02

Implementando Herança

Em Java, podemos usar herança simples entre classes com o `extends`. A nomenclatura usada é de **classe mãe** (*parent class*) e **classe filha** (*child class*), ou **superclasse** e **subclasse**.

Herança entre classes permite que um código seja reaproveitado, de maneira que a classe filha reutilize o código da parte mãe, preocupando-se principalmente em sua especialização. A filha especializa a classe mais genérica. Herança em Java pode ser entre classes, reaproveitando membros, ou herança de uma interface, com a qual reaproveitamos interfaces de métodos.

```
class Mae {
}
class Filha extends Mae {
}
class Neta extends Filha {
}
```

COPIAR CÓDIGO

Vale lembrar que toda classe que não define de quem está herdando herda de `Object`:

```
class Explicito extends Object {
}
class Implicito {
    // extends Object por padrão
```

```
}
class FilhoDeImplicito extends Implicito{
    // também herda de Object, indiretamente
}
```

COPIAR CÓDIGO

Mas não podemos herdar de duas classes:

```
class Simples1 {}
class Simples2 {}
class Complexa extends Simples1, Simples2 {
    // não compila
}
```

COPIAR CÓDIGO

Para podermos herdar de uma classe, a classe mãe precisa ser visível pela classe filha e pelo menos um de seus construtores também:

```
class Pai {
    Pai(int x) {
    }
}
class Filho1 extends Pai{
    // não compila pois o construtor padrão chama super()
    // e o Pai não tem construtor vazio
}
class Filho2 extends Pai{
    Filho2() {
        super(15); //compila
    }
}
```

COPIAR CÓDIGO

Além disso, a classe mãe não pode ser `final` :

```
final class Pai {
}
class Filho extends Pai {
    // não compila, Pai é final
}
class Mae {
}
final class Filha extends Mae {
    // uma classe final pode estender de alguém, compila
}
```

COPIAR CÓDIGO

Herança de métodos e atributos

Todos os métodos e atributos de uma classe mãe são herdados (independente das visibilidades).

```
class X {
    int x;
    public void y() {
    }
}
class Y {
    // tenho um x, e o método y
}
```

COPIAR CÓDIGO

Dependendo da visibilidade e das classes envolvidas, a classe filha não consegue enxergar o membro herdado. No exemplo a seguir, herdamos o atributo mas não o enxergamos diretamente.

```
class X {
    private int x;

    public void setX(int x) {
        this.x = x;
    }

    public int getX() {
        return x;
    }
}

class Y extends X {
    public void metodo () {
        this.x = 5; // não compila: "x has private access in
X"

        this.setX(10); // compila e altero o x herdado mas
                       // não visível
    }
}
```

COPIAR CÓDIGO

Métodos estáticos e herança

Não existe herança de métodos estáticos. Mas quando herdamos de uma classe com métodos estáticos, podemos chamar o método da classe mãe usando o nome da filha (embora não seja uma boa prática):

```
class W {
    static void metodo() {
    }
}

class  Z extends W {
}

class Teste {
    public static void main(String[] args) {
        Z.metodo(); // melhor seria escrever W.metodo()
    }
}
```

COPIAR CÓDIGO

```
class W {
    public static void metodo() {
        System.out.println("w");
    }
}
class Z extends W {
    public static void metodo() {

        // não existe super em contexto estático, não compila
        super.metodo();

    }
}
```

COPIAR CÓDIGO

Por não existir herança, o modificador `abstract` não é aceito em métodos estáticos.

Podemos até escrever na classe filha um método estático de mesmo nome, mas isso **não é sobrescrita** (alguns chamam de redefinição):

```
class W {
    public static void metodo() {
        System.out.println("w");
    }
}
class Z extends W {
    public static void metodo() {
        System.out.println("z");
    }
}
public class Teste {
    public static void main(String[] args) {
        System.out.println(W.metodo()); // w
        System.out.println(Z.metodo()); // z
    }
}
```

COPIAR CÓDIGO

Na verdade você até segue as regras de sobrescrita de método (visibilidade e retorno), mas no polimorfismo ele não funciona como métodos normais. Ele simplesmente funciona com o tipo da variável em tempo de compilação e não o tipo do objeto em tempo de execução:

```
public class Teste {
    public static void main(String[] args) {
        W w = new W();
        w.metodo(); // w

        Z z = new Z();
        z.metodo(); // z

        W zPolimorfadoComoW = z;
        zPolimorfadoComoW.metodo();
        // este último imprime w,
        // pois o binding é feito em compilação:
        // zPolimorfadoComoW.metodo é uma referencia
        // em compilação para W
    }
}
```

COPIAR CÓDIGO

Construtores e herança

Não existe herança de construtores. O que existe é a classe filha chamar o construtor da mãe.

Sobrescrita de atributos

Não existe sobrescrita de atributos. Podemos, sim, ter um atributo na classe filha com mesmo nome da mãe, mas não chamamos de sobrescrita. Nesses casos, o objeto vai ter 2 atributos diferentes, um da mãe (acessível com `super`) e um na filha (acessível com `this`).

Object

Em Java, toda classe é obrigada a usar a herança. Quando não escrevemos `extends` explicitamente, estamos herdando de `java.lang.Object` automaticamente.

Isso quer dizer que todo objeto em Java **é um** `Object` e, portanto, herda todos os métodos da classe `Object` (por isso esses são muito importantes).

Há vários métodos em `Object`, que veremos ao longo do curso, mas o mais simples talvez seja o `toString`, que podemos sobrescrever em nossas classes para devolver alguma `String` que represente o objeto:

```
class Carro {
    String cor;

    public String toString() {
        return "Um carro de cor " + this.cor;
    }
}
```

COPIAR CÓDIGO

Temos que lembrar que o `toString` é chamado automaticamente para nós quando usamos o objeto no contexto de `String`:

```
Carro c = new Carro();
c.cor = "Verde";

System.out.println(c); // chama toString
```

```
String s = "Mensagem: " + c; // chama toString
System.out.println(s);
```

COPIAR CÓDIGO