Conceito Geral de Métodos

M**étodos em Java**, abrangendo:

- Conceito geral de métodos;
- Tipos de métodos (com exemplos em Java);
- Sobrecarga de métodos;
- Sobrescrita de métodos;
- Métodos construtores e destrutores;
- Métodos getters e setters;
- Métodos especiais.

Um **método** é um bloco de instruções que executa uma ação específica dentro de uma classe. Métodos definem comportamentos que objetos podem realizar.

Tipos de Métodos

1. Métodos Simples (sem retorno e sem parâmetros):

```
public class BoasVindas {
    public void exibirSaudacao() {
        System.out.println("Seja bem-vindo à aula de Java!");
    }

    public static void main(String[] args) {
        BoasVindas mensagem = new BoasVindas();
        mensagem.exibirSaudacao();
    }
}
```

Saída:

```
Seja bem-vindo à aula de Java!
```

2. Métodos com Retorno (sem parâmetros):

```
public class DadosAluno {
    public String nomeAluno() {
        return "Maria Silva";
    }

    public static void main(String[] args) {
        DadosAluno aluno = new DadosAluno();
        System.out.println("Nome do aluno: " + aluno.nomeAluno());
    }
}
```

Saída:

```
Nome do aluno: Maria Silva
```

3. Métodos com Parâmetros (sem retorno):

```
public class Impressora {
    public void imprimirNome(String nome) {
        System.out.println("Nome recebido: " + nome);
    }

    public static void main(String[] args) {
        Impressora imp = new Impressora();
        imp.imprimirNome("João");
    }
}
```

Saída:

```
Nome recebido: João
```

4. Métodos com Retorno e Parâmetros:

```
public class Calculadora {
    public int multiplicar(int x, int y) {
        return x * y;
    }

    public static void main(String[] args) {
        Calculadora calc = new Calculadora();
        int resultado = calc.multiplicar(4, 5);
        System.out.println("Resultado: " + resultado);
    }
}
```

Saída:

```
Resultado: 20
```

Sobrecarga de Métodos (Overloading)

Sobrecarga ocorre quando vários métodos têm o mesmo nome, mas parâmetros diferentes (em quantidade ou tipo).

```
public class SobrecargaExemplo {
    public void exibir(int x) {
        System.out.println("Valor inteiro: " + x);
    }

    public void exibir(String texto) {
        System.out.println("Texto: " + texto);
    }

    public void exibir(int x, String texto) {
        System.out.println(x + " - " + texto);
    }

    public static void main(String[] args) {
        SobrecargaExemplo sobrecarga = new SobrecargaExemplo();
```

```
        sobrecarga.exibir(10);
        sobrecarga.exibir("Exemplo");
        sobrecarga.exibir(7, "Java");
    }
}
```

Saída:

```
Valor inteiro: 10
Texto: Exemplo
7 - Java
```

Sobrescrita de Métodos (Overriding)

Sobrescrita acontece quando uma subclasse redefine um método da classe mãe.

```
class Animal {
    public void emitirSom() {
        System.out.println("Animal faz som");
    }
}

class Cachorro extends Animal {
    @Override
    public void emitirSom() {
        System.out.println("Au Au!");
    }
}

public class Sobrescrita {
    public static void main(String[] args) {
        Animal animal = new Animal();
        Animal cachorro = new Cachorro();

        animal.emitirSom();
        cachorro.emitirSom();
    }
}
```

Saída:

```
Animal faz som
Au Au!
```

Métodos Construtores e Destrutores

Construtores

Construtor é um método especial usado para inicializar objetos.

```
public class Aluno {
    String nome;
```

```
    // Método construtor
    public Aluno(String nome) {
        this.nome = nome;
    }

    public void exibirAluno() {
        System.out.println("Aluno: " + nome);
    }

    public static void main(String[] args) {
        Aluno aluno = new Aluno("Pedro");
        aluno.exibirAluno();
    }
}
```

Saída:

```
Aluno: Pedro
```

Destrutores

Java **não possui destrutores explícitos**, mas existe o método especial `finalize()` (raramente utilizado).

```
public class ExemploDestrutor {
    protected void finalize() {
        System.out.println("Objeto está sendo removido pelo Garbage
Collector");
    }

    public static void main(String[] args) {
        ExemploDestrutor exemplo = new ExemploDestrutor();
        exemplo = null;  // torna objeto disponível para remoção
        System.gc();     // sugere execução do Garbage Collector
    }
}
```

Nota: O uso de `finalize()` é desencorajado e raro em Java moderno.

Métodos Getters e Setters

São métodos especiais que acessam e modificam atributos privados das classes.

```
public class Pessoa {
    private String nome;

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

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

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

```
        Pessoa pessoa = new Pessoa();
        pessoa.setNome("Ana");
        System.out.println("Nome: " + pessoa.getNome());
    }
}
```

Saída:

```
Nome: Ana
```

Métodos Especiais (Static)

Métodos `static` pertencem diretamente à classe e não exigem instância para serem chamados.

```
public class MetodoEspecial {
    public static void mensagemEstatica() {
        System.out.println("Método estático chamado!");
    }

    public static void main(String[] args) {
        MetodoEspecial.mensagemEstatica();
    }
}
```

Saída:

```
Método estático chamado!
```

Resumo em Tabela

Método	Descrição
Simples	Sem retorno e parâmetros
Com retorno	Retorna valor após execução
Com parâmetros	Recebe valores para execução
Sobrecarga	Mesmo nome, parâmetros diferentes
Sobrescrita	Redefinição em subclasses
Construtores	Inicializam objetos
Destrutores	`finalize()` (raro)
Getters e Setters	Acessam e modificam atributos privados
Métodos especiais	`static`: Chamados diretamente pela classe