Modificadores de acceso

Se pueden establecer distintos niveles de encapsulación para los miembros de una clase (atributos y operaciones) en función de desde dónde queremos que se pueda acceder a ellos:

Visibilidad	Significado	Java	UML
Pública	Se puede acceder al miembro de la clase desde cualquier lugar.	`public`	`+`
Protegida	Sólo se puede acceder al miembro de la clase desde la propia clase o desde una clase que herede de ella.	`protected`	`#`
Por defecto	**Se puede acceder a los miembros de una clase desde cualquier clase en el mismo paquete**		`~`
Privada	Sólo se puede acceder al miembro de la clase desde la propia clase.	`private`	`-`

La encapsulación
permite agrupar datos y operaciones en un objeto,
de tal forma los detalles del objeto se ocultan a sus usuarios
(ocultamiento de información):

A un objeto se accede a través de sus métodos públicos (su **interfaz**), por lo que no es necesario conocer su **implementación**.

Para encapsular el estado de un objeto,
sus atributos se declaran como variables de instancia privadas.

```
package economics;

public class Costes
{
  private double costeInicial;
  private double costeMarginal;
  …
}
```

Como consecuencia, se han de emplear métodos `get` para permitir que se pueda acceder al estado de un objeto:

```
public class Costes…
  public double getCosteInicial ()
  {
    return costeInicial;
  }
  public double getCosteMarginal ()
  {
    return costeMarginal;
  }
```

Si queremos permitir que se pueda modificar el estado de un objeto desde el exterior, implementaremos métodos `set`:

```
public class Costes…
  public void setCosteInicial (double inicial)
  {
    this.costeInicial = inicial;
  }
  public void setCosteMarginal (double marginal)
  {
    this.costeMarginal = marginal;
  }
```

OBSERVACIONES FINALES:

- Que los miembros de una clase sean privados quiere decir que no se puede acceder a ellos desde el exterior de la clase (ni siquiera desde sus propias subclases), lo que permite mantener la encapsulación de los objetos.

- La visibilidad protegida relaja esta restricción ya que permite acceder a los miembros de una clase desde sus subclases.

  No obstante, su uso tiende a crear jerarquías de clases fuertemente acopladas, algo que procuraremos evitar.

```
public class Figura
{
  protected double x;
  protected double y;
  protected Color  color;
  protected Panel  panel;
  …
}

public class Cuadrado extends Figura
{
  …
  // Desde cualquier sitio de la implementación
  // de la clase Cuadrado se puede acceder a los
  // miembros protegidos de la clase Figura
  …
}
```

  p.ej. Si tuviésemos que localizar un error que afectase al color de la figura no bastaría con examinar el código de la clase `Figura`. También tendríamos que analizar el uso que se hace del atributo color en todas las subclases de `Figura`.