IMPLEMENTACIÓN DE CLASES

AUTOR: WALTER MADRIGAL CHAVES

NOVIEMBRE: 2020

San Marcos

Contenido

INTRODUCCIÓN

El paradigma de la programación modular, en especial la programación orientada a objetos, son tendencias con mucho auge en el mercado actual, sus características de robustez lo convierten en poderosas herramientas para solucionar problemas complejos.

Dentro de las principales preocupaciones en la construcción de grandes proyectos bajo la modalidad POO, están: como tener un control efectivo de los objetos, como hacer para limitar su uso y dar libertad de aplicarlo solo a un grupo de elementos, es por lo que se concretó el termino de encapsulamiento de datos.

Esta lectura además de analizar de una forma general el procedimiento de encapsulamiento, se analizará algunas técnicas propias de este término como lo son la composición, la delegación y las clases envoltorio.

ENCAPSULAMIENTO

Para (Blasco, 2020) el encapsulamiento es una propiedad de la programación orientada a objetos cuya finalidad es impedir la accesibilidad a los atributos de un objeto desde el exterior de dicho objeto; de tal modo que el acceso a dichos atributos solamente sea posible desde los métodos de dicho objeto.

Visto desde otro punto de vistas el encapsulamiento es un mecanismo que organiza los datos y métodos de una estructura, acordando entre las partes el modo en que el objeto se implementará, es decir, evitando el acceso a datos por cualquier otro medio distinto a los permitidos. Por lo tanto, la encapsulación garantiza la integridad de los datos que contiene un objeto.

Además, el encapsulamiento da control en el manejo de la información ya que no es necesario conocer los detalles de cómo están implementadas las propiedades para poder utilizarlas, es como tener a las clases en una caja negra en donde sólo se conoce el comportamiento, pero no los detalles internos.

AMBITO DE ACCESO

El encapsulamiento busca de alguna forma controlar el acceso a los datos, para esto usa los modificadores de acceso, que son mecanismos que permiten dar un nivel de seguridad mayor a las aplicaciones, restringiendo el acceso a diferentes atributos, métodos y constructores y asegurándose con esto, que el usuario siga una ruta específica para acceder a la información.

Modificador de acceso Private

Este es el más prohibitivo de todos, básicamente cualquier elemento de una clase que sea privado puede ser accedido únicamente por la misma clase. Es decir, si, por ejemplo, un atributo es privado solo puede ser accedido por lo

métodos o constructores de la misma clase. Ninguna otra clase sin importar la relación que tengan podrá tener acceso a ellos.

Imagen 1 Ámbito privado en clase

```
1  package ambito_privado;
2  public class ejerciciol
3  {
4      private int atributol;//Este atributo es privado
5
6      //Si un atributo es privado podemos crear método get y set ...
7      //... para éste y permitir el acceso a él desde otras instancias
8
9      public void setAtributol(int valor)
10     {
11         atributol = valor;//Establecemos el valor del atributo
12     }
13
14     public int getAtributol()
15     {
16         return atributol;//Retornamos el valor actual del atributo
17     }
18
19 }
```

Fuente: Elaboración propia.

En la primera parte de la imagen 1 se declara un atributo con un ámbito privado, esto provoca que no sea accesible desde el exterior de la clase. En la parte inferior del código se declara dos métodos Set y Get, ambos con ámbito público que soluciona el problema de la visibilidad del atributo privado. Estos métodos permiten referenciar el atributo privado desde el exterior de la clase.

Modificador de acceso Public

El modificador de acceso public es el más permisivo de todos, si un componente de una clase es public, se tendrá acceso a él desde cualquier otra clase o instancia sin importar el paquete o procedencia de ésta.

Imagen 2 Creación de atributo con ámbito público

```
1   package ambito_publico;
2
3   public class ejercicio2
4   {
5       public static int atributo1; //Atributo publico
6
7       public static void metodo1()
8       {
9           System.out.println("Método publico");
10      }
11  }
```

Fuente: Elaboración propia.

Imagen 3 Invocación al atributo público

```
1   package paquete.externo;
2
3   import ambito_publico.ejercicio2; //importamos la clase del ejemplo4
4
5   public class ClaseExterna
6   {
7       public static void main(String[] args)
8       {
9           System.out.println(ejercicio2.atributo1);
10          //se puede acceder directamente al atributo1 por ser público
11
12          ejercicio2.metodo1(); //Metodo1 también es publico
13      }
14  }
```

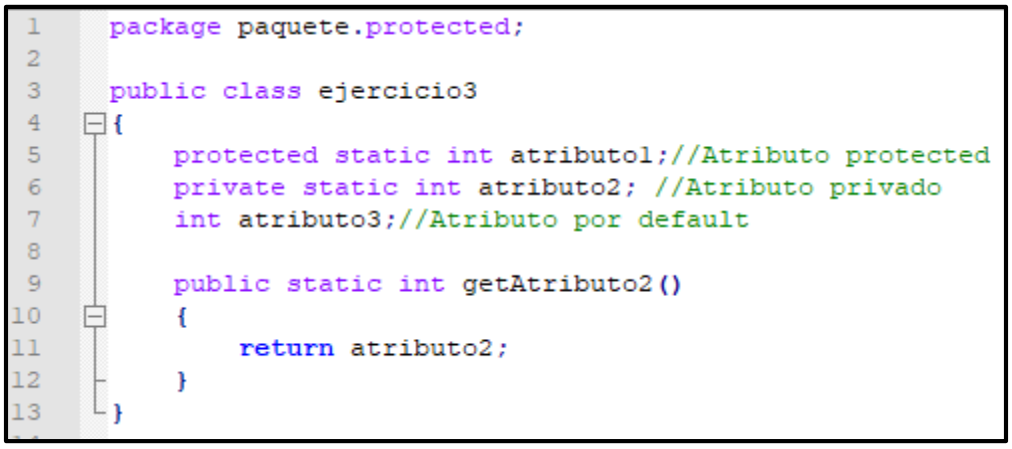
Fuente: Elaboración propia.

En la imagen 2 se crea un atributo y un método con un ámbito público, lo que contempla este tipo de asignación es la posibilidad de invocarlos desde cualquier otra parte del programa. Así se realiza el ejemplo en la imagen 3 en donde en otra clase externa se llama al atributo y método públicos, estos responden sin problemas.

Modificador de acceso Protected

El modificador de acceso protected permite acceso a los componentes con dicho modificador desde la misma clase, clases del mismo paquete y clases que hereden de ella (incluso en diferentes paquetes).

Imagen 4 Creación de atributo con ámbito Protected



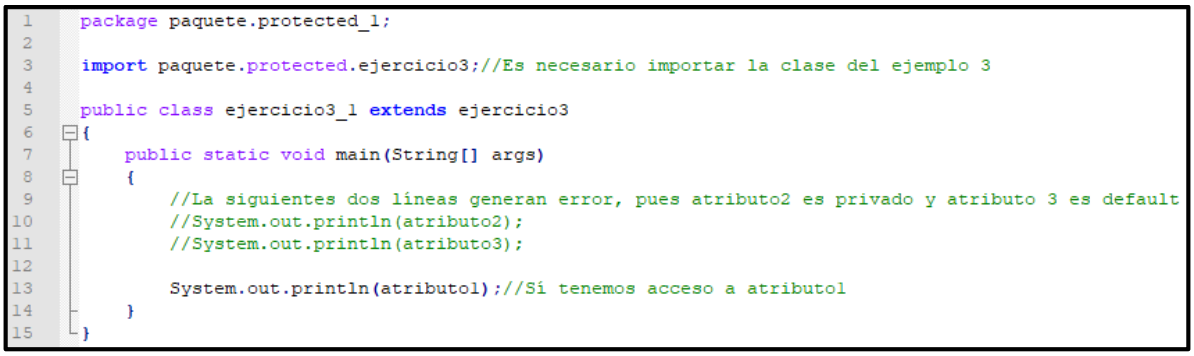
```
package paquete.protected;

public class ejercicio3
{
    protected static int atributo1;//Atributo protected
    private static int atributo2; //Atributo privado
    int atributo3;//Atributo por default

    public static int getAtributo2()
    {
        return atributo2;
    }
}
```

Fuente: Elaboración propia.

Imagen 5 Invocación al atributo protected



```
package paquete.protected_1;

import paquete.protected.ejercicio3;//Es necesario importar la clase del ejemplo 3

public class ejercicio3_1 extends ejercicio3
{
    public static void main(String[] args)
    {
        //La siguientes dos lineas generan error, pues atributo2 es privado y atributo 3 es default
        //System.out.println(atributo2);
        //System.out.println(atributo3);

        System.out.println(atributo1);//Sí tenemos acceso a atributo1
    }
}
```

Fuente: Elaboración propia.

En la imagen 4 se crea un atributo con ámbito de acción Protected, por tal motivo solo puede ser invocado dentro de la misma clase, paquete o como la imagen 5 que se utiliza desde una clase heredada.

Cuando no se especifica el modificador del ámbito, se aplica el ámbito por defecto, esto implica que está accesible desde cualquier atributo, método y clase, a los que se le haya aplicado el modificador de ámbito y desde cualquier otra parte de la clase del mismo paquete.

METODOS GET Y SET

Sus nombres reales son Setters y Getters, son métodos de acceso, lo que representa que por lo general son una interfaz pública para cambiar miembros de las clases privadas. se utilizan para definir una propiedad, a estos se accede como propiedades situadas fuera de la clase, aunque las defina dentro de la clase como métodos.

Una buena práctica de programación es declarar las variables como privadas y luego acceder

- **Set:** sirve para asignar un valor inicial a un atributo, se realiza de forma explícita, permite dar acceso público a ciertos atributos, es decir, permiten cambiar el valor de los atributos.
- **Get:** sirve para obtener (recuperar o acceder) el valor ya asignado a un atributo y utilizarlo para cierto método. Por lo que devuelven el valor de los atributos.

Imagen 6 Implementación de Get y Set

```
1  public class Sample {
2    private String  name;
3    private int age;
4
5    public int getAge() {
6      return age;
7    }
8
9    public void setAge(int age) {
10     this.age = age;
11   }
12
13   public String getName() {
14     return name;
15   }
16
17   public void setName(String name) {
18     this.name = name;
19   }
20 }
```

Fuente: Elaboración propia.

Como se muestra en la imagen 6, las variables "name" y "age", son privadas y no se pueden acceder desde otra fuente externa, es por eso que se declaran los métodos Get (en la línea 5), para retornar el valor de la variable "age" y el método Set para modificar el valor de esta variable desde una fuente externa.

COMPOSICIÓN

Según (Blasco, 2020) Implementar composición supone básicamente, que, desde un objeto tenemos, como atributos, referencias a otros objetos. Puede dar la impresión equivocada de que el objeto de contiene a otros. En realidad, no es así. Lo que el objeto "contenedor" tiene de los "contenidos" son las referencias a dichos objetos "contenidos" como atributos.

En términos más sencillos se dice que composición es el agrupamiento de uno o varios objetos y valores, que conforman el valor de los distintos objetos de una clase. Normalmente, los atributos contenidos se declaran con acceso privado (private) y se inicializan en el constructor de la nueva clase.

DELEGACIÓN

Es un patrón de diseño que se puede ver como una manera de realizar herencia múltiple manualmente mediante composición de objetos, se trata de una técnica en la que un objeto permite mostrar cierto método al exterior, pero internamente la implementación o las acciones desencadenadas por el llamado de este método se delega a otro objeto de una clase distinta pero asociado.

La delegación es utilizada como un mecanismo para centralizar en torno a un solo objeto, los compartimentos(métodos) de varios objetos donde dichos comportamientos mantienen cierto nivel de relación.

CLASES ENVOLTORIO

Existen una serie de tipos de datos llamados primitivos, algunos de ellos son:

byte, short, int, long, float, double, char y boolean. Estos tipos de datos son llamados así por ser los más básicos en Java. En ocasiones es necesario utilizar estos datos como objetos, por lo que se crearon las clases envoltorio.

Para (Blasco, 2020), Las clases de tipo envoltorio son clases predefinidas, cuyo fin es aportar encapsulamiento a un tipo primitivo. De tal modo que, un objeto de clase envoltorio, permite acceso mediante sus métodos al dato de tipo primitivo que encapsula.

Este tipo de clases son muy útiles para dotar a los datos primitivos (int, boolean, etc.) de un envoltorio (wrapper class) que permite tratarlos como objetos. Por ejemplo, se pueden definir una clase envoltorio de la siguiente manera:

- Byte nombreVariable=new Byte(parámetros);
- Short nombreVariable=new Short(parámetros);
- Integer nombreVariabe=new Integer(parámetros);
- Long nombreVariable=new Long(parámetros);
- Float nombreVariable=new Float(parámetros);
- Double nombreVariable=new Double(parámetros);
- Character nombreVariable=new Character(parámetros);
- Boolean nombreVariable=new Boolean(parámetros);

Para ahondar más en el tema de Encapsulación y clases especiales debe realizar la lectura de las páginas 121 a la 193 del libro: Programación orientada a objetos con Java. (2020) de Francisco Blasco.

CONCLUSIONES Y RECOMENDACIONES

- El encapsulamiento es muy importante en POO, protege la integridad de los objetos, hay mejor control sobre los elementos y reduce la complejidad del proyecto.

- Cuando se utilice una clase de otro paquete, se debe importar usando la palabra reservada import. Cuando dos clases se encuentran en el mismo paquete no es necesario hacer el import, pero esto no significa que se pueda acceder a sus componentes directamente.

- Es gracias a los modificadores de acceso que se puede asegurar de que un valor no será modificado incorrectamente por parte de otro programador o usuario.

- Siempre se recomienda que los atributos de una clase sean privados y por tanto cada atributo debe tener sus propios métodos get y set para obtener y establecer respectivamente el valor del atributo.

- Las variables primitivas tienen mecanismos de reserva y liberación de memoria más eficaces y rápidos que los objetos por lo que deben usarse datos primitivos en lugar de sus correspondientes envolturas siempre que se pueda.

REFERENCIAS BIBLIOGRÁFICAS

Abellán, B. (11 de 15 de 2020). *Programacion Orientada a Objetos* . Obtenido de http://picarcodigo.blogspot.com/2012/12/clases-envoltorio.html

Barnes, D., & Kölling, M. (2017). *Programación oerientada a objetos con Java usando Bluej 6a. Ed.* Madrid: Pearson.

Blasco, F. (2020). *Programación orientada a objetos en Java.* Bogotá: Ediciones la U.

San Marcos

www.usanmarcos.ac.cr

San José, Costa Rica