

7.B. Composición.

1. Composición.

1.1. Sintaxis de la composición.

Para indicar que una clase contiene objetos de otra clase no es necesaria ninguna sintaxis especial. Cada uno de esos objetos no es más que un atributo y, por tanto, debe ser declarado como tal:

```
class <nombreClase> {  
    [modificadores] <NombreClase1> nombreAtributo1;  
    [modificadores] <NombreClase2> nombreAtributo2;  
    ...  
}
```



En unidades anteriores has trabajado con la clase **Punto**, que definía las coordenadas de un punto en el plano, y con la clase **Rectangulo**, que definía una figura de tipo rectángulo también en el plano a partir de dos de sus **vértices** (**inferior izquierdo** y **superior derecho**). Tal y como hemos formalizado ahora los tipos de relaciones entre clases, parece bastante claro que aquí tendrías un caso de **composición**: “un rectángulo contiene puntos”. Por tanto, podrías ahora redefinir los atributos de la clase **Rectangulo** (cuatro **números reales**) como dos objetos de tipo **Punto**:

```
class Rectangulo {  
    private Punto vertice1;  
    private Punto vertice2;  
    ...  
}
```

Ahora los métodos de esta clase deberán tener en cuenta que ya no hay cuatro atributos de tipo **double**, sino dos atributos de tipo **Punto** (cada uno de los cuales contendrá en su interior dos atributos de tipo **double**).

Autoevaluación

Para declarar un objeto de una clase determinada, como atributo de otra clase, es necesario especificar que existe una relación de composición entre ambas clases mediante el modificador object. ¿Verdadero o Falso?

- ☐ Verdadero
☐ Falso

Ejercicio resuelto

Intenta describir los siguientes los métodos de la clase **Rectangulo** teniendo en cuenta ahora su nueva estructura de atributos (dos objetos de la clase **Punto**, en lugar de cuatro elementos de tipo **double**):

1. Método `calcularSuperficie`, que calcula y devuelve el área de la superficie encerrada por la figura.
2. Método `calcularPerimetro`, que calcula y devuelve la longitud del perímetro de la figura.

Solución:

En ambos casos la `interfaz` no se ve modificada en absoluto (desde fuera su funcionamiento es el mismo), pero internamente deberás tener en cuenta que ya no existen los atributos `x1`, `y1`, `x2`, `y2`, de tipo `double`, sino los atributos `vertice1` y `vertice2` de tipo `Punto`.

```
public double calcularSuperficie () {
```

```
    double area, base, altura;    // Variables locales
```

```
    base= vertice2.obtenerX () - vertice1.obtenerX ();    // Antes era x2 - x1
```

```
    altura= vertice2.obtenerY () - vertice1.obtenerY ();    // Antes era y2 - y1
```

```
    area= base * altura;
```

```
    return area;
```

```
}
```

```
public double CalcularPerimetro () {
```

```
    double perimetro, base, altura;    // Variables locales
```

```
    base= vertice2.obtenerX () - vertice1.obtenerX ();    // Antes era x2 - x1
```

```
    altura= vertice2.obtenerY () - vertice1.obtenerY ();    // Antes era y2 - y1
```

```
    perimetro= 2*base + 2*altura;
```

```
    return perimetro;
```

```
}
```