

Programação III

Aula 4 – Herança e Polimorfismo



Definição de Herança em Java

```
class A {
    int i, j;
    void showij() {
        System.out.println("i and j: "
            + i + " " + j);
    }
}

class B extends A {
    int k;
    void showk() {
        System.out.println("k: " + k);
    }
    void sum() {
        System.out.println("i+j+k: " +
            (i + j + k));
    }
}
```

```
class SimpleInheritance {
    public static void main(String[] args) {
        A superOb = new A(); // Objeto da superclasse
        B subOb = new B(); // Objeto da subclasse

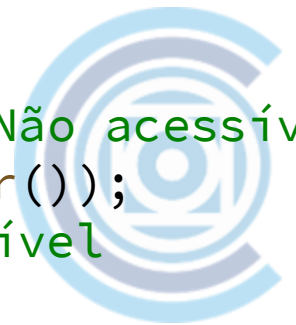
        superOb.i = 10;
        superOb.j = 20;
        superOb.showij(); // Acessa métodos da classe A

        subOb.i = 7;
        subOb.j = 8;
        subOb.k = 9;
        subOb.showij(); // Herdado de A
        subOb.showk(); // Método exclusivo de B
        subOb.sum(); // Combina atributos de A e B
    }
}
```



Modificadores de Acesso

```
class SuperClass {  
    private int privateVar = 10; // Apenas acessível na própria classe  
    protected int protectedVar = 20; // Acessível na classe e nas subclasses  
    public int publicVar = 30; // Acessível em qualquer lugar  
  
    public int getPrivateVar() { // Método público para acessar privateVar  
        return privateVar;  
    }  
}  
  
class SubClass extends SuperClass {  
    void showValues() {  
        // System.out.println("privateVar: " + privateVar); // ERRO: Não acessível  
        System.out.println("privateVar (via getter): " + getPrivateVar());  
        System.out.println("protectedVar: " + protectedVar); // Acessível  
        System.out.println("publicVar: " + publicVar); // Acessível  
    }  
}
```




Modificadores de Acesso

```
public static void main(String[] args) {  
    SuperClass superClass = new SuperClass();  
    SubClass subClass = new SubClass();  
  
    // Acessando membros da superclasse diretamente  
    // System.out.println("privateVar: " + superClass.privateVar); // ERRO  
    System.out.println("privateVar (via getter): " + superClass.getPrivateVar());  
    // System.out.println("protectedVar: " + superClass.protectedVar); // ERRO  
    System.out.println("publicVar: " + superClass.publicVar); // Acessível  
  
    System.out.println("\nValores na subclasse:");  
    subClass.showValues(); // Mostra os valores acessíveis  
}
```



Uso da palavra-chave **super**

- A palavra-chave **super** é usada em uma subclasse para se referir diretamente aos membros (atributos ou métodos) da sua superclasse.
- Pode ser usada para:
 - **Acessar membros da superclasse** que foram ocultados pela subclasse.
 - Chamar o construtor da superclasse



```
class SuperClass {  
    int num = 100;  
    void display() { System.out.println("SuperClass"); }  
}  
class SubClass extends SuperClass {  
    int num = 200;  
    void show() {  
        System.out.println(super.num); // 100  
        super.display(); // Chama o método da superclasse  
    }  
}
```



Usando `super()` no construtor

- Necessidade:
 - Para inicializar membros herdados ou configurar a superclasse.
 - Deve ser a primeira instrução no construtor da subclasse.

```
class SuperClass {  
    SuperClass(String msg) { System.out.println(msg); }  
}
```

```
class SubClass extends SuperClass {  
    SubClass(String msg) { super(msg); } // Chama o construtor da SuperClass  
}
```



Sobrecarga de métodos

- Definição
 - Subclasse redefine um método da superclasse com mesma assinatura (nome e parâmetros).
 - O método da subclasse substitui o da superclasse para objetos da subclasse.

```
class A {  
    void show() { System.out.println("Superclasse"); }  
}  
class B extends A {  
    @Override  
    void show() { System.out.println("Subclasse"); }  
}  
public class Test {  
    public static void main(String[] args) {  
        B obj = new B();  
        obj.show(); // Subclasse  
    }  
}
```



Sobrecarga de métodos – uso do super

```
class A {  
    void show() { System.out.println("Superclasse"); }  
}  
class B extends A {  
    @Override  
    void show() {  
        super.show(); // Chama o método da superclasse  
        System.out.println("Subclasse");  
    }  
}  
public class Test {  
    public static void main(String[] args) {  
        B obj = new B();  
        obj.show();  
    }  
}
```



Polimorfismo em Java

```
class Animal {
    void sound() {
        System.out.println("Animal faz som");
    }
}
class Dog extends Animal {
    @Override
    void sound() {System.out.println("Cachorro late");}
}
class Cat extends Animal {
    @Override
    void sound() {System.out.println("Gato mia");}
}

public class Teste {
    public static void main(String[] args) {
        Animal a; // Referência genérica
        a = new Dog();
        a.sound(); // Cachorro late
        a = new Cat();
        a.sound(); // Gato mia
    }
}

//Exemplo com Array
Animal[] animals = {new Dog(), new Cat()};
for (Animal a : animals) {
    a.sound();
}
```



Classes abstratas

- Uma classe abstrata é uma classe que **não pode ser instanciada**.
- Serve como base para outras classes (superclasse).
- Pode conter **métodos abstratos** (sem implementação) e métodos concretos (com implementação).
 - Métodos abstratos **devem** ser implementados nas subclasses

```
abstract class NomeDaClasse {  
    abstract void metodoAbstrato(); // Sem corpo  
    void metodoConcreto() {  
        System.out.println("Método concreto");  
    }  
}
```



Classes abstratas - Exemplo

```
abstract class Animal {
    abstract void sound(); // Método abstrato
    void eat() { // Método concreto
        System.out.println("Animal está comendo");
    }
}

class Dog extends Animal {
    @Override
    void sound() {
        System.out.println("Cachorro late");
    }
}

class Cat extends Animal {
    @Override
    void sound() {
        System.out.println("Gato mia");
    }
}

public class Main {
    public static void main(String[] args)
    {
        Animal a;
        a = new Dog();
        a.sound(); // Cachorro late
        a.eat(); // Animal está comendo
    }
}
```



Classe **Object** em Java

- Definição:
 - A classe **Object** é a superclasse de todas as classes em Java.
 - Todos os objetos em Java herdam direta ou indiretamente dela.
- Principais Características:
 - Proporciona um comportamento comum a todas as classes.
 - Métodos herdados de **Object** podem ser sobrescritos para personalização.



Principais métodos da classe `Object`

Método	Descrição
<code>equals(Object o)</code>	Compara dois objetos para determinar se são iguais.
<code>hashCode()</code>	Retorna um valor numérico usado para identificar o objeto em coleções baseadas em hash.
<code>toString()</code>	Retorna uma representação em string do objeto.
<code>getClass()</code>	Retorna o tipo da classe em tempo de execução.
<code>clone()</code>	Cria uma cópia do objeto (necessita de <code>Cloneable</code>).
<code>finalize()</code>	Método chamado pelo garbage collector antes de destruir o objeto (obsoleto).

Classe Object - Exemplo

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

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

    @Override
    public String toString() {
        return "Pessoa [nome=" + nome + ", idade=" + idade + "]";
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Pessoa pessoa = (Pessoa) o;
        return idade == pessoa.idade && nome.equals(pessoa.nome);
    }
}

public class Main {
    public static void main(String[] args) {
        Pessoa p1 = new Pessoa("Ana", 30);
        Pessoa p2 = new Pessoa("Ana", 30);
        System.out.println(p1); // Pessoa [nome=Ana, idade=30]
        System.out.println(p1.equals(p2)); // true
    }
}
```

