Encapsulamiento

Encapsulamiento

Dijimos antes que la programación orientada a objetos trata de **emular los objetos** del mundo real.

Ejemplo:

El **control remoto** es un elemento con el que interactuamos habitualmente. Al abrirlo y observar su “estado interno”, se ven una serie de conectores, placas, integrados. No es algo fácil de entender, a simple vista, para alguien que no sea electrónico.

Existe algo que se ubica entre nosotros y ese “estado interno” que **nos abstrae de la complejidad subyacente que tiene ese objeto: la *interfaz.***

"El principio de encapsulamiento es **ocultar un proceso o algún conjunto de datos de modo que sea inaccesible para el mundo exterior, excepto a través de la interfaz** que se proporciona para hacer uso adecuado de ellos".

Alan Kay

En otras palabras, es el mecanismo que nos permite **separar la representación interna de un objeto respecto a su representación externa,** o su interfaz.

El concepto de encapsulamiento es un concepto fundamental a la hora de diseñar **sistemas informáticos complejos.**

Conceptos clave

Representación interna

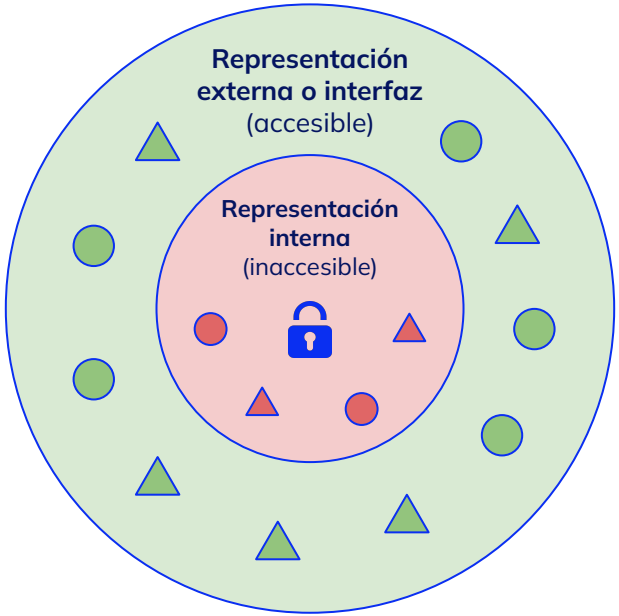
Se refiere a los **datos miembro** (atributos/propiedades) y **métodos privados de la clase.**

Representación externa o interfaz

Son los **métodos públicos** que permiten acceder y modificar el estado interno del objeto de manera controlada.

Esto promueve el **principio de abstracción** al ocultar la complejidad interna y proporcionar una interfaz simple para el uso del objeto.

Representación:



Niveles de visibilidad

Para aplicar este concepto necesitamos un mecanismo que permita **ocultar la implementación interna** (estado interno y detalles de funcionamiento) de un objeto y sólo exponer una interfaz externa (métodos públicos).

Esto se logra, por lo general, con **niveles de visibilidad** o convenciones del lenguaje:

+	**Público** *(public)*	▪ Accesible desde cualquier parte del programa, incluyendo otras clases y paquetes. ▪ No hay restricciones en su acceso.
#	**Protegido** *(protected)*	▪ Accesible desde la misma clase, clases derivadas (subclases) y clases dentro del mismo paquete. ▪ No es accesible desde clases fuera del paquete si no son subclases.
-	**Privado** *(private)*	▪ Accesible sólo desde la misma clase. ▪ No es accesible ni siquiera por clases derivadas (subclases) fuera de la clase que lo define.

Métodos de acceso y modificación

	Getter	*Setter*
Características	▪ Método que devuelve el valor de un atributo privado. ▪ Permite leer el estado de un objeto sin modificarlo directamente.	▪ Método que asigna un valor a un atributo privado. ▪ Permite modificar el estado de un objeto de acuerdo a ciertas validaciones.
Ejemplo	`public TipoDato getAtributo() { return this.atributo; }`	`public void setAtributo(TipoDato valor) { this.atributo = valor; }`

Ventajas de los *Getters* y *Setters*

- **Encapsulamiento:** Controla el acceso a los atributos. Mantiene el principio de encapsulación.
- **Flexibilidad:** Permite establecer validaciones y lógica de negocio al acceder o modificar atributos.
- **Seguridad:** Evita el acceso directo a atributos privados desde fuera de la clase.