Introdução ao Curso de Programação Orientada a Objetos com Java(12H)

1. Apresentação do Curso

Este curso tem como objetivo ensinar os conceitos fundamentais da **Programação Orientada a Objetos (POO)** utilizando **Java**. Ao longo do curso, abordaremos desde a instalação do ambiente de desenvolvimento até a criação de aplicações completas, utilizando conceitos como **classes, objetos, herança, polimorfismo, encapsulamento e abstração**.

1.1 O que é Programação Orientada a Objetos?

A **POO** é um paradigma de programação baseado na estruturação do código em **objetos**, que representam entidades do mundo real. Cada objeto tem seus **atributos** (dados) e **métodos** (comportamentos). Isso facilita o desenvolvimento de software modular, reutilizável e escalável.

Exemplo prático:

- Em um sistema de gerenciamento de alunos, cada **Aluno** pode ser representado como um **objeto**, contendo informações como nome, matrícula e nota.

---

2. Configuração do Ambiente de Desenvolvimento

Antes de começarmos a programar, precisamos configurar nosso ambiente de desenvolvimento.

2.1 Instalando o JDK (Java Development Kit)

O **JDK** é o kit de desenvolvimento necessário para compilar e executar programas Java.

1. Acesse o site oficial do Java: https://www.oracle.com/java/technologies/javase-downloads.html
2. Baixe e instale a versão mais recente do **JDK** compatível com seu sistema operacional.
3. Configure a variável de ambiente `JAVA_HOME`:
    - **Windows**: Vá até Painel de Controle → Sistema → Configurações Avançadas → Variáveis de Ambiente, crie uma variável `JAVA_HOME` apontando para o diretório de instalação do JDK.

- **Linux/Mac**: Adicione a linha `export JAVA_HOME=/caminho/do/jdk` ao arquivo `.bashrc` ou `.zshrc`.

Para verificar a instalação, abra o terminal (ou prompt de comando) e digite:

```
java -version
```

4. Se estiver tudo certo, será exibida a versão do Java instalada.

2.2 Instalando uma IDE

Uma **IDE (Integrated Development Environment)** facilita a escrita e execução do código Java. As mais recomendadas são:

- **Eclipse**: https://www.eclipse.org/downloads/
- **IntelliJ IDEA**: https://www.jetbrains.com/idea/
- **NetBeans**: https://netbeans.apache.org/
- VSCode: Java Extension Package

Após instalar a IDE de sua escolha, crie um novo projeto Java.

---

3. Primeiro Programa em Java

Vamos começar com um programa simples para entender a estrutura básica de um código Java.

```
public class OlaMundo {
    public static void main(String[] args) {
        System.out.println("Olá, Mundo!");
    }
}
```

Explicação do código:

- `public class OlaMundo`: Define uma **classe** chamada `OlaMundo`.
- `public static void main(String[] args)`: Método principal onde a execução do programa começa.
- `System.out.println("Olá, Mundo!");`: Exibe uma mensagem no console.

Para executar esse código:

1. Salve o arquivo como `OlaMundo.java`.

No terminal, compile com:

```
javac OlaMundo.java
```

2.

Execute com:

```
java OlaMundo
```

A saída será:

```
Olá, Mundo!
```

---

4. Introdução à Programação Orientada a Objetos

Agora que temos o ambiente configurado, vamos entender os pilares da POO.

4.1 Conceitos Fundamentais da POO

1. **Classes e Objetos**
   - **Classe**: Modelo para criação de objetos.
   - **Objeto**: Instância de uma classe, contendo atributos e métodos.
2. **Encapsulamento**
   - Controle do acesso aos atributos e métodos através de modificadores (`private`, `public`, `protected`).
3. **Herança**
   - Permite que uma classe derive atributos e comportamentos de outra.
4. **Polimorfismo**
   - Permite que métodos tenham diferentes comportamentos dependendo da classe que os implementa.
5. **Abstração**
   - Define comportamentos genéricos sem necessidade de implementação concreta.

---

5. Criando Classes e Objetos

Vamos criar um exemplo prático de **classe e objeto** em Java.

Exemplo: Criando uma classe `Carro`

```
public class Carro {
    String modelo;
    int ano;

    void exibirInformacoes() {
        System.out.println("Modelo: " + modelo);
        System.out.println("Ano: " + ano);
    }
}
```

Agora, vamos criar um **objeto** dessa classe no `main`:

```
public class TesteCarro {
    public static void main(String[] args) {
        Carro meuCarro = new Carro();
        meuCarro.modelo = "Honda Civic";
        meuCarro.ano = 2022;

        meuCarro.exibirInformacoes();
    }
}
```

Saída esperada:

```
Modelo: Honda Civic
Ano: 2022
```

---

6. Implementando Herança

Agora, vamos criar uma classe `CarroEletrico`, que **herda** de `Carro`.

```
class CarroEletrico extends Carro {
    int autonomia;

    void exibirAutonomia() {
        System.out.println("Autonomia: " + autonomia + " km");
    }
}
```

E no `main`, criamos um objeto da classe `CarroEletrico`:

```
public class TesteHeranca {
    public static void main(String[] args) {
        CarroEletrico tesla = new CarroEletrico();
        tesla.modelo = "Tesla Model S";
        tesla.ano = 2023;
        tesla.autonomia = 600;

        tesla.exibirInformacoes();
        tesla.exibirAutonomia();
    }
}
```

Saída esperada:

```
Modelo: Tesla Model S
Ano: 2023
Autonomia: 600 km
```

---

7. Conclusão

Neste primeiro módulo, aprendemos:

- Como configurar o ambiente Java (JDK e IDE).

- Como criar e executar um programa Java.
- Os conceitos fundamentais de POO.
- Como criar **classes e objetos**.
- Como aplicar **herança** em Java.

Nos próximos módulos, exploraremos **encapsulamento, polimorfismo, interfaces e boas práticas**.

---

8. Exercícios

0. Sistema de Gestão de Cursos Descrição: Crie um sistema que gerencie a estrutura de uma escola ou universidade, incluindo professores, alunos e cursos. A ideia é aplicar herança, polimorfismo e encapsulamento.

Instruções:

- Crie uma classe Pessoa com atributos como nome e cpf, e métodos como exibirInformacoes().
- Crie subclasses Aluno e Professor.
- Aluno deve conter um atributo matricula e nota, enquanto Professor deve ter um atributo salario. Ambos devem sobrescrever o método exibirInformacoes().
- Crie uma classe Curso com uma lista de Alunos e um Professor associado.
- O curso deve ter métodos para adicionar alunos e exibir informações detalhadas sobre o curso. Adicione um método na classe Curso para calcular a média de notas dos alunos e determinar o aprovado e reprovado.

Requisitos Avançados:

- Utilize polimorfismo para o método exibirInformacoes() de Aluno e Professor.
- Implemente encapsulamento nos atributos da classe Pessoa, utilizando getters e setters.
- Crie uma interface Avaliavel com o método avaliarDesempenho() e implemente-a nas classes Aluno.

Resposta:

```java
// Classe Pessoa
public abstract class Pessoa {
    //atributos
    private String nome;
    private String cpf;

    //construtor
    public Pessoa(String nome, String cpf) {
        this.nome = nome;
        this.cpf = cpf;
    }

    //getter and setter
    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;
    }
    //métodos exibir informações
    public void exibirInformacoes() {
        System.out.println("Nome: " + nome);
        System.out.println("CPF: " + cpf);
    }
}

// Interface Avaliavel
interface Avaliavel {
    //método abstrato
    void avaliarDesempenho();
}
```

```java
// Classe Aluno
public class Aluno extends Pessoa implements Avaliavel {
    //atributos
    private String matricula;
    private double nota;
    //construtor
    public Aluno(String nome, String cpf, String matricula, double nota) {
        super(nome, cpf);
        this.matricula = matricula;
        this.nota = nota;
    }
    //getters and setters
    public String getMatricula() {
        return matricula;
    }
    public void setMatricula(String matricula) {
        this.matricula = matricula;
    }
    public double getNota() {
        return nota;
    }
    public void setNota(double nota) {
        this.nota = nota;
    }
    //reescrita do método da classe Pessoas
    @Override
    public void exibirInformacoes() {
        super.exibirInformacoes();
        System.out.println("Matrícula: " + matricula);
        System.out.println("Nota: " + nota);
    }

    @Override
    public void avaliarDesempenho() {
        if (nota >= 6.0) {
            System.out.println("Aluno aprovado.");
        } else {
            System.out.println("Aluno reprovado.");
        }
    }
}
```

```java
// Classe Professor
public class Professor extends Pessoa {
    private double salario;

    public Professor(String nome, String cpf, double salario) {
        super(nome, cpf);
        this.salario = salario;
    }

    public double getSalario() {
        return salario;
    }

    public void setSalario(double salario) {
        this.salario = salario;
    }

    @Override
    public void exibirInformacoes() {
        super.exibirInformacoes();
        System.out.println("Salário: R$ " + salario);
    }
}
```

```java
// Classe Curso
import java.util.ArrayList;
import java.util.List;

public class Curso {
    private String nomeCurso;
    private Professor professor;
    private List<Aluno> alunos;

    public Curso(String nomeCurso, Professor professor) {
        this.nomeCurso = nomeCurso;
        this.professor = professor;
        this.alunos = new ArrayList<>();
    }

    public void adicionarAluno(Aluno aluno) {
        alunos.add(aluno);
    }

    public void exibirInformacoesCurso() {
        System.out.println("Curso: " + nomeCurso);
        System.out.println("Professor Responsável:");
        professor.exibirInformacoes();
        System.out.println("\nLista de Alunos:");
        for (Aluno aluno : alunos) {
            aluno.exibirInformacoes();
            aluno.avaliarDesempenho();
            System.out.println("----------------------");
        }
    }

    public double calcularMediaNotas() {
        double soma = 0;
        for (Aluno aluno : alunos) {
            soma += aluno.getNota();
        }
        return alunos.isEmpty() ? 0 : soma / alunos.size();
    }
}
```

```java
// Classe Principal com Menu
import java.util.Scanner;

public class SistemaGestaoCursos {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        Professor professor = new Professor("Carlos Silva", "123.456.789-00", 4500.0);
        Curso curso = new Curso("Programação Orientada a Objetos", professor);

        int opcao;
        do {
            System.out.println("\n=== Sistema de Gestão de Cursos ===");
            System.out.println("1. Adicionar Aluno");
            System.out.println("2. Consultar Informações do Curso");
            System.out.println("3. Consultar Média das Notas");
            System.out.println("4. Sair");
            System.out.print("Escolha uma opção: ");
            opcao = scanner.nextInt();
            scanner.nextLine(); // Limpar o buffer

            switch (opcao) {
                case 1:
                    System.out.print("Nome do aluno: ");
                    String nome = scanner.nextLine();
                    System.out.print("CPF do aluno: ");
                    String cpf = scanner.nextLine();
                    System.out.print("Matrícula do aluno: ");
                    String matricula = scanner.nextLine();
                    System.out.print("Nota do aluno: ");
                    double nota = scanner.nextDouble();
                    scanner.nextLine();

                    Aluno novoAluno = new Aluno(nome, cpf, matricula, nota);
                    curso.adicionarAluno(novoAluno);
                    System.out.println("Aluno adicionado com sucesso!");
                    break;

                case 2:
                    curso.exibirInformacoesCurso();
                    break;
```

```
                case 3:
                    System.out.println("Média das notas da turma: " + curso.calcularMediaNota
                    break;

                case 4:
                    System.out.println("Saindo do sistema. Até logo!");
                    break;

                default:
                    System.out.println("Opção inválida. Tente novamente.");
            }
        } while (opcao != 4);

        scanner.close();
    }
}
```

---

1. Sistema de Gerenciamento de Biblioteca

Descrição: Crie um sistema para gerenciar livros, autores e usuários de uma biblioteca.
Instruções:

- Crie uma classe `Pessoa` com atributos `nome` e `cpf`.
- Crie subclasses `Usuario` e `Autor`, onde `Usuario` possui um atributo `numeroDeRegistro` e `Autor` possui um atributo `biografia`.
- Crie uma classe `Livro` com título, autor e uma lista de usuários que pegaram o livro emprestado.
- Implemente métodos para: adicionar livros, emprestar livros para usuários e exibir informações dos empréstimos.
- **Requisito avançado:** Crie uma interface `Emprestavel` com o método `emprestar()` e implemente-a na classe `Livro`.

---

2. Sistema de Controle de Vendas

Descrição: Desenvolva um sistema para controlar as vendas de uma loja, com foco em produtos, clientes e vendedores.
Instruções:

- Crie uma classe `Pessoa` com atributos básicos (`nome`, `cpf`).

- Crie subclasses `Cliente` e `Vendedor`. O `Cliente` terá um atributo `limiteDeCredito` e o `Vendedor` terá um `salario` e `comissão`.
- Crie uma classe `Venda` que contém informações sobre o produto, o cliente e o vendedor.
- Implemente métodos para registrar vendas, calcular o total de vendas por vendedor e exibir relatórios de vendas.
- **Requisito avançado:** Crie uma interface `Comissionavel` com o método `calcularComissao()` e implemente-a na classe `Vendedor`.

---

3. Sistema de Gestão de Projetos

Descrição: Crie um sistema para gerenciar projetos de uma empresa, com equipes e tarefas.
Instruções:

- Crie uma classe `Pessoa` com atributos `nome` e `email`.
- Crie subclasses `Gerente` e `Funcionario`, onde o `Gerente` possui um atributo `departamento` e o `Funcionario` possui um `cargo`.
- Crie uma classe `Projeto` que gerencia uma lista de funcionários e um gerente responsável.
- Implemente métodos para adicionar tarefas ao projeto, atribuir tarefas aos funcionários e exibir o progresso do projeto.
- **Requisito avançado:** Crie uma interface `Avaliavel` com o método `avaliarProgresso()` e implemente-a na classe `Projeto`.

---

4. Sistema de Controle de Frota de Veículos

Descrição: Desenvolva um sistema para controlar veículos de uma empresa, incluindo motoristas e manutenções.
Instruções:

- Crie uma classe `Pessoa` com atributos `nome` e `cnh`.
- Crie uma classe `Motorista` que herda de `Pessoa`, adicionando o atributo `categoriaCNH`.
- Crie uma classe `Veiculo` com informações como `placa`, `modelo` e uma lista de manutenções realizadas.
- Implemente métodos para cadastrar veículos, registrar manutenções e associar motoristas aos veículos.

- **Requisito avançado:** Crie uma interface `Manutenivel` com o método `realizarManutencao()` e implemente-a na classe `Veiculo`.

---

5. Sistema de Gestão de Consultório Médico

Descrição: Crie um sistema para gerenciar pacientes, médicos e consultas em um consultório.
Instruções:

- Crie uma classe `Pessoa` com atributos `nome` e `cpf`.
- Crie subclasses `Paciente` e `Medico`, onde `Paciente` possui um atributo `historicoMedico` e `Medico` possui `especialidade` e `crm`.
- Crie uma classe `Consulta` que associa um paciente a um médico, com informações sobre a data e o diagnóstico.
- Implemente métodos para agendar consultas, registrar diagnósticos e exibir o histórico médico dos pacientes.
- **Requisito avançado:** Crie uma interface `Agendavel` com o método `agendarConsulta()` e implemente-a na classe `Consulta`.