**⚙️ Diferença principal:**

* **Herança (extends)**:  
  Uma classe herda **implementações e atributos** de outra classe.
* **Interface (implements)**:  
  Uma classe **se compromete a implementar comportamentos (métodos)**, mas **não herda implementação concreta** (a menos que a interface tenha métodos default ou static).

**🟦 Herança (com extends)**

**✅ Exemplo:**

class Animal {

void comer() {

System.out.println("Animal comendo");

}

}

class Cachorro extends Animal {

void latir() {

System.out.println("Au au!");

}

}

**🟢 Características:**

* Uma **classe pode estender apenas uma outra classe** (herança simples).
* Herda **métodos concretos e atributos**.
* Usa a palavra-chave extends.

**🟨 Interface (com implements)**

**✅ Exemplo:**

interface Animal {

void comer(); // só a assinatura

}

class Gato implements Animal {

public void comer() {

System.out.println("Gato comendo");

}

}

**🟢 Características:**

* Uma **classe pode implementar várias interfaces** (herança múltipla de tipo).
* A interface define apenas **a assinatura** dos métodos (ou métodos default, desde Java 8).
* Usa a palavra-chave implements.
* Interface não tem **estado (atributos de instância)**.

**🧠 Resumo prático:**

| **Característica** | **Interface** | **Herança (classe)** |
| --- | --- | --- |
| Palavra-chave | implements | extends |
| Quantidade permitida | Múltiplas interfaces | Apenas uma classe |
| Contém implementação? | Não (a menos que seja default) | Sim |
| Pode ter atributos? | Só constantes (public static final) | Sim, com estado |
| Uso principal | **Contrato de comportamento** | **Reuso de código** |

**🧠 Por que Java tem tanto herança quanto interface?**

**🟩 1. Limitação da herança múltipla em classes**

* Em Java, **uma classe só pode estender uma única superclasse** (herança simples).
* Isso evita ambiguidade (como o famoso “**problema do diamante**”).

class A { void metodo() {} }

class B extends A {}

class C extends B {} // ok

// class D extends A, B {} // ❌ Não pode!

🟡 **Interfaces resolvem isso!**

* Uma classe pode implementar **várias interfaces ao mesmo tempo**:

class MinhaClasse implements Interface1, Interface2, Interface3 { ... }

**🟨 2. Herança é sobre o que uma classe *é***

* Ex: Cachorro extends Animal → um cachorro **é** um animal.

**🟦 Interface é sobre o que uma classe *pode fazer***

* Ex: Cachorro implements Corrivel → um cachorro **pode correr**.
* Ex: Drone implements Voador → um drone **pode voar** (mas não é um animal).

📌 Isso permite que **coisas diferentes compartilhem comportamento**, sem estar na mesma hierarquia.

**🟧 3. Separação de papéis**

* **Herança**: para **reuso de código e estrutura**.
* **Interface**: para **contrato de comportamento**, sem impor estrutura ou implementação.

**🟦 4. Design mais flexível e testável**

Interfaces ajudam a:

* Definir **contratos desacoplados**.
* Substituir implementações facilmente (ideal para testes ou injeção de dependência).
* Criar código mais modular e reutilizável.

**🧪 Exemplo prático:**

interface Nadador {

void nadar();

}

class Animal {

void respirar() {

System.out.println("Respirando");

}

}

class Peixe extends Animal implements Nadador {

public void nadar() {

System.out.println("Peixe nadando");

}

}

Aqui:

* Peixe **herda** de Animal porque é um animal.
* Peixe **implementa** Nadador porque sabe nadar.
* Se amanhã Submarino também precisar nadar, ele pode **implementar Nadador** sem ter nada a ver com animais.

**✅ Resumo:**

| **Conceito** | **Herança (extends)** | **Interface (implements)** |
| --- | --- | --- |
| Reuso de código? | ✅ Sim | 🔸 Só via default (desde Java 8) |
| Várias por classe? | ❌ Só uma classe base | ✅ Várias interfaces |
| Representa | Uma relação **"é um(a)"** | Uma relação **"sabe fazer"** |
| Estrutura + método | ✅ Herda atributos e métodos | ❌ Só contratos (e default se usar) |
| Exemplo | class Gato extends Animal | class Drone implements Voador |

Java mantém os dois justamente porque **interfaces e herança resolvem problemas diferentes** — e, quando usados juntos, ajudam a construir sistemas mais limpos e flexíveis.

**✅ Exemplo permitido:**

class Animal {

void respirar() {

System.out.println("Respirando...");

}

}

class Gato extends Animal {

void miar() {

System.out.println("Miau");

}

}

class Cachorro extends Animal {

void latir() {

System.out.println("Au au");

}

}

Neste caso:

* Gato e Cachorro **ambos estendem Animal** → isso é totalmente válido.

**❌ Exemplo proibido:**

class Mamifero { }

class Ave { }

// Isso não compila:

class Morcego extends Mamifero, Ave { } // ❌ Erro: herança múltipla de classes

**🧠 Regras de herança em Java:**

| **Situação** | **Pode?** |
| --- | --- |
| Uma classe estender **outra única classe** | ✅ Sim |
| Várias classes estenderem **a mesma superclasse** | ✅ Sim |
| Uma classe estender **mais de uma classe** | ❌ Não |

**Como resolver a limitação?**

Use **interfaces**! Java permite implementar várias interfaces:

interface Voador {

void voar();

}

interface Mamifero {

void amamentar();

}

class Morcego implements Voador, Mamifero {

public void voar() {

System.out.println("Morcego voando");

}

public void amamentar() {

System.out.println("Morcego amamentando");

}

}