7.B. Composición.

1. Composición.

1.2. Uso de la composición (I). Preservación de la ocultación.

Como ya has observado, la relación de **composición** no tiene más misterio a la hora de implementarse que simplemente declarar **atributos** de las clases que necesites dentro de la clase que estés diseñando.

Ahora bien, cuando escribas clases que contienen objetos de otras clases (lo cual será lo más habitual) deberás tener un poco de precaución con aquellos métodos que devuelvan información acerca de los **atributos** de la clase (métodos "**obtenedores**" o de tipo **get**).

Como ya viste en la unidad dedicada a la creación de clases, lo normal suele ser declarar los **atributos** como **privados** (o **protegidos**, como veremos un poco más adelante) para ocultarlos a los posibles **clientes** de la clase (otros objetos que en el futuro harán uso de la clase). Para que otros objetos puedan acceder a la información contenida en los **atributos**, o al menos a una parte de ella, deberán hacerlo a través de **métodos que sirvan de interfaz**, de manera que sólo se podrá tener acceso a aquella información que el creador de la clase haya considerado oportuna. Del mismo modo, los **atributos** solamente serán modificados desde los métodos de la clase, que decidirán cómo y bajo qué circunstancias deben realizarse esas modificaciones. Con esa metodología de acceso se tenía perfectamente separada la parte de manipulación interna de los atributos de la **interfaz** con el exterior.

Hasta ahora los métodos de tipo **get** devolvían **tipos primitivos**, es decir, copias del contenido (a veces con algún tipo de modificación o de formato) que había almacenado en los **atributos**, pero los atributos seguían "a salvo" como elementos privados de la clase. Pero, a partir de este momento, al tener objetos dentro de las clases y no sólo tipos primitivos, es posible que en un determinado momento interese devolver un **objeto completo**.

Ahora bien, cuando vayas a devolver un objeto habrás de obrar con mucha precaución. Si en un método de la clase devuelves directamente un objeto que es un atributo, estarás ofreciendo directamente una **referencia** a un objeto atributo que probablemente has definido como privado. ¡De esta forma estás **volviendo a hacer público un atributo que inicialmente era privado**!

Para evitar ese tipo de situaciones (ofrecer al exterior referencias a objetos privados) puedes optar por diversas alternativas, procurando siempre evitar la devolución directa de un atributo que sea un objeto:

- Una opción podría ser devolver siempre tipos primitivos.

- Dado que esto no siempre es posible, o como mínimo poco práctico, otra posibilidad es crear un nuevo objeto que sea una copia del atributo que quieres devolver y utilizar ese objeto como valor de retorno. Es decir, **crear una copia del objeto** especialmente para devolverlo. De esta manera, el código cliente de ese método podrá manipular a su antojo ese nuevo objeto, pues no será una referencia al atributo original, sino un nuevo objeto con el mismo contenido.

Por último, debes tener en cuenta que es posible que en algunos casos sí se necesite realmente la referencia al atributo original (algo muy habitual en el caso de atributos estáticos). En tales casos, no habrá problema en devolver directamente el atributo para que el código llamante (cliente) haga el uso que estime oportuno de él.

Debes evitar por todos los medios la devolución de un atributo que sea un objeto (estarías dando directamente una referencia al atributo, visible y manipulable desde fuera), salvo que se trate de un caso en el que deba ser así.

Para entender estas situaciones un poco mejor, podemos volver al objeto `Rectangulo` y observar sus nuevos métodos de tipo **get**.

Ejercicio resuelto

Dada la clase `Rectangulo`, escribe sus nuevos métodos `obtenerVertice1` y `obtenerVertice2` para que devuelvan los vértices inferior izquierdo y superior derecho del rectángulo (objetos de tipo `Punto`), teniendo en cuenta su nueva estructura de atributos (dos objetos de la clase `Punto`, en lugar de cuatro elementos de tipo `double`):

Solución:

Los métodos de obtención de vértices devolverán objetos de la clase **Punto**:

```
public Punto obtenerVertice1 ()
{
    return vertice1;
```

```
}

public Punto obtenerVertice2 ()
{
    return vertice2;
}
```

Esto funcionaría perfectamente, pero deberías tener cuidado con este tipo de métodos que devuelven directamente una referencia a un objeto atributo que probablemente has definido como privado. Estás de alguna manera haciendo público un atributo que fue declarado como privado.

Para evitar que esto suceda bastaría con crear un nuevo objeto que fuera una copia del atributo que se desea devolver (en este caso un objeto de la clase Punto).

Aquí tienes algunas posibilidades:

```
public Punto obtenerVertice1 ()   // Creación de un nuevo punto extrayendo sus atributos
{
    double x, y;
    Punto p;
    x= vertice1.obtenerX();
    y= vertice1.obtenerY();
    p= new Punto (x,y);

    return p;
}

public Punto obtenerVertice1 ()  // Utilizando el constructor copia de Punto (si es que está definido)
{
    Punto p;
    p= new Punto (this.vertice1); // Uso del constructor copia
    return p;
}
```

De esta manera, se devuelve un punto totalmente nuevo que podrá ser manipulado sin ningún temor por parte del código cliente de la clase pues es una copia para él.

Para el método obtenerVertice2 sería exactamente igual.