✅ Resumo: Classes e Objetos em Java

🔹 Classe

É um **molde** ou **modelo** para criação de objetos. Define os **atributos (campos)** e **comportamentos (métodos)** que os objetos terão.

```
public class Cliente {
    String nome;
    String cidade;
    String estado;

    String listarDados() {
        return "Nome: " + nome + "\nCidade: " + cidade + "\nEstado:
" + estado;
    }
}
```

🔹 Objeto

É uma **instância** de uma classe. Ou seja, quando você usa o `new`, está criando um objeto real baseado na classe.

```
public class Principal {
    public static void main(String[] args) {
        Cliente cliente1 = new Cliente(); // objeto instanciado da
classe Cliente
        cliente1.nome = "Ana";
        cliente1.cidade = "Rio de Janeiro";
        cliente1.estado = "RJ";

        System.out.println(cliente1.listarDados());
    }
}
```

Saída esperada:

```
Nome: Ana
Cidade: Rio de Janeiro
Estado: RJ
```

🔹 Campos (Atributos ou Variáveis de Instância)

São as **características** do objeto, definidos dentro da classe, fora dos métodos.

```
String nome;     // campo
String cidade;   // campo
```

🔹 Métodos

São as **ações** que os objetos podem realizar. Equivalem a funções, e podem ou não retornar valores.

```
String listarDados() {
    return "Nome: " + nome;
}
```

🔹 Instância

É o processo de **criar um objeto a partir de uma classe**, usando o operador `new`.

```
Cliente cliente1 = new Cliente();
```

✅ Comparando os termos:

Termo	O que é?	Onde é usado?
Classe	Molde de um objeto	Definida uma vez, em arquivo `.java`
Objeto	Instância real de uma classe	Criado com `new`
Campo	Variável da classe (atributo)	Dentro da classe, fora dos métodos
Método	Ação que o objeto pode executar	Dentro da classe
Variável local	Só existe dentro de métodos	Declarada dentro de um método

---

💬 Dica final:

Você pode usar o `System.out.println()` para testar os valores dos objetos e verificar o que foi armazenado ou retornado por métodos, como no exemplo com `listarDados()`.

✅ Métodos Construtores em Java

🔹 O que é um Método Construtor?

Um **método construtor** é um método especial de uma classe que é chamado automaticamente quando um **objeto é criado**. Ele tem a responsabilidade de inicializar os dados do objeto.

🔹 Sintaxe do Construtor

1. **Construtor Vazio (sem parâmetros)**:
    - Caso você não defina um construtor, o compilador cria um **construtor padrão vazio**.
    - O **construtor vazio** não realiza nenhuma inicialização, deixando os atributos do objeto em seus valores padrão (ex: `null` para objetos, `0` para inteiros).

Exemplo de construtor padrão vazio:

```
public Cliente() {
}
```

2.

Construtor com Inicialização de Campos: Você pode definir um construtor para **inicializar** os valores padrão dos campos do objeto quando ele for criado.

Exemplo de construtor com valores default:

```
public Cliente() {
    nome = "INDEFINIDO";
    cidade = "São Paulo";
    estado = "SP";
}
```

3.

🔹 Sobrecarga de Construtores

Você pode definir **vários construtores** na mesma classe, desde que eles tenham **parâmetros diferentes**. Isso é chamado de **sobrecarga de construtores**.

- **Construtor padrão**: Sem parâmetros.
- **Construtor com parâmetros**: Aceita parâmetros para inicializar o objeto com valores específicos.

Exemplo:

```
public class Cliente {
    String nome;
    String cidade;
    String estado;

    // Construtor padrão
    public Cliente() {
        nome = "INDEFINIDO";
        cidade = "São Paulo";
        estado = "SP";
    }

    // Construtor com parâmetros
    public Cliente(String nome, String cidade, String estado) {
        this.nome = nome;
        this.cidade = cidade;
        this.estado = estado;
    }
}
```

Agora, você pode criar um `Cliente` de duas formas:

Usando o **construtor padrão** (sem parâmetros):

```
Cliente cliente1 = new Cliente();  // nome = "INDEFINIDO", cidade =
"São Paulo", estado = "SP"
```

-

Usando o **construtor com parâmetros**:

```
Cliente cliente2 = new Cliente("João", "Rio de Janeiro", "RJ");  //
nome = "João", cidade = "Rio de Janeiro", estado = "RJ"
```

-

🔹 O que é o `this` no Construtor?

Dentro de um construtor, quando os **nomes dos parâmetros** são iguais aos **nomes dos campos da classe**, o `this` é necessário para **diferenciar** os campos dos parâmetros.

- O `this` se refere ao **objeto atual** da classe e permite **acessar** os atributos (campos) da classe. Sem o `this`, o compilador ficaria confuso e não saberia se você está se referindo a um **parâmetro** ou a um **campo da classe**.

Exemplo com `this`:r

```
public Cliente(String nome, String cidade, String estado) {
    this.nome = nome;     // 'this.nome' é o campo da classe, 'nome'
é o parâmetro
    this.cidade = cidade; // 'this.cidade' é o campo da classe,
'cidade' é o parâmetro
    this.estado = estado; // 'this.estado' é o campo da classe,
'estado' é o parâmetro
}
```

🔹 Garbage Collector

O **Garbage Collector (GC)** é um processo automático da JVM que **limpa objetos não utilizados** na memória. Quando um objeto não tem mais referências, o GC o remove da memória para **evitar vazamentos** e **melhorar o desempenho**.

- **Importante**: Embora o GC automatize a limpeza de objetos, é sempre bom ter em mente o desempenho do seu sistema e evitar criar objetos desnecessários.

✅ Resumo e Exemplos

Construtor Padrão:

```
public Cliente() {
    nome = "INDEFINIDO";
    cidade = "São Paulo";
```

```
    estado = "SP";
}
```

1.

Construtor com Parâmetros:

```
public Cliente(String nome, String cidade, String estado) {
    this.nome = nome;
    this.cidade = cidade;
    this.estado = estado;
}
```

2.

Instanciando um Objeto:

```
Cliente cliente1 = new Cliente();  // Usando o construtor padrão
Cliente cliente2 = new Cliente("Ana", "Marília", "SP");  // Usando o construtor com parâmetros
```

Boa, Patrick! Esse trecho aí tá explicando um dos conceitos mais importantes e que gera bastante dúvida no início: o uso do `static` em Java. Bora destrinchar com mais clareza e alguns exemplos práticos pra ficar cravado na mente 🔍

📌 O que é um método estático?

Um **método estático** é aquele que **pertence à classe, e não a uma instância (objeto)** da classe. Ou seja, você pode chamá-lo **sem precisar criar um objeto**.

Exemplo:

```
public class Util {
   public static int somar(int a, int b) {
      return a + b;
   }
}
```

Chamando o método:

```
int resultado = Util.somar(10, 20); // Não precisa de "new Util()"
```

🧠 Situações típicas de uso de métodos estáticos:

- Métodos **utilitários**, como conversões (`Integer.parseInt`, `Double.parseDouble`);
- Métodos **fábrica** (`static factory methods`);
- Métodos de **validação**, **formatação**, **cálculo** etc.

⚠️ Regras importantes:

Recurso	Pode usar em métodos estáticos?
Variáveis **não estáticas**	❌ NÃO pode
Variáveis **estáticas**	✅ SIM
Outros **métodos estáticos**	✅ SIM

Por quê? Porque métodos estáticos são carregados **antes** da criação de qualquer objeto. Então eles **não sabem o que é `this`**, nem têm acesso ao "mundo de instância" da classe.

🧪 Exemplo completo:

```
public class Cliente {
    private String nome;
    private static int totalClientes = 0;

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

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

    public static int getTotalClientes() {
        return totalClientes; // Pode acessar porque é static também
    }
}
```

Uso:

```
Cliente c1 = new Cliente("Ana");
Cliente c2 = new Cliente("Carlos");

System.out.println(Cliente.getTotalClientes()); // Saída: 2
```

🎯 Dica:

Use `static` quando:

- O método **não depende de informações específicas de um objeto**;
- Você quer ter **um único valor compartilhado** por todos os objetos da classe (como um contador, por exemplo).
