {
    private String nome;

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

    // Enviar mensagem
    public void fazerPedido(Restaurante restaurante, String prato) {
        System.out.println(nome + " fazendo pedido...");
        restaurante.receberPedido(this, prato);
    }

    // Receber mensagem
    public void receberConfirmacao(String msg) {
        System.out.println(nome + " recebeu: " + msg);
    }

    public String getNome() { return nome; }
}

// Classe Restaurante
class Restaurante {
    public void receberPedido(Cliente cliente, String prato) {
        System.out.println("Pedido recebido: " + prato);
        // Responde ao cliente
        cliente.receberConfirmacao("Pedido confirmado!");
    }
}
```

UML – Visão Geral

Conteúdo teórico detalhado sobre **UML – Visão Geral**.

- **Definições:** UML (Unified Modeling Language) é uma linguagem de modelagem padrão para visualizar, especificar, construir e documentar sistemas orientados a objetos através de diagramas que representam diferentes aspectos do sistema.
- Exemplos práticos
- Diagramas UML quando aplicável

Exemplo prático – UML – Visão Geral



```
public class Biblioteca {
    private String nome;
    private List<Livro> livros;

    public Biblioteca(String nome) {
        this.nome = nome;
        this.livros = new ArrayList<>();
    }

    public void addLivro(Livro livro) {
        livros.add(livro);
    }

    public Livro buscarLivro(String titulo) {
        return livros.stream()
            .filter(l -> l.getTitulo().equals(titulo))
            .findFirst().orElse(null);
    }
}

// Relacionamento: Biblioteca ◇— Livro (agregação)
public class Livro {
    private String titulo;
    private String autor;

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

    public String getTitulo() { return titulo; }
}
```

Exercício – Classe

Exercício Prático:

1. Crie uma classe `Produto` com os atributos:

- `nome` (String)
- `preco` (double)
- `categoria` (String)

2. Implemente:

- Construtor que receba todos os parâmetros
- Método `aplicarDesconto(double percentual)` que reduza o preço
- Método `mostrarInfo()` que exiba as informações do produto
- Getters para todos os atributos

Exercício – Objeto

Exercício Prático:

1. Utilizando a classe `Produto` criada anteriormente:
 - Crie 3 objetos diferentes com produtos de categorias distintas
 - Aplique descontos diferentes para cada produto (5%, 15%, 20%)
 - Exiba as informações de todos os produtos
2. Demonstre que cada objeto mantém seu próprio estado:
 - Modifique o preço de apenas um produto
 - Verifique que os outros produtos não foram afetados
3. Crie um array de produtos e percorra exibindo as informações

Exercício – Atributos

Exercício Prático:

1. Crie uma classe `Funcionario` com:

- Atributos de instância: `nome`, `salario`, `departamento`
- Atributo de classe: `totalFuncionarios` (contador estático)
- Atributo de classe: `salarioMinimo` (valor fixo)

2. Implemente:

- Construtor que incremente o contador de funcionários
- Método para aumentar salário (só se for \geq salário mínimo)
- Método estático `getTotalFuncionarios()`
- Método estático `setSalarioMinimo(double valor)`

Exercício – Métodos

Exercício Prático:

1. Crie uma classe `ConversorTemperatura` com métodos estáticos:

- `celsiusParaFahrenheit(double celsius)`
- `fahrenheitParaCelsius(double fahrenheit)`
- `celsiusParaKelvin(double celsius)`

2. Implemente sobrecarga de métodos na classe `Retangulo`:

- `calcularArea()` - sem parâmetros (usa atributos)
- `calcularArea(double largura, double altura)` - método estático
- `redimensionar(double fator)` - multiplica por fator
- `redimensionar(double novaLargura, double novaAltura)` - define novos valores

Exercício – Encapsulamento

Exercício Prático:

1. Crie uma classe `ContaPoupanca` com atributos privados:

- `saldo` , `numeroAgencia` , `numeroConta` , `senha`

2. Implemente métodos públicos com validações:

- `depositar(double valor)` - apenas valores > 0
- `sacar(double valor, String senha)` - validar senha e saldo suficiente
- `getSaldo(String senha)` - retorna saldo só com senha correta
- `alterarSenha(String senhaAtual, String novaSenha)` - valida senha atual

3. Demonstre que não é possível acessar diretamente os atributos privados

Exercício – Herança

Exercício Prático:

1. Crie uma hierarquia de classes:

- **Classe pai:** `Funcionario` com `nome`, `salario`, `id`
- **Classes filhas:** `FuncionarioCLT` e `FuncionarioTerceirizado`

2. Implemente na classe pai:

- Construtor, getters e método `calcularSalarioLiquido()`

3. Nas classes filhas:

- `FuncionarioCLT`: adicione `beneficios` e sobrescreva o cálculo do salário líquido
- `FuncionarioTerceirizado`: adicione `valorHora` e `horasTrabalhadas`

Exercício – Mensagens e Interação

Exercício Prático:

1. Crie um sistema de **Loja Online** com as classes:

- **Cliente** : pode fazer pedidos e receber notificações
- **Loja** : processa pedidos e comunica com estoque
- **Estoque** : verifica disponibilidade e reserva produtos
- **Produto** : representa itens no estoque

2. Implemente as interações:

- Cliente faz pedido → Loja verifica estoque → Estoque confirma/nega
- Loja notifica cliente sobre status do pedido
- Se aprovado: Estoque reduz quantidade disponível

Exercício – Polimorfismo

Exercício Prático:

1. Crie uma classe abstrata `Veiculo` com:

- Atributos: `marca`, `ano`, `preco`
- Método abstrato: `calcularImposto()`
- Método concreto: `exibirInfo()`

2. Crie subclasses concretas:

- `Carro` : imposto = 5% do preço
- `Moto` : imposto = 3% do preço
- `Caminhao` : imposto = 8% do preço

3. Demonstre polimorfismo:

Exercício – UML – Visão Geral

Exercício Prático:

1. **Análise de Código:** Para o sistema Biblioteca-Livro apresentado:

- Desenhe o diagrama de classes UML completo
- Identifique os tipos de relacionamento (agregação, composição, etc.)
- Inclua visibilidade (+, -, #) para todos os membros

2. **Implementação:** Crie o sistema completo baseado no diagrama:

- Adicione classe `Usuario` que pode emprestar livros
- Implemente relacionamento: Biblioteca ← Usuario → Livro
- Adicione regras: limite de 3 livros por usuário

3. **Validação:**

Referências

- Larman, C. *Utilizando UML e Padrões*. Saraiva.
- Fundamentos do desenho orientado a objeto com UML
por Meilir Page-Jones (autor), Celso Roberto Paschoa (tradutor), José Davi Furlan (revisor)
<https://plataforma.bvirtual.com.br/Acervo/Publicacao/33>
- Gerenciando projetos de desenvolvimento de software com PMI, RUP e UML
5ª edição por José Carlos Cordeiro Martins (autor), Fabricio Ramirez (colaborador)
<https://plataforma.bvirtual.com.br/Acervo/Publicacao/216084>
- Sistemas orientados a objetos: teoria e prática com UML e Java
por Pablo Rangel (autor), José Gomes de Carvalho Junior (autor)
<https://plataforma.bvirtual.com.br/Acervo/Publicacao/197367>