Java Constructores this() y super()

En Java creamos objetos todos los días y para ellos usamos constructores. Todos estamos muy habituados a programarlos pero siempre hay algunos bordes que se nos escapan, vamos a revisarlos. Supongamos que tenemos la siguiente clase:

```
1
2   package com.arquitecturajava;
3
4   public class Persona {
5
6       private String nombre;
7
8       public String getNombre() {
9           return nombre;
10      }
11
12      public void setNombre(String nombre) {
13          this.nombre = nombre;
14      }
15  }
```

Java Constructores por defecto

¿Tiene esta clase algún constructor? La respuesta es sí toda clase Java si no se le incluye ningún constructor el compilador añade **un constructor por defecto**. Así pues el código para el compilador sería el siguiente:

```
1
2   package com.arquitecturajava;
3
4   public class Persona {
5
6       private String nombre;
7
8       public Persona() {
9
10      }
11
12      public String getNombre() {
13        return nombre;
14      }
15
16      public void setNombre(String nombre) {
17        this.nombre = nombre;
18      }
19  }
```

Como vemos para el compilador existe un constructor por defecto vacío. ¿Ahora bien y si el código de nuestra clase incluyera un constructor con un parámetro?

```
1   package com.arquitecturajava;
2
3   public class Persona {
```

```
4    private String nombre;
5
6    public Persona(String nombre) {
7       this.nombre = nombre;
8    }
9
10   public String getNombre() {
11      return nombre;
12   }
13
14   public void setNombre(String nombre) {
15      this.nombre = nombre;
16   }
   }
```

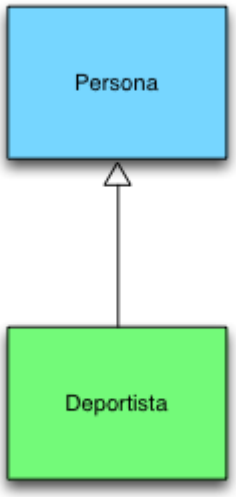
En ese caso Java no añade el constructor por defecto. Así pues en ambos casos **tenemos un único constructor.**

Cuestión de buenas prácticas…

Crear un **constructor parametrizado** que contenga **todos los atributos** de la clase (por razones de claridad, detección y corrección de errores y depuración).

Java constructores y super()

Las dudas con los constructores aparecen ligadas a las jerarquías de clases y a la palabra **super()**. Supongamos que tenemos la siguiente jerarquía:



En este caso podemos tener dos clases con el siguiente código:

```
1    package com.arquitecturajava;
2
3    public class Persona {
4        private String nombre;
5
6       public String getNombre() {
7           return nombre;
        }
```

```
    public void setNombre(String nombre) {
       this.nombre = nombre;
    }
}
```

```
package com.arquitecturajava;

public class Deportista extends Persona{
 }
```

Aunque en el código no aparezcan constructores existen **dos constructores por defecto uno** en cada clase (Persona, Deportista) con el siguiente código:

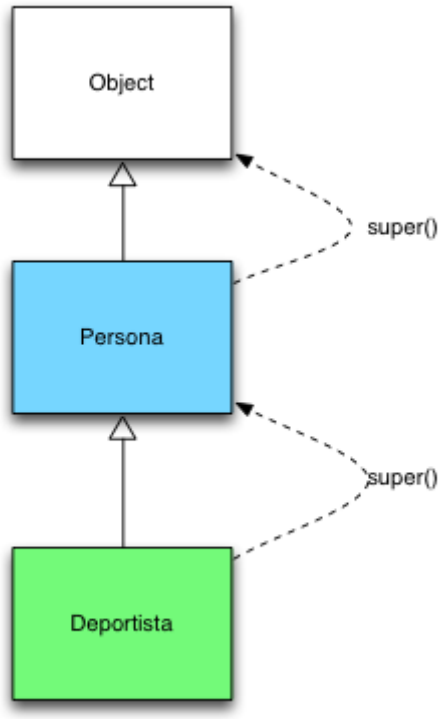
Persona.java

```
public Persona() {
 super();  // invocarlo o no, es lo mismo, Java llama siempre
 }         // a este constructor por defecto de forma implícita
```

Deportista.java

```
public Deportista() {
  super();
}
```

Como podemos ver todos los constructores **llaman por defecto al constructor de la clase superior** a través de una llamada a **super()** (en este caso al constructor por defecto). **Esto es debido a que los constructores no se heredan entre jerarquías de clases.** Por lo tanto la palabra super() siempre es la primera línea de un constructor e invoca al constructor de la clase superior que comparta el mismo tipo de parametrización.



Aunque nosotros no pongamos la palabra super() esta siempre será añadida salvo que nosotros lo hagamos. Por ejemplo si nuestros constructores tienen parámetros, las cláusulas *super* que deberemos construir serán las siguientes entre Persona y Deportista para que el código compile:

```
1
2   package com.arquitecturajava;
3
4   public class Persona {
5
6     public Persona(String nombre) {
7       super();
8       this.nombre = nombre;
9     }
10
11    private String nombre;
12
13    public String getNombre() {
14      return nombre;
15    }
16
17    public void setNombre(String nombre) {
18      this.nombre = nombre;
19    }
  }
```

```
1
2   package com.arquitecturajava;
3
4   public class Deportista extends Persona{
5
6   public Deportista(String nombre) {
7       super(nombre);
8   }
9
10  }
```

Ya que si no el compilador añadirá super() por defecto y el código no compilará al carecer la clase Persona de un constructor por defecto.

Usando this()

La otra posibilidad a super() es el uso de this() en la primera linea de un constructor. Esto lo que hace es invocar a otro constructor que este **en la misma clase** y que soporte el conjunto de parámetros que le pasamos.

```
1   package com.arquitecturajava;
2
3   public class Deportista extends Persona{
4
5      private String deporte;
6
```

```
7    public Deportista(String nombre) {
8       super(nombre);
9    }
10
11   public Deportista(String nombre, String deporte) {
12      this(nombre);
13      this.deporte = deporte;
14   }
15
16   public void setDeporte(String deporte) {
17      this.deporte = deporte;
18   }
19
20    public String getDeporte() {
21      return deporte;
22   }

     }
```

IMPORTANTE

- El uso de this() y de super() es excluyente o usamos uno u otro.
- Los constructores NUNCA devuelven un valor.
- Los constructores NO pueden ser declarados como *static*, *final* ni *abstract*.
- Por regla general se declaran los constructores como públicos (*public*) para que puedan ser utilizados por cualquier otra clase.