Programación

Herencia

Marcia Fernández Estévez

IES Antón Losada Diéguez 21/22

Constructores y herencia

Cuando se trata de herencia, los constructores se comportan de manera muy diferente al resto de elementos.

Particularidades:

1. Al contrario de lo que ocurre con métodos y campos que son transferidos de una superclase a una subclase, **los métodos constructores no se heredan**.

Por tanto, cada clase tiene sus propios métodos constructores.

2. Aunque no se declare, cada clase tiene de forma implícita, un constructor por defecto (sin parámetros ni código). Pero si se declara cualquier constructor, el constructor por defecto no se creará.

```
class Clase{

    Clase(){} // sobrescribe el constructor por defecto

    Clase(String parametro1){} //constructor con parámetro

}
```

3. Cuando se crea un objeto de una clase, no se invoca únicamente a su constructor sino que se ejecuta el de todas las superclases. Se recorrerán todas las clases de la jerarquía hasta llegar a la clase padre, formando una pila de llamadas y buscando el constructor correspondiente.

a) Si se invoca al constructor con algún argumento, se buscará el método constructor cuyos parámetros “encajen” con dicha llamada. En caso contrario se buscará el método constructor que no tenga parámetros, el cual podrá ser el constructor por defecto (sin parámetros ni código) o bien uno declarado por el programador (también sin parámetros).
b) Si se desea forzar la llamada a un constructor concreto, será necesario invocarlo con la sentencia super como primera instrucción del constructor actual y pasarle los argumentos que espere recibir

```java
class A{
    A(){
        System.out.println("Estoy en constructor de A");
    }
    A(String mensaje){
        System.out.println(mensaje);
    }
    A(int valorA){
         System.out.println(valorA);
    }
}

class B extends A{
    B(){
        System.out.println("Estoy en constructor de B");
    }
    B(String mensaje){
        System.out.println(mensaje);
    }
    B(int valorB){
        super(valorB);
        System.out.println(valorB);
    }
}

class C extends B{
    C(){
        System.out.println("Estoy en constructor de C");
    }
    C(String mensaje){
        System.out.println(mensaje);
    }
    C(int valorC){
        super(valorC);
        System.out.println(valorC);
    }
}

public class Herencia {
    public static void main(String [] args){
        C obj1 = new C();
        System.out.println("----------------------");
        C obj2 = new C("DAM1");
        System.out.println("----------------------");
        C obj3 = new C(1234);


    }

}
```

Acceso a campos de la superclase

En una relación de herencia, cuando un campo tiene el mismo identificador que un campo de la clase base, este último se oculta (incluso cuando los tipos de ambos no son coincidentes).

El campo ocultado podrá ser accedido:

1. Desde instancias de la subclase:
    a. Mediante la palabra reservada super (únicamente se puede subir un nivel de la jerarquía).
    b. Mediante cast explícito a la superclase cuyo campo se ocultó.
2. Desde fuera de la subclase mediante un cast explícito a la superclase cuyo campo se ocultó.

Veamos el ejemplo:

```java
class A {

    String campo1 = "rojo";
    String campo2 = "negro";
}

class B extends A {

    int campo1 = 1;

    //Sugerencia Brais
    void verCamposAB() {
        System.out.println("verCamposAB");
        System.out.println("campo1 de A" + super.campo1);
    }
}
```

```java
class C extends B {

    char campo1 = 'X';

    void verCampos() {
        System.out.println("Acceso desde clase C");
        System.out.println("campo2 de C " + campo2); //negro
        System.out.println("campo1 de C " + campo1); //X
        System.out.println("campo1 de C " + this.campo1); //X
        System.out.println("campo1 de B " + super.campo1); // 1
        System.out.println("campo1 de B " + ((B) this).campo1); //1
        System.out.println("campo1 de A " + ((A) this).campo1); //rojo
    }
}
```

```java
public class HerenciaSobrescritura {

    public static void main(String[] args) {
        C objeto = new C();
        System.out.println("Acceso desde main");
        System.out.println("Campo2 de C = " + objeto.campo2);
        System.out.println("Campo1 de C = " + objeto.campo1);
        System.out.println("Campo1 oculto de B = " + ((B) objeto).campo1);
        System.out.println("Campo1 oculto de A = " + ((A) objeto).campo1);

        objeto.verCampos();
    }
```

Acceso a métodos de la superclase

De la misma forma que se permite el acceso a los constructores y campos de la superclase, también se puede hacer uso de sus métodos desde las subclases, siempre y cuando estos sean visibles.

Para ello es necesario hace uso nuevamente de la sentencia super, pero en este caso no es necesario que sea la primera instrucción del bloque como en los constructores.

Sobrescritura de métodos (@override)

Una subclase heredará de su superclase los métodos que le sena accesibles, a menos que la subclase los sobrescriba, en cuyo caso ya no dispondrá de ellos.

Para sobrescribir un método es necesario declararlo, con las mismas características (identificador, número y tipo de parámetros) que el método de la superclase.

Generalmente, la sobrescritura de métodos sirve para agregar o modificar la funcionalidad del método heredado de la superclase.

Cuando se sobrescribe un método, es recomendable marcarlo con @override

Al sobrescribir el método, este se personaliza para los objetos de la clase, pero el método original seguirá también estando disponible para objetos de la superclase u otras subclases que no lo sobrescriban.

Bibliografía

Dorrego Martín, M. (2019). *Programación.* Madrid: Síntesis.