Programação avançada – AULA 02

Matheus Moresco

Análise e Desenvolvimento de Sistemas - 5º Período

2025/01

Revisão da ultima aula

- **Programação Avançada** aprofunda os conceitos essenciais para o desenvolvimento de sistemas robustos, escaláveis e eficientes.
- **Ementa** da disciplina
- Vantagens do **Java.**
- Instalação do **JDK** e **Eclipse.**

Introdução à Programação Orientada a Objetos (POO)

O que é POO?

- Paradigma baseado em objetos que representam entidades do mundo real.
- Facilita reutilização de código, organização e manutenção.
- Amplamente usado em aplicações escaláveis e complexas.

Introdução à Programação Orientada a Objetos (POO)

Por que usar POO?

- Modularidade: Código organizado em pequenas partes reutilizáveis.
- Reutilização: Uso de classes já existentes.
- Facilidade de manutenção: Melhor organização e menos repetição de código.

Estrutura Básica de uma Classe em Java

- Uma **classe** é a estrutura fundamental na Programação Orientada a Objetos. Ela define um modelo para a criação de **objetos**, especificando seus atributos (dados) e métodos (comportamentos).

```
public class NomeDaClasse {
    // Atributos (dados da classe)
    tipo nomeAtributo;

    // Métodos (comportamentos da classe)
    public void nomeMetodo() {
        // Implementação do método
    }
}
```

Métodos e Atributos

- **Atributos:** São variáveis que armazenam o estado do objeto.
- **Métodos:** São funções dentro da classe que definem seu comportamento.

```
2
3  public class Pessoa {
4      // Atributos
5      String nome;
6      int idade;
7
8      // Método
9      public void exibirInformacoes() {
10         System.out.println("Nome: " + nome);
11         System.out.println("Idade: " + idade);
12     }
13 }
14
```

Instanciação de Objetos

- A instanciação é o processo de criar um objeto a partir de uma classe usando a palavra-chave ***new***.

```
2
3 public class main {
4     static void main(String[] args) {
5         //Criando um objeto
6         Pessoa pessoa = new Pessoa();
7
8         //Definindo valores para os Atributos
9         pessoa.nome = "João da Silva";
10        pessoa.idade = 22;
11        //Chamando Método para exibir informações
12        pessoa.exibirInformacoes();
13    }
14 }
15
```

Princípios da POO - Encapsulamento

- O **encapsulamento** é um dos princípios da Programação Orientada a Objetos (POO) que restringe o acesso direto aos dados internos de um objeto e expõe apenas métodos seguros para manipular esses dados.

```
 2
 3 public class Pessoa {
 4     private String nome;
 5
 6     public void setNome(String nome) {
 7         this.nome = nome;
 8     }
 9
10     public String getNome() {
11         return nome;
12     }
13 }
```

Princípios da POO - Herança

- A herança é um dos pilares da Programação Orientada a Objetos (POO) e permite que uma classe (filha) reutilize atributos e métodos de outra classe (pai).

```
2
3  public class Pessoa {
4      protected String nome;
5      protected int idade;
6
7      public Pessoa(String nome, int idade) {
8          this.nome = nome;
9          this.idade = idade;
10     }
11
12     public void exibirInformacoes() {
13         System.out.println("Nome: " + nome);
14         System.out.println("Idade: " + idade);
15     }
16 }
17
```

Princípios da POO - Herança

```
 2
 3 public class Aluno extends Pessoa {
 4  private String curso;
 5
 6  public Aluno(String nome, int idade, String curso) {
 7      this.nome = nome;
 8      this.idade = idade;
 9      this.curso = curso;
10  }
11 }
12
```

```
 2
 3 public class Professor extends Pessoa {
 4  private String disciplina;
 5
 6  public Professor(String nome, int idade, String disciplina) {
 7      this.nome = nome;
 8      this.idade = idade;
 9      this.disciplina = disciplina;
10  }
11 }
12
```

Princípios da POO - Polimorfismo

- O polimorfismo permite que um mesmo método tenha comportamentos diferentes dependendo do objeto que o utiliza. Isso é possível por meio de **métodos sobrescritos** em subclasses.

```
2
3  public class Aluno extends Pessoa {
4      private String curso;
5
6      public Aluno(String nome, int idade, String curso) {
7          this.nome = nome;
8          this.idade = idade;
9          this.curso = curso;
10     }
11
12     @Override
13     public void exibirInformacoes() {
14         System.out.println("Nome: " + nome);
15         System.out.println("Idade: " + idade);
16         System.out.println("Curso: " + curso);
17     }
18 }
19
```

```
2
3  public class Professor extends Pessoa {
4      private String disciplina;
5
6      public Professor(String nome, int idade, String disciplina) {
7          this.nome = nome;
8          this.idade = idade;
9          this.disciplina = disciplina;
10     }
11
12     @Override
13     public void exibirInformacoes() {
14         System.out.println("Nome: " + nome);
15         System.out.println("Idade: " + idade);
16         System.out.println("Disciplina: " + disciplina);
17     }
18 }
19
```

Princípios da POO - Abstração

- A **abstração** em POO consiste em ocultar detalhes internos da implementação e expor apenas os comportamentos essenciais.
  - Utilizamos **classes abstratas** para criar um modelo base que define métodos genéricos, sem implementação completa.
  - Subclasses concretas devem implementar esses métodos.

```
 2
 3 public abstract class Pessoa {
 4     protected String nome;
 5     protected int idade;
 6
 7     public Pessoa(String nome, int idade) {
 8         this.nome = nome;
 9         this.idade = idade;
10     }
11
12     public abstract void exibirInformacoes();
13
14 }
15
```

O que é Sobrecarga?

- A sobrecarga de métodos é um conceito do polimorfismo que consiste basicamente em criar variações de um mesmo método, ou seja, a criação de dois ou mais métodos com nomes totalmente iguais em uma classe.
- Permite que utilizemos o mesmo nome em mais de um método contanto que suas listas de argumentos sejam diferentes para que seja feita a separação dos mesmos.
- Melhora a legibilidade e a organização do código, permitindo que uma mesma operação seja realizada de formas diferentes, dependendo dos argumentos fornecidos.

Benefícios da Sobrecarga

- ✓**Reutilização de Código:** Permite evitar a duplicação de métodos semelhantes, tornando o código mais limpo e organizado.
- ✓**Facilidade de Uso:** Oferece múltiplas maneiras de chamar um método, melhorando a experiência do desenvolvedor.
- ✓**Melhor Legibilidade:** Mantém nomes coerentes para funcionalidades semelhantes, tornando o código mais intuitivo.
- ✓**Flexibilidade:** Permite tratar diferentes cenários sem a necessidade de criar múltiplos métodos com nomes diferentes.

Regras da Sobrecarga de Métodos

- Os métodos devem ter o mesmo nome.
- Devem possuir assinaturas diferentes (quantidade ou tipo de parâmetros).
- Não podem diferir apenas pelo tipo de retorno.

```
2
3   public class Calculadora {
4       public int soma(int a, int b, int c) {
5           return a + b + c;
6       }
7       public int soma(int a, int b) {
8           return a + b;
9       }
10      public double soma(int a, int b) {
11          return a + b;
12      }
13  }
```

✖ Errado

```
2
3   public class Calculadora {
4       public int soma(int a, int b) {
5           return a + b;
6       }
7       public int soma(int a, int b, int c) {
8           return a + b + c;
9       }
10      public double soma(double a, double b) {
11          return a + b;
12      }
13  }
```

✓ Correto

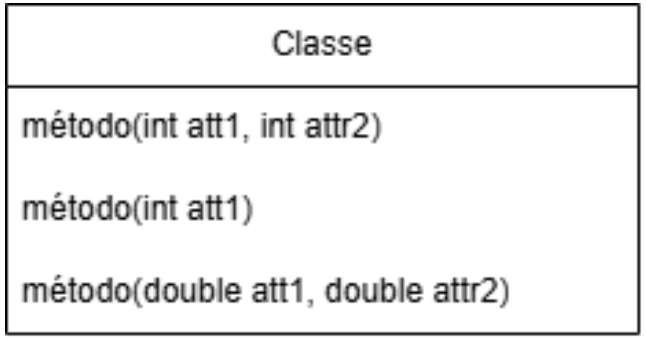
Sobrecarga *vs* Sobreposição

- A **sobrecarga** está ligada a variância de estados de um método, podemos entende-la como um conjunto de opções que o programa principal tem para escolher quando recebe os parâmetros passados pelo usuário.

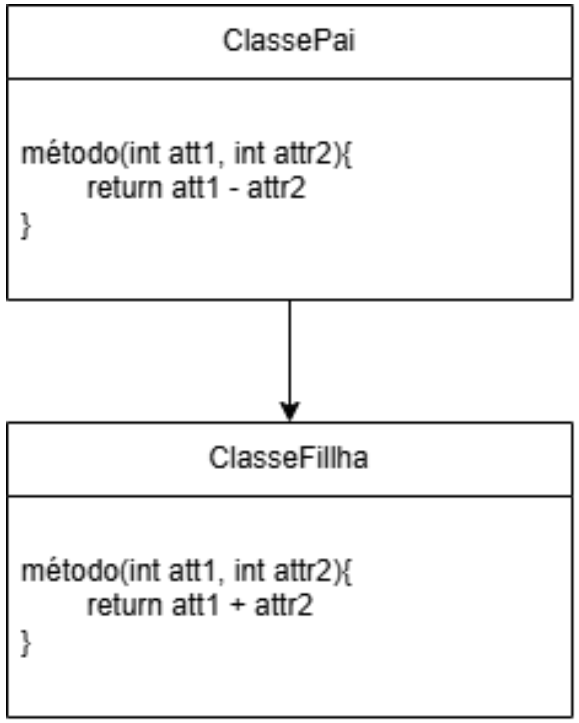
- A **sobreposição** funciona por meio do sistema de herança, e para a mesma funcionar o nome e lista de argumentos dos métodos devem ser totalmente iguais aos da classe herdada.

Sobrecarga *vs* Sobreposição

Sobrecarga



Sobreposição



Sobrecarga *vs* Sobreposição

Sobrecarga

```java

public class Calculadora {
    public int soma(int a, int b) {
        return a + b;
    }
    public int soma(int a, int b, int c) {
        return a + b + c;
    }
    public double soma(double a, double b) {
        return a + b;
    }
}

```

Sobreposição

```java

public class Pessoa {
    String nome;
    int idade;

    public void exibirInformacoes() {
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
    }
}
```

```java

public class Aluno extends Pessoa {
    String disciplina;

    @Override
    public void exibirInformacoes() {
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
        System.out.println("Disciplina: " + disciplina);
    }
}

```

Métodos Construtores

- Os métodos construtores são métodos especiais em Java utilizados para inicializar objetos de uma classe. Eles têm o mesmo nome da classe e são chamados automaticamente quando um objeto é instanciado usando new.

Objeto objeto1 = new Objeto(attr1, attr2)

Métodos Construtores

- Principais Características dos Construtores:
  - **Mesmo nome da classe** – O nome do construtor deve ser idêntico ao da classe.
  - **Não possuem um tipo de retorno** – Nem mesmo void deve ser declarado.
  - **Executados automaticamente** – Quando um objeto é criado, o construtor é chamado sem necessidade de invocação explícita.
  - **Podem ser sobrecarregados** – Permite múltiplos construtores na mesma classe com assinaturas diferentes.

Métodos Construtores

```java

public class Pessoa {
    String nome;
    int idade;

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


    public void exibirInformacoes() {
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
    }
}

```

```java

public class App {
    Run | Debug | Run main | Debug main
    public static void main(String[] args) throws Exception {
        Pessoa pessoa = new Pessoa(nome:"João", idade:20);

        pessoa.exibirInformacoes();
    }
}


```

Tipos de Construtores em Java

1. Construtor Padrão (Sem Parâmetros)
2. Construtor Parametrizado
3. Sobrecarga de Construtores
4. Construtor Copiador

1. Construtor Padrão (Sem Parâmetros)

- Caso nenhum construtor seja definido, Java cria um construtor padrão automaticamente.

```
1
2   public class Pessoa {
3       String nome;
4       int idade;
5
6       public Pessoa() {
7           this.nome = "Sem Nome";
8       }
9
10      public void exibirInformacoes() {
11          System.out.println("Nome: " + nome);
12          System.out.println("Idade: " + idade);
13      }
14  }
15
```

2. Construtor Parametrizado

- Construtores podem aceitar parâmetros para inicializar atributos.

```java

public class Pessoa {
    String nome;
    int idade;

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


    public void exibirInformacoes() {
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
    }
}

```

3. Sobrecarga de Construtores

- Podemos criar vários construtores com diferentes parâmetros.

```java
public class Pessoa {
    String nome;
    int idade;

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

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

    public Pessoa() {
        this.nome = "Sem Nome";
    }

    public void exibirInformacoes() {
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
    }
}
```

4. Construtor Copiador

- Permite criar um novo objeto a partir de outro já existente.

```java

public class Pessoa {
    String nome;
    int idade;

    public Pessoa(Pessoa outraPessoa) {
        this.nome = outraPessoa.nome;
        this.idade = outraPessoa.idade;
    }

    public void exibirInformacoes() {
        System.out.println("Nome: " + nome);
        System.out.println("Idade: " + idade);
    }
}

```

Vantagens do Uso de Construtores

- ✓Facilitam a inicialização dos objetos
- ✓Garantem que um objeto tenha valores válidos
- ✓Melhoram a legibilidade e manutenção do código
- ✓Permitem flexibilidade na criação de instâncias (com sobrecarga)

Trabalhos da disciplina

- Serão vários trabalhos durante o bimestre.
- Alguns trabalhos serão interligados.
- Entregues todos juntos e um único projeto.
- Entrega no final do Bimestre.
- Grupos de até 4 pessoas.