Resumen PDOO

Herencia

Usualmente, la clase hija hereda **TODO** el código de la clase padre, pero esto **NO IMPLICA** que se pueda acceder a cualquier elemento de la clase padre, el acceso depende de la visibilidad. Se redefine cuando una clase proporciona una implementación alternativa a la que ha heredado.

Super

En *Java*, en el **constructor** debe aparecer en la primera línea, y se puede llamar a cualquier método de la clase padre `super.metodo1()`.

En *Ruby*, solo se puede acceder al **método con el mismo nombre**. Si no se le dan argumentos se usan los mismos que los que tiene el método que se está redefiniendo. En el método initialize es como si no hiciese nada.

Particularidades

Java

- Hay sobrecarga de métodos.
- No se redefinen los métodos privados ni final.
- Se puede aumentar la accesibilidad en las clases hijas o cambiar el valor del tipo retornado (puede ser una subclase).

Ruby

- No hay sobrecarga, luego al crear un método con el mismo nombre directamente se redefine (no hay sobrecarga).
- Podemos tener ausencia de initialize.
- La responsabilidad de llamar a super en initialize es **nuestra** (si no, no se llama, no se crean los valores de las variables).

Visibilidad

Java

- **private** solo es accesible desde el código de la propia clase (ámbito de instancia o de clase).
- Desde instancias se puede acceder a elementos privados de la clase o de otras instancias distintas de la misma clase.
- **package** indica que es *público dentro del paquete* y privados fuera.
- **protected** es público dentro del paquete (con independencia de la herencia o no que exista), y *accesible* desde *subclases de otros paquetes*. Para acceder a elementos protegidos de una **instancia distinta** (ambito clase/instancia), la instancia debe ser de la misma clase que la **propietaria** del código desde el que se realiza

el acceso o de una subclase de de esta y además el elemento debe estar declarado en la clase propietaria del código desde el que se realiza el acceso o en una superclase de la misma.

Es decir, si se llama desde `Hija h1` en otro paquete, para poder acceder a un protected debe ser un `Hija h2` o un `Nieta n1`, no vale un `Padre p` aunque una hija sea un padre. Además lo que se quiere acceder debe estar al menos en la *clase hija o en una superclase* (no puedo acceder a un protegido de una subclase).

```
package base;
public class A { protected int protegidoA = 0;}
public class B extends A {protected int protegidoB = 1};

import base2.C;
public class D extends C { protected int protegidoD = 3;}

//****************************************

package base2;
import package base.*;

public class C extends B {
    protected int protegidoC = 2;

    public void test(){
        A a = new A();
        a.protegidoA = 666;
        // Fallo, otro paquete, protegido, es de un Abuelo

        B b = new B();
        b.protegidoB = 666;
        // Fallo, otro paquete, protegido, es de un padre

        C c = new C();
        c.protegidoA = 555;
        // Correcto, otro paquete, protegido, es de la misma clase
        // que desde donde se intenta acceder

        D d = new D();
        d2.protegidoB = 555;
        // Correcto, otro paquete, protegido, es de una subclase
        // que desde donde se intenta acceder

        d2.protegidoD = 555;
        // Fallo, otro paquete, protegido, el atributo está declarado en una subclase



    }
}
```

Ruby

- *Atributos* e *initialize* **siempre privados**, métodos públicos pero se puede cambiar. Un especificador afecta a **todo lo que viene después**.

- **private**, un método privado no puede ser usado como receptor de mensaje explícito, salvo `self`. Solo se puede usar un método privado de la propia instancia. Si $B < A$, desde *ámbito de instancia* (respectivamente clase) de **B** se puede llamar a métodos de instancia (respectivamente clase) *privados* de A.

  No se puede llamar a métodos privados de *clase (resp. instancia)* desde ámbito de instancia (resp. clase).

  En resumen: se puede llamar a métodos privados de super clases sin mezclar los ámbitos.

- Los métodos **protected** pueden ser invocados con receptor de mensaje explícito (debe ser la misma o una subclase). *No existen* métodos protegidos *de clase*.