5.D. Encapsulación, control de acceso y visibilidad.

1. Encapsulación, control de acceso y visibilidad.

1.1. Ocultación de atributos. Métodos de acceso.

Los atributos de una clase suelen ser declarados como privados a la clase o, como mucho, `protected` (accesibles también por clases heredadas), pero no como `public`. De esta manera puedes evitar que sean manipulados inadecuadamente (por ejemplos modificarlos sin ningún tipo de control) desde el exterior del objeto.

En estos casos lo que se suele hacer es declarar esos atributos como privados o protegidos y crear métodos públicos que permitan acceder a esos atributos. Si se trata de un atributo cuyo contenido puede ser observado pero no modificado directamente, puede implementarse un método de "obtención" del atributo (en inglés se les suele llamar método de tipo **get**) y si el atributo puede ser modificado, puedes también implementar otro método para la modificación o "establecimiento" del valor del atributo (en inglés se le suele llamar método de tipo **set**). Esto ya lo has visto en apartados anteriores.

Si recuerdas la clase `Punto` que hemos utilizado como ejemplo, ya hiciste algo así con los métodos de obtención y establecimiento de las coordenadas:

```
private int x, y;
```

```
// Métodos get
```

```
public int obtenerX () { return x; }
```

```
public int obtenerY () { return y; }
```

```
// Métodos set
```

```
public void establecerX (int x) { this.x= x; }
```

```
public void establecerY (int y) { this.y= y; }
```

Así, para poder obtener el valor del atributo `x` de un objeto de tipo `Punto` será necesario utilizar el método `obtenerX()` y no se podrá acceder directamente al atributo `x` del objeto.

En algunos casos los programadores directamente utilizan nombres en inglés para nombrar a estos métodos: `getX ,getY (), setX, setY, getNombre, setNombre, getColor`, etc.

También pueden darse casos en los que no interesa que pueda observarse directamente el valor de un atributo, sino un determinado procesamiento o cálculo que se haga con el atributo (pero no el valor original). Por ejemplo podrías tener un atributo **DNI** que almacene los 8 dígitos del DNI pero no la letra del **NIF** (pues se puede calcular a partir de los dígitos). El método de acceso para el DNI (método `getDNI`) podría proporcionar el DNI completo (es decir, el NIF, incluyendo la letra), mientras que la letra no es almacenada realmente en el atributo del objeto. Algo similar podría suceder con el **dígito de control de una cuenta bancaria**, que puede no ser almacenado en el objeto, pero sí calculado y devuelto cuando se nos pide el número de cuenta completo.

En otros casos puede interesar disponer de métodos de modificación de un atributo pero a través de un determinado procesamiento previo para por ejemplo poder controlar errores o valores inadecuados. Volviendo al ejemplo del NIF, un método para modificar un DNI (método `setDNI`) podría incluir la letra (NIF completo), de manera que así podría comprobarse si el número de DNI y la letra coinciden (es un NIF válido). En tal caso se almacenará el DNI y en caso contrario se producirá un error de validación (por ejemplo lanzando una excepción). En cualquier caso, el DNI que se almacenara sería solamente el número y no la letra (pues la letra es calculable a partir del número de DNI).

Autoevaluación

Los atributos de una clase suelen ser declarados como public **para facilitar el acceso y la visibilidad de los miembros de la clase. ¿Verdadero o falso?**

- ○ Verdadero.
- ○ Falso.