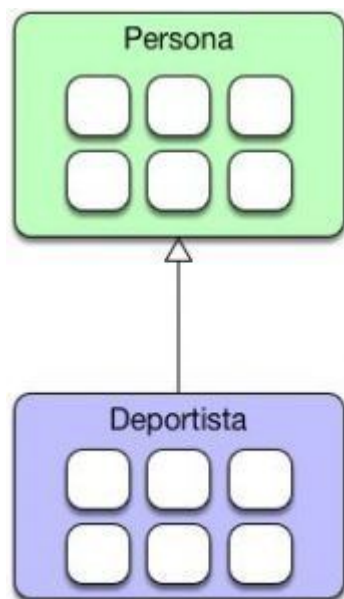


Hablar de Java Herencia y sus limitaciones es muy común cuando nos disponemos a preparar una certificación de Java como la de programador. La mayor parte de los desarrolladores piensa que cuando heredamos o extendemos de una clase se hereda todo.



Sin embargo esto no es así , existen una serie de limitaciones. Vamos a ver algunos ejemplos que nos ayuden a clarificar. Para ello vamos a utilizar la clase Persona y la clase Deportista.

```
package com.arquitecturajava.ejemplo1;
```

```
public class Persona {
```

```
    private String nombre;
```

```
    private int edad;
```

```
    public String getNombre() {
```

```

        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
    public int getEdad() {
        return edad;
    }
    public void setEdad(int edad) {
        this.edad = edad;
    }
    public Persona(String nombre, int edad) {
        super();
        this.nombre = nombre;
        this.edad = edad;
    }
    public void andar() {
        System.out.println(getNombre()+"anda");
    }
    void correr() {
        System.out.println(getNombre()+"corre");
    }
}

```

```
package com.arquitecturajava.ejemplo2;
```

```
import com.arquitecturajava.ejemplo1.Persona;
```

```
public class Deportista extends Persona {
```

```
}
```

Java Herencia

Rápidamente nos daremos cuenta que el código ni siquiera compila y eso que el ejemplo parece muy sencillo. Esto es debido a que los constructores no se heredan y nos veremos obligados a generar nuestro propio constructor que delega en el constructor de Persona.

```
package com.arquitecturajava.ejemplo2;

import com.arquitecturajava.ejemplo1.Persona;

public class Deportista extends Persona {

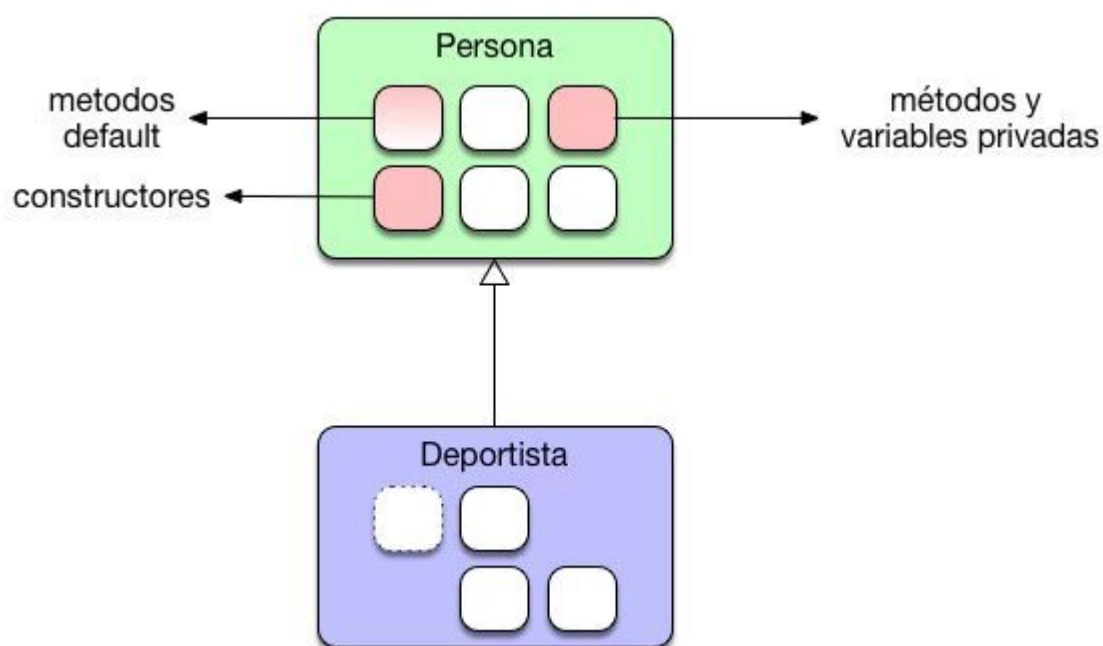
    public Deportista(String nombre, int edad) {
        super(nombre, edad);
    }
}
```

El código nos compila, es momento de sobrescribir el método correr de la clase Persona en el Deportista ya que este corre más rápido.

```
@Override
    public void correr() {
        System.out.println(getNombre()+" corre mas rapido");
    }
```

El código no compila. Esto es debido a que solo se heredan los métodos que tienen visibilidad pública y los métodos de visibilidad default en el caso de que ambas clases estén

en el mismo package . Por lo tanto ya podemos ver que las cosas no son como parecen. Los constructores no se heredan , los métodos que no son públicos tampoco salvo casos puntuales. ¿Existe alguna limitación más? sí , los métodos y variables privadas a los cuales no tendremos acceso.



El uso de java Herencia es muy útil como característica del lenguaje pero a veces hay que tener en cuenta sus limitaciones .

Otros artículos relacionados

1. [Java Multiple Inheritance con default methods](#)
2. [Herencia y relaciones entre objetos](#)
3. [Uso de Herencia y sus problemas](#)
4. [Java Inheritance](#)