Modificadores de acceso, Polimorfismo y Enlazado dinámico

Tabla modificadores de acceso

MODIFICADOR	CLASE	PACKAGE	SUBCLASE	TODOS
Public	Sí	Sí	Sí	Sí
Protected	Sí	Sí	Sí	No
Private	Sí	No	No	No
Por defecto	Sí	Sí	No	No

Los Modificadores de acceso se pueden aplicar a clases, métodos, variables y constantes. Son de vital importancia ya que marcan la accesibilidad del elemento al que se aplique. Por ejemplo, una variable **protected** tal y como marca la tabla anterior, será accesible desde su propia clase, desde su propio paquete, desde una subclase (clase que hereda de la clase donde está declarada la variable) pero no será accesible desde otros paquetes.

El **modificador por defecto** es quizás el más confuso. En ocasiones por descuido del programador no se especifica modificador alguno por lo que se pasa a utilizar el modificador por defecto. Quizás sin saberlo, estamos dando acceso a la variable, método, constante o clase acceso desde otras clases del mismo paquete. Hay que tener especial cuidado con este modificador.

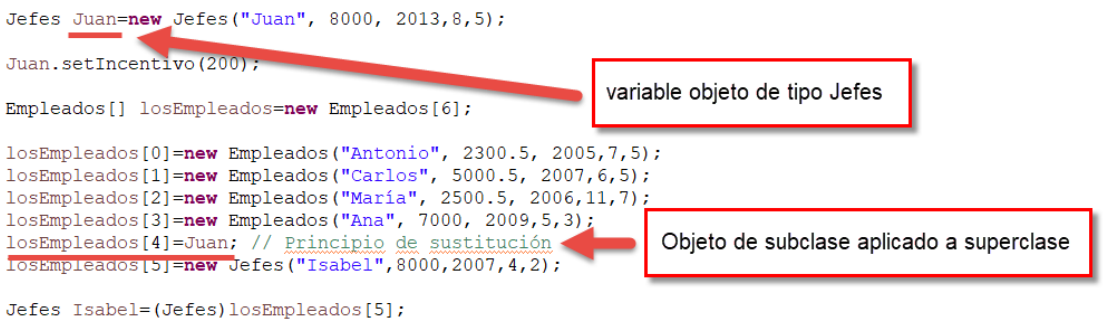
Polimorfismo

Hay una regla sencilla que nos permite saber si la herencia es o no el diseño correcto para nuestros programas, para nuestras clases.

La regla **"es-un"** afirma que todo objeto de la subclase es un objeto de la superclase. Por ejemplo, imaginemos dos clases: Empleados y Jefes. En el caso de que una deba heredar de otra ¿cuál sería el diseño de herencia correcto? ¿Debe heredar la clase Jefes de la clase Empleados? ¿O debe ser al revés? En casos como este es donde debemos aplicar la regla **"es-un"** para que el diseño de la herencia sea correcto. Aplicando dicha regla cabe decir que un Jefe "es-un" Empleado obligatoriamente. Sin embargo, no podemos afirmar lo contrario: un Empleado no "es-un" Jefe obligatoriamente. Así que el diseño correcto debería ser que la clase

Jefes herede de Empleados, o dicho de otra forma, la clase Jefes debe ser subclase de Empleados o Empleados superclase de Jefes.

Otra manera de formular la regla **"es-un"** es el llamado **principio de sustitución**. Este afirma que se puede utilizar un objeto del tipo de la subclase siempre que el programa espere un objeto de la superclase. Por ejemplo, se puede asignar un objeto de subclase a una variable de superclase.

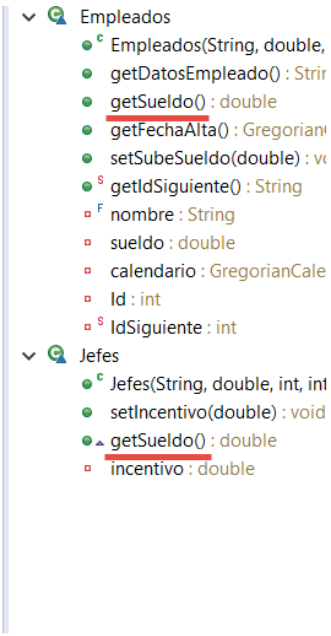


En la imagen anterior vemos como por un lado tenemos un objeto de tipo Jefes llamado "Juan" y por otro lado tenemos un Array de tipo Empleados llamado "losEmpleados". Se puede observar como se aplica el **principio de sustitución** asignando el objeto de tipo Jefes a la variable de tipo Empleados. Este fenómeno es lo que se conoce como polimorfismo: una variable de tipo Empleados puede referirse a un objeto de tipo Empleados o a cualquier objeto de una subclase de Empleados (en este caso Jefes).

Enlazado dinámico

En **enlazado dinámico** es otra de las características de Java que dota a este lenguaje de potentes capacidades que parecen casi mágicas. El enlazado dinámico está íntimamente relacionado con el polimorfismo y es el proceso mediante el cual el intérprete Java conoce de qué tipo es una variable polimórfica en tiempo de ejecución (durante la ejecución de un programa, de ahí lo de dinámico). Pongamos un ejemplo para entenderlo mejor:

```
Jefes Juan=new Jefes("Juan", 8000, 2013,8,5);

Juan.setIncentivo(200);

Empleados[] losEmpleados=new Empleados[6];

losEmpleados[0]=new Empleados("Antonio", 2300.5, 2005,7,5);
losEmpleados[1]=new Empleados("Carlos", 5000.5, 2007,6,5);
losEmpleados[2]=new Empleados("María", 2500.5, 2006,11,7);
losEmpleados[3]=new Empleados("Ana", 7000, 2009,5,3);
losEmpleados[4]=Juan; // Principio de sustitución
losEmpleados[5]=new Jefes("Isabel",8000,2007,4,2);

Jefes Isabel=(Jefes)losEmpleados[5];

Isabel.setIncentivo(500);


for (Empleados obj : losEmpleados){


    System.out.println(obj.getDatosEmpleado() +
            " y un salario de: " + obj.getSueldo());

}
```



Como se observa en la imagen anterior, tanto la clase Empleados como la subclase Jefes tienen un método llamado getSueldo(). También vemos como tenemos un objeto llamado “obj” que es de tipo Empleados. La pregunta que surge es: ¿a cuál de los dos métodos getSueldo llama el objeto “obj” a cada vuelta de bucle for-each? Lo normal sería pensar que llama el método getSueldo() de la clase Empleados. Sin embargo, dentro del bucle for-each el **enlazado dinámico** entra en acción debido a la naturaleza **polimórfica** del objeto “obj”. A cada vuelta de bucle el objeto “obj” se comportará de diferente forma, unas veces como Empleados y otra como Jefes dando lugar al enlazado dinámico proceso por el cual en ocasiones llamará al método getSueldo() de la clase Empleados y en otras ocasiones al método getSueldo() de la clase Jefes, dependiendo dela forma que tome obj.