APRENDERAPROGRAMAR.COM

# EJEMPLO DE HERENCIA EN JAVA. USO DE PALABRAS CLAVE EXTENDS Y SUPER. CONSTRUCTORES CON HERENCIA. (CU00686B)

Sección: Cursos
Categoría: Curso “Aprender programación Java desde cero”

Fecha revisión: 2029

**Resumen:** Entrega nº86 curso Aprender programación Java desde cero.

Autor: Alex Rodríguez

## EJEMPLO DE HERENCIA EN JAVA. EXTENDS Y SUPER.

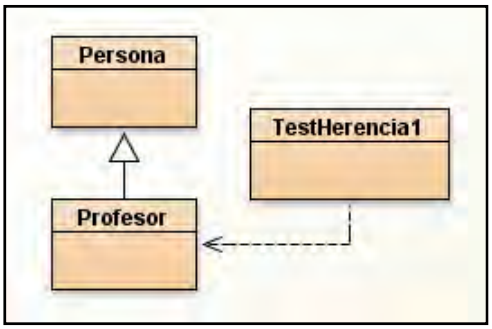
Para declarar la herencia en Java usamos la palabra clave *extends*. Ejemplo: *public class MiClase2 extends Miclase1*. Para familiarizarte con la herencia te proponemos que escribas y estudies un pequeño programa donde se hace uso de ella. Escribe el código de las clases que mostramos a continuación.


```
//Código de la clase Persona ejemplo aprenderaprogramar.com
public class Persona {
   private String nombre;
   private String apellidos;
   private int edad;

   //Constructor
   public Persona (String nombre, String apellidos, int edad) {
      this.nombre = nombre;
      this.apellidos = apellidos;
      this.edad = edad;                }

   //Métodos
   public String getNombre () { return nombre;  }
   public String getApellidos () { return apellidos;  }
   public int getEdad () { return edad;   }
} //Cierre de la clase
```

```
//Código de la clase profesor, subclase de la clase Persona ejemplo aprenderaprogramar.com
public class Profesor extends Persona {
   //Campos específicos de la subclase.
   private String IdProfesor;
   //Constructor de la subclase: incluimos como parámetros al menos los del constructor de la superclase
   public Profesor (String nombre, String apellidos, int edad) {
      super(nombre, apellidos, edad);
      IdProfesor = "Unknown";   } //Cierre del constructor

   //Métodos específicos de la subclase
   public void setIdProfesor (String IdProfesor) { this.IdProfesor = IdProfesor;   }

   public String getIdProfesor () { return IdProfesor;   }

   public void mostrarNombreApellidosYCarnet() {
      // nombre = "Paco"; Si tratáramos de acceder directamente a un campo privado de la superclase, salta un error
      // Sí podemos acceder a variables de instancia a través de los métodos de acceso públicos de la superclase
      System.out.println ("Profesor de nombre: " + getNombre() + " " +  getApellidos() +
       " con Id de profesor: " + getIdProfesor() ); }
} //Cierre de la clase
```

```
//Código de test aprenderaprogramar.com
public class TestHerencia1 {
   public static void main (String [ ] Args) {
      Profesor profesor1 = new Profesor ("Juan", "Hernández García", 33);
      profesor1.setIdProfesor("Prof 22-387-11");
      profesor1.mostrarNombreApellidosYCarnet();}
} //Cierre de la clase
```

El diagrama de clases y el resultado del test son del tipo que mostramos a continuación:




```
Profesor de nombre: Juan Hernández García
con Id de profesor: Prof 22-387-11
```

Los aspectos a destacar del código son:

a) La clase persona es una clase "normal" definida tal y como lo venimos haciendo habitualmente mientras que la clase Profesor es una subclase de Persona con ciertas peculiaridades.

b) Los objetos de la subclase van a tener campos *nombre*, *apellidos* y *edad* (heredados de Persona) y un campo específico *IdProfesor*. El constructor de una subclase ha de llevar obligatoriamente como parámetros al menos los mismos parámetros que el constructor de la superclase.

c) **El constructor de la subclase invoca al constructor de la superclase.** Para ello se incluye, obligatoriamente, la palabra clave super como primera línea del constructor de la subclase. La palabra super irá seguida de paréntesis dentro de los cuales pondremos los parámetros que requiera el constructor de la superclase al que queramos invocar. En este caso solo teníamos un constructor de superclase que requería tres parámetros. Si p.ej. hubiéramos tenido otro constructor que no requiriera ningún parámetro podríamos haber usado uno u otro, es decir, super(nombre, apellidos, edad) ó super(), o bien ambos teniendo dos constructores para la superclase y dos constructores para la subclase. Ejemplo:

```
En la superclase:        public Persona() {
                         nombre = "";
                         apellidos = "";
                         edad = 0; }

                         public Persona (String nombre, String apellidos, int edad) {
                         this.nombre = nombre;
                         this.apellidos = apellidos;
                         this.edad = edad;              }
```

En la subclase:

```
public Profesor () {
super();
IdProfesor = "Unknown";}

public Profesor (String nombre, String apellidos, int edad) {
super(nombre, apellidos, edad);
IdProfesor = "Unknown";   }
```

Modifica el código de las clases Persona y Profesor para que queden con dos constructores tal y como hemos mostrado aquí. Crea objetos de ambos tipos en BlueJ y prueba sus métodos.

**¿Qué ocurre si olvidamos poner super como primera línea de la subclase?** Hay dos posibilidades: si la superclase tiene un constructor sin parámetros, el compilador incluirá en segundo plano super de forma automática y no saltará un error. De cualquier manera se considera contrario al buen estilo de programación, ya que no queda claro si se trata de un olvido. Por ello incluiremos siempre la palabra clave super. La otra posibilidad es que no haya un constructor sin parámetros, en cuyo caso saltará un error.

A modo de resumen: la inicialización de un objeto de una subclase comprende dos pasos. La invocación al constructor de la superclase (primera línea del constructor: super…) y el resto de instrucciones propias del constructor de la subclase.

**EJERCICIO**

Se plantea desarrollar un programa Java que permita la gestión de una empresa agroalimentaria que trabaja con tres tipos de productos: productos frescos, productos refrigerados y productos congelados. Todos los productos llevan esta información común: fecha de caducidad y número de lote. A su vez, cada tipo de producto lleva alguna información específica. Los productos frescos deben llevar la fecha de envasado y el país de origen. Los productos refrigerados deben llevar el código del organismo de supervisión alimentaria. Los productos congelados deben llevar la temperatura de congelación recomendada. Crear el código de las clases Java implementando una relación de herencia desde la superclase Producto hasta las subclases ProductoFresco, ProductoRefrigerado y ProductoCongelado. Cada clase debe disponer de constructor y permitir establecer (set) y recuperar (get) el valor de sus atributos y tener un método que permita mostrar la información del objeto. Crear una clase testHerencia2 con el método main donde se cree un objeto de cada tipo y se muestren los datos de cada uno de los objetos creados.

Puedes comprobar si tu respuesta es correcta consultando en los foros aprenderaprogramar.com.

**Próxima entrega:** CU00687B

**Acceso al curso completo** en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:

http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=68&Itemid=188