Semana 07

Encapsulación en Java

La encapsulación es un concepto fundamental en la programación orientada a objetos (POO) que se refiere a la agrupación de datos y métodos que operan con esos datos dentro de una sola unidad, que se denomina clase en Java. La encapsulación es una forma de ocultar los detalles de implementación de una clase del acceso externo y solo exponer una interfaz pública que se puede usar para interactuar con la clase.

En Java, la encapsulación se logra declarando las variables de instancia de una clase como privadas, lo que significa que solo se puede acceder a ellas dentro de la clase. Para permitir el acceso externo a las variables de instancia, se definen métodos públicos llamados getters y setters, que se utilizan para recuperar y modificar los valores de las variables de instancia, respectivamente. Mediante el uso de getters y setters, la clase puede aplicar sus propias reglas de validación de datos y garantizar que su estado interno se mantenga constante.

Ejemplo

```
public class Person {
	private String name;
	private int age;

	public String getName() { return name; }

	public void setName(String name) { this.name = name; }

	public int getAge() { return age; }

	public void setAge(int age) { this.age = age; }
}

public class Main {
	public static void main(String[] args)
	{
		Person person = new Person();
		person.setName("John");
		person.setAge(30);
```

```
                System.out.println("Name: " + person.getName());
                System.out.println("Age: " + person.getAge());
        }
}
```

Definición

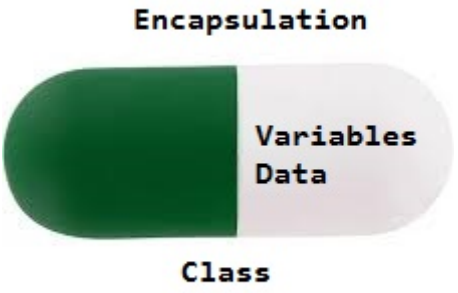
La **encapsulación** se define como el empaquetado de datos en una sola unidad. Es el mecanismo que une el código y los datos que manipula. Otra forma de pensar en la encapsulación es que es un escudo protector que evita que el código acceda a los datos fuera de este escudo.

- Técnicamente, en la encapsulación, las variables o los datos de una clase se ocultan de cualquier otra clase y solo se puede acceder a ellos a través de cualquier función miembro de su propia clase en la que se declare.
- Al igual que en la encapsulación, los datos de una clase se ocultan de otras clases utilizando el concepto de ocultación de datos que se logra al hacer que los miembros o métodos de una clase sean privados, y la clase se expone al usuario final o al mundo sin proporcionar ningún detalle. detrás de la implementación utilizando el concepto de abstracción, por lo que también se conoce como una **combinación de ocultación de datos y abstracción**.
- La encapsulación se puede lograr declarando todas las variables de la clase como privadas y escribiendo métodos públicos en la clase para establecer y obtener los valores de las variables.
- Está más definido con el método setter y getter.

Ventajas de la encapsulación

- **Ocultación de datos:** es una forma de restringir el acceso de nuestros miembros de datos ocultando los detalles de implementación. La encapsulación también proporciona una forma de ocultar datos. El usuario no tendrá idea de la implementación interna de la clase. No será visible para el usuario cómo la clase almacena valores en las variables. El usuario solo sabrá que estamos pasando los valores a un método de establecimiento y que las variables se inicializan con ese valor.
- **Mayor flexibilidad:** podemos hacer que las variables de la clase sean de solo lectura o de solo escritura según nuestros requisitos. Si deseamos que las variables sean de solo lectura, debemos omitir los métodos de establecimiento como setName(), setAge(), etc. del programa anterior o si deseamos que las variables sean de solo escritura, debemos omitir el obtenga métodos como getName(), getAge(), etc. del programa anterior.

- **Reutilización:** la encapsulación también mejora la reutilización y es fácil de cambiar con nuevos requisitos.
- **Probar el código es fácil:** el código encapsulado es fácil de probar para las pruebas unitarias.



Ejemplos

```
// fields to calculate area
class Area {

	int length;
	int breadth;

	// constructor to initialize values
	Area(int length, int breadth)
	{
		this.length = length;
		this.breadth = breadth;
	}

	// method to calculate area
	public void getArea()
	{
		int area = length * breadth;
		System.out.println("Area: " + area);
	}
}

class Main {
	public static void main(String[] args)
	{
```

```
        Area rectangle = new Area(2, 16);
        rectangle.getArea();
    }
}
```

El programa para acceder a variables de la clase Encapsulate se muestra a continuación:

```
// Java program to demonstrate encapsulation

class Encapsulate {
    // private variables declared
    // these can only be accessed by
    // public methods of class
    private String geekName;
    private int geekRoll;
    private int geekAge;

    // get method for age to access
    // private variable geekAge
    public int getAge() { return geekAge; }

    // get method for name to access
    // private variable geekName
    public String getName() { return geekName; }

    // get method for roll to access
    // private variable geekRoll
    public int getRoll() { return geekRoll; }

    // set method for age to access
    // private variable geekage
    public void setAge(int newAge) { geekAge = newAge; }

    // set method for name to access
    // private variable geekName
    public void setName(String newName)
    {
        geekName = newName;
    }

    // set method for roll to access
    // private variable geekRoll
    public void setRoll(int newRoll) { geekRoll = newRoll; }
```

```
}

public class TestEncapsulation {
	public static void main(String[] args)
	{
		Encapsulate obj = new Encapsulate();

		// setting values of the variables
		obj.setName("Harsh");
		obj.setAge(19);
		obj.setRoll(51);

		// Displaying values of the variables
		System.out.println("Geek's name: " + obj.getName());
		System.out.println("Geek's age: " + obj.getAge());
		System.out.println("Geek's roll: " + obj.getRoll());

		// Direct access of geekRoll is not possible
		// due to encapsulation
		// System.out.println("Geek's roll: " +
		// obj.geekName);
	}
}
```

En el programa anterior, la clase Encapsular se encapsula ya que las variables se declaran privadas. Los métodos get como getAge(), getName() y getRoll() se establecen como públicos, estos métodos se utilizan para acceder a estas variables. Los métodos setter como setName(), setAge(), setRoll() también se declaran como públicos y se utilizan para establecer los valores de las variables.

```
class Name {

	private int age; // Private is using to hide the data

	public int getAge() { return age; } // getter

	public void setAge(int age)
	{
		this.age = age;
	} // setter
}

class GFG {
	public static void main(String[] args)
```

```
	{

		Name n1 = new Name();

		n1.setAge(19);

		System.out.println("The age of the person is: "
					+ n1.getAge());
	}
}

class Account {
//private data members to hide the data
private long acc_no;
private String name,email;
private float amount;
//public getter and setter methods
public long getAcc_no() {
	return acc_no;
}
public void setAcc_no(long acc_no) {
	this.acc_no = acc_no;
}
public String getName() {
	return name;
}
public void setName(String name) {
	this.name = name;
}
public String getEmail() {
	return email;
}
public void setEmail(String email) {
	this.email = email;
}
public float getAmount() {
	return amount;
}
public void setAmount(float amount) {
	this.amount = amount;
}

}
public class GFG {
```

```
public static void main(String[] args) {
        //creating instance of Account class
        Account acc=new Account();
        //setting values through setter methods
        acc.setAcc_no(7310805450L);
        acc.setName("MD FAIZ");
        acc.setEmail("mdfaiz689@gmail.com");
        acc.setAmount(100000f);
        //getting values through getter methods
        System.out.println(acc.getAcc_no()+" "+acc.getName()+" "+acc.getEmail()+"
"+acc.getAmount());
}
}
```

Ventajas y desventajas

Ventajas

1. Mejora la seguridad del estado interno de un objeto ocultándolo del mundo exterior.
2. Aumenta la modularidad y la capacidad de mantenimiento al facilitar el cambio de implementación sin afectar otras partes del código.
3. Habilita la abstracción de datos, lo que permite que los objetos se traten como una sola unidad.
4. Permite agregar fácilmente nuevos métodos y campos sin afectar el código existente.
5. Admite el principio de ocultación de información orientado a objetos, lo que facilita cambiar la implementación sin afectar el resto del código.

Desventajas

1. Puede conducir a una mayor complejidad, especialmente si no se usa correctamente.
2. Puede hacer que sea más difícil entender cómo funciona el sistema.
3. Puede limitar la flexibilidad de la implementación.