Control de acceso y herencia

Al utilizar el mecanismos de herencia y por tanto subclases, aparece *protected* en el escenario del control de acceso a miembros

De la web oficial extraemos la siguiente tabla, que debes interpretar con las explicaciones de abajo. Observa que la explicación oficial analiza la tabla de columna en columna. No es una tabla muy clara pero recoge por completo todas las posibilidades.

Access Levels

Modifier	Class	Package	Subclass	World
`public`	Y	Y	Y	Y
`protected`	Y	Y	Y	N
no modifier	Y	Y	N	N
`private`	Y	N	N	N

The first data column indicates whether the class itself has access to the member defined by the access level. As you can see, a class always has access to its own members. The second column indicates whether classes in the same package as the class (regardless of their parentage) have access to the member. The third column indicates whether subclasses of the class declared outside this package have access to the member. The fourth column indicates whether all classes have access to the member.

Access levels affect you in two ways. First, when you use classes that come from another source, such as the classes in the Java platform, access levels determine which members of those classes your own classes can use. Second, when you write a class, you need to decide what access level every member variable and every method in your class should have.

Para hacer un análisis columna a columna podemos leer la tabla de una forma parecida a esta:

- 1ª col describe los accesos entre miembros que pertenecen todos a la misma clase. No hay restricciones, es todo "*yes*".

  En las siguientes columnas tenemos que pensar en accesos entre dos clases diferentes, vamos a pensar que desde un metodoB() de una clase B quiero acceder a un métodoA() de una clase A.

- 2ª col. Las dos clases pertenecen al mismo paquete, entonces pensando en nuestro metodoA() y metodoB():
  - Si metodoA() es público metodoB() tiene acceso
  - Si metodoA() es protected metodoB() tiene acceso
  - Si metodoA() es no modifier metodoB() tiene acceso
  - Si metodoA() es private metodoB() **no** tiene acceso

  Conclusión: dentro de un paquete hay acceso libre excepto para private

- 3ªcol. Ahora suponemos que B y A están en paquetes diferentes PERO B ES SUBCLASE DE A.
  - Si metodoA() es público metodoB() tiene acceso
  - Si metodoA() es protected metodoB() tiene acceso

- Si metodoA() es no modifier o private metodoB() no tiene acceso

- 4ª col. Ahora B y A están en paquetes diferentes y no hay relación de herencia entre ellas:

  Sólo es posible si metodoA() es public (y logicamente la clase A también tiene que ser public)

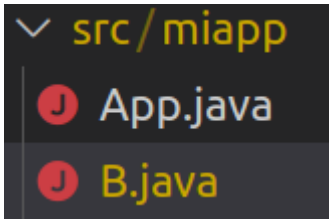
A continuación vamos a analizar de forma práctica "la 3ª columna"

CONTROL DE ACCESO PROTECTED

Un miembro protected es accesible a todas las clases de su paquete y para todas las subclases aunque pertenezcan a paquetes diferentes . Está a caballo entre public y package. Es más restrictivo que *public* pero menos que *package* ya que hay la posibilidad de acceso entre paquetes diferentes. Mejor un ejemplo...

Por ser más gráfico, utilizaremos IDE para las explicaciones de este boletín.

Ejemplo: Comprueba como dentro del mismo paquete es posible el acceso protected




```
//B.java
package miapp;
class B{
   private int w=1;
   int x=1;
   protected int y=2;
   public int z= 3;
}
```

```
//App.java
package miapp;

class App{
   public static void main(String[] args) {
       B b =new B();
```

```
      //System.out.println(b.w); // error por ser acceso private
      System.out.println(b.x);
      System.out.println(b.y); //es posible el acceso protected
      System.out.println(b.z);
   }
}
```

Observa que en el ejemplo anterior no hay subclases. No hay relación de herencia entre App y B, la relación que tienen es que pertenecen al mismo paquete, el paquete miapp. Si no hay relación de herencia, los accesos protected y el acceso paquete son equivalentes.

Ejemplo: comprueba que si cambiamos la clase B a un paquete diferente, App sólo puede acceder a los miembros public.

La clase B ahora la escribimos en el paquete *otropaquete* y debe ser public

```
package otropaquete;
public class B{
   private int w=1;
   int x=1;
   protected int y=2;
   public int z= 3;
}
```

La clase App ahora debe indicar siempre otropaquete.B o bien usar import

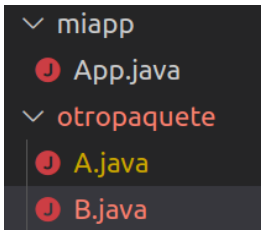
package App;

```
//App.java
package miapp;

import otropaquete.B;

class App{
  public static void main(String[] args) {
     B b =new B();
     //System.out.println(b.w); // error por ser acceso private
     System.out.println(b.x); //error porque x tiene acceso paquete
     System.out.println(b.y); //error porque y tiene acceso protected
     System.out.println(b.z);
  }
}
```

Observaremos que App.java nos presenta errores de acceso excepto para z que es un miembro public

Ahora vamos a analizar cómo actúan los niveles de acceso cuando hay relación de Herencia

Ejemplo: Comprobamos que una subclase no puede acceder a un miembro private de su superclase. Trabajamos con la siguiente estructura, en la que B extiende a A




```
//A.java
package otropaquete;
public class A {
  private int n=5;
}
```

```
//B.java
package otropaquete;
public class B extends A{
  public int x=1;
  public int getN(){return n;}
}
```

```
//App.java
package miapp;

import otropaquete.B;

class App{
  public static void main(String[] args) {
     B b =new B();

     System.out.println(b.x);
     System.out.println(b.getN());

  }
}
```

Observa:

- aunque B sea subclase de A, no puede acceder al miembro *n* de A por ser private
- App.java No tiene errores de compilación PERO NO SE PUEDE EJECUTAR ya que B tiene errores de compilación.

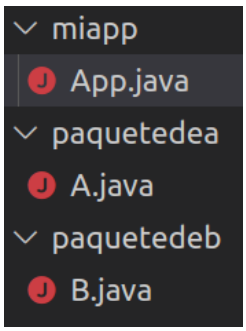
Si ahora cambiamos private por protected o modo paquete desaparece el error como ya vimos

```
//A.java
package otropaquete;

public class A {
    protected int n=5;
}
```

Ejemplo: Comprobamos que una subclase puede acceder a un miembro protected de su superclase aunque pertenezcan a paquetes diferentes.

Aquí está la diferencia entre package y protected por estar en paquetes diferentes.

Si A es superclase de B y A y B y están en paquetes diferentes desde B se puede acceder a algo protected de A




```
//B.java
package paquetedeb;
import paquetedea.A;
public class B extends A{
  public int x=1;
  public int getN(){return n;}
}
```

```
//B.java
package paquetedeb;
import paquetedea.A;
public class B extends A{
```

```
  public int x=1;
  public int getN(){return n;}
}


//App.java
package miapp;

import paquetedeb.B;

class App{
  public static void main(String[] args) {
     B b =new B();

     System.out.println(b.x);
     System.out.println(b.getN());

  }
}
```

QUE USAR: ¿PUBLIC, PROTECTED O PRIVATE?

La explicación perfecta viene en:

http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

pegamos la explicación:

Tips on Choosing an Access Level:

If other programmers use your class, you want to ensure that errors from misuse cannot happen. Access levels can help you do this.

- *Use the most restrictive access level that makes sense for a particular member. Use private unless you have a good reason not to.*
- *Avoid public fields except for constants. (Many of the examples in the tutorial use public fields. This may help to illustrate some points concisely, but is not recommended for production code.) Public fields tend to link you to a particular implementation and limit your flexibility in changing your code.*

Una conclusión de lo anterior: se deben evitar los atributos public(salvo constantes), incluso los protected. Los atributos siempre deben ser private y se accedería a ellos a través de métodos protected o public. Pero en pequeños programas, para evitar el tedio de métodos set y get se relaja esta restricción.