ENCAPSULAMIENTO VS. OCULTAMIENTO DE INFORMACIÓN

CESSI #ArgentinaPrograma #YoProgramo

Leonardo Blautzik - Federico Gasior - Lucas Videla

Agosto -Diciembre de 2021

ENCAPSULR

Definición:

Encapsular: Acción de poner juntas ciertas cosas dado que hay una razón para ello.

En la POO aquellas cosas serán los **datos** y los **métodos** que operan sobre esos datos.

Mediante el **encapsulamiento** es que creamos las entidades que deseamos manejar en nuestros sistemas.

Tiene dos sentidos:

- **Especialización:** El propio objeto es quien sabrá cómo manejar los datos que contiene, y lo hará con los métodos asociados a sus dados.
- **Completitud**: Nos permite descansar en que la abstracción construida representa a la entidad, y a aquella **responsabilidad** que tendrá asignada dentro de nuestro sistema.

INTERFAZ PUBLICA

Definición:

Se define como **interfaz pública** de una clase, al conjunto de responsabilidades que los objetos de esa clase exponen o brindan al exterior de la misma, al universo.

Nos interesa mucho más **la interfaz pública de los objetos** que sus **representaciones internas** y datos asociados.

OCULTAMIENTO DE INFORMACIÓN

Definición:

Ocultar la información es utilizar las técnicas que nos brinda el lenguaje para abstraer a nuestros módulos cliente de los detalles de implementación.

Nos indica **buenas prácticas de programación:** “necesito de este objeto, pero no me importa cómo resuelva sus **responsabilidades** mientras lo haga por mí”.

Reglas

Encapsulamiento	Ocultamiento de información
1. Ubicar los datos y las operaciones que trabajan sobre esos datos en la misma clase. 2. Utilizar diseño guiado por las responsabilidades para determinar la agrupación de datos y operaciones dentro de clases.	1. No exponer atributos 2. No exponer diferencia entre atributos propiamente dichos y atributos calculados. 3. No exponer la estructura interna de una clase. 4. No exponer detalles de implementación de una clase

Getters y Setters

Accesores

Bajo ciertas circunstancias puede ser necesario cambiar algún valor interno de un objeto. Es importante proporcionar un mecanismo para poder hacerlo sin romper el encapsulamiento, o sea, sin hacer públicos algunos -o todos- sus miembros.

Los **accesores** serán los que nos permitan acceder a esos miembros privados de una manera controlada.

- **Getters (obtener):** sirven para obtener el valor de un atributo.
- **Setters (establecer):** sirven para establecer el valor de un atributo.

Getters y Setters

Típicamente, un getter tiene esta estructura:

```
public Integer getEdad() {
  return this.edad;
}
```

Un setter, en cambio, responde a la siguiente estructura:

```
public void setEdad(Integer edad) {
  this.edad = edad;
}
```

Al definir una clase, se define la complejidad de éstos métodos y qué tan directamente se permite el acceso a los miembros de la misma.

Un mal ejemplo de ocultamiento:

Suponga que se tiene una clase Fecha que incluye los atributos dia, mes, anio. Una mala implementación permite el acceso directo a los atributos de los datos, por ejemplo:

```
public class Fecha {
    public int dia;
    public int mes;
    public int anio;
}
```

Como puede verse, no es una buena implementación, ya que el cliente accede a los atributos y puede cometer errores.

Un mal ejemplo de ocultamiento:

```
Fecha unaFecha = new Fecha();

unaFecha.dia = 32; // día inválido
unaFecha.mes = 2;
uanFecha.dia = 30; // mal
unaFecha.dia += 1;
//no se realiza la validación para pasar al siguiente día.
```

No se ejerce ningún control sobre los accesos a los atributos.

Un mejor ejemlo de ocultamiento

Utilizamos el modificador de acceso `private` para ocultar los atributos.

```
public class Fecha {
    private int dia;
    private int mes;
    private int anio;
```

Un mejor ejemlo de ocultamiento

Proveemos los `setters` y `getters` adecuados.

```
    public boolean setDia(int d) {...};
    public boolean setMes(int m) {...};
    public boolean setAnio(int a) {...};

    public int getDia() {...};
    public int getMes() {...};
    public int getAnio() {...};
}
```

Usamos la class Fecha

```
Fecha fecha = new Fecha();
fecha.setDia(32);  // devolverá false.
fecha.setMes(2);
fecha.setDia(30); //devolverá falso.
fecha.setDia(fecha.getDia() + 1);
/* devolverá falso o  incrementará el día y el mes en caso
de que el día ya se encuentre en el valor máximo para ese mes.
*/
```

Ahora cambiamos la implementación interna

Reemplazamos los atributos por la cantidad de días que transcurren desde el comienzo de una época. Dado que el programador encapsuló los atributos detrás de una interfaz pública, **estos cambios no afectan el código del cliente.**

```
public class Fecha{
    private long fecha;

    public boolean setDia(int d) {...};
    public boolean setMes(int m) {...};
    public boolean setAnio(int a) {...};
    public int getDia() {...};
    public int getMes() {...};
    public int getAnio() {...};
}
```

CONSTRUCTORES

Definición:

Un constructor es una subrutina cuyo propósito es inicializar el estado de un objeto

Declaración y definición

```
class <nombre de la clase> {
    <nombre de la clase>(<parametros>) {
        sentencia 1;
        sentencia 2;
        ...
        sentencia n;
    }
}
```

CONSTRUCTORES VS. MÉTODOS

Los constructores, a diferencia de los métodos:

- Se ejecutan automáticamente al momento de instanciar un objeto.
- Son rutinas de inicialización, por lo tanto no devuelven valores ni pueden ser invocados.

MODIFICADORES DE ACCESO

Permiten limitar el acceso a los miembros de una clase Los modificadores de acceso preceden a la declaración de un miembro de la clase

Modificador	Visibilidad
public	desde cualquier otra clase
(no se indica nada)	solo dentro del paquete
protected	dentro de la nisma clase o de sus clases derivadas
private	solo dentro de la misma clase

INTERFAZ

Lo que otras clases ven de una clase:

- Declaración de los constructores públicos.
- Declaración de los métodos públicos.

La class Nota

Implementar la clase Nota para cumplir con la siguiente interfaz:

```
class Nota {
    /**
     * pre : valorInicial esta comprendido entre 0 y 10.
     * post: inicializa la Nota con el valor indicado.
     */
    public Nota(int valorInicial) { }
    /**
     * post: devuelve el valor numerico de la Nota,
     *       comprendido entre 0 y 10.
     */
    public int obtenerValor() { }
```

```
    /**
     * post: indica si la Nota permite o no la aprobacion.
     */
    public boolean aprobado() { }
    /**
     * post: indica si la Nota implica desaprobacion.
     */
    public boolean desaprobado() { }
    /**
     * pre : nuevoValor esta comprendido entre 0 y 10.
     * post: modifica el valor numerico de la Nota, cambiandolo
     *       por nuevoValor, siempre y cuando nuevoValor sea superior al
     *       valor numerico actual de la Nota.
     */
    public void recuperar(int nuevoValor) { }
}
```

PAUSA, cafecito y..

Vamos a eclipse...