Práctica 5. Modificadores de acceso y encapsulamiento

Modificadores de acceso

En Java, existen ciertas palabras clave que indican el nivel de acceso de una clase, atributo o método. Por "acceso", estamos hablando de herencia, la creación de objetos en el caso de clases, lectura y modificación en el caso de atributos y métodos de llamada. El nivel de acceso se refiere a la visibilidad de estos miembros, en otras palabras, mediante el uso de modificadores de acceso podemos controlar qué clases pueden acceder a qué atributos o métodos. Esta es una herramienta importante para mantener nuestro software bajo control y es parte de los conceptos fundamentales detrás de la Programación Orientada a Objetos. Al definir estas "reglas", podemos prevenir comportamientos no deseados y evitar depuraciones innecesarias.

Notas:

Los modificadores de acceso se pueden asignar a clases (acceso de nivel superior) y sus miembros (acceso a nivel de miembro, para atributos y métodos).

En el nivel superior, generalmente definimos las clases como públicas o privadas de paquetes (predeterminado).

El **acceso público** significa que todo el mundo podrá acceder a la clase o miembro en cuestión. No habrá restricciones, cualquier código que pueda leer el objeto o miembro podrá modificarlo según lo necesite.

```
public class MyClass{
```

El modificador de acceso predeterminado (*default access*) indica que solo las clases del mismo paquete pueden acceder a la clase o miembro. Cuando no se asigna ningún modificador de acceso a una clase o miembro, se define automáticamente como predeterminado.

```
String name;
int number;
```

Es relativamente fácil trabajar con estos dos modificadores, ya que no nos restringen demasiado. Pero es una buena práctica de programación escribir software que solo tenga acceso a los recursos necesarios para su propio propósito. Esto se conoce como el principio de privilegio mínimo, una práctica que nos ayudará a crear programas que sean más seguros y fáciles de mantener. En Java, generalmente aplicamos este principio en el acceso a nivel de miembro.

El modificador de **acceso privado** significa que solo se podrá acceder a un atributo o método desde la misma clase en la que está declarado. Si se intenta acceder al miembro desde cualquier otra clase, se producirá un error de compilación.

```
private String model;
private int mileage;
private double speed;
```

El modificador de acceso protegido (*protected access*) indica que solo se puede acceder al miembro desde clases en el mismo paquete o subclases del que está declarado.

```
protected String title;
protected String author;
```

La siguiente es una tabla que muestra diferentes modificadores y sus niveles de acceso:

Modifier	Class	Package	Subclass	World
public	✓	✓	✓	✓
protected	✓	✓	✓	✗
default	✓	✓	✗	✗
private	✓	✗	✗	✗

Ejercicio 1:
Agregar un modificador de acceso privado a una declaración de clase e intente crear una instancia de un objeto.

¿Funciona?

¿En qué casos sería necesario declarar una clase privada?

Encapsulamiento:
Tener control sobre quién puede acceder a los miembros de una clase es genial, pero ¿cómo trabajamos realmente con los atributos de una clase si se declaran privados?
Aquí es donde aplicamos el concepto POO de encapsulación en forma de métodos definidos para acceder a dichos atributos.

Para acceder a un atributo de clase privada, podemos definir un método especial conocido como "**getter**". Este método será declarado público, tendrá el mismo tipo de retorno que el atributo que queremos "obtener", no tomará ningún parámetro y devolverá dicho atributo. De esta forma, aunque el atributo sea privado, tenemos una forma de acceder a él sin arriesgar la posibilidad de que algún código desconocido lo modifique sin permiso.

Si necesitamos modificar el contenido de un objeto y se declara privado, podemos hacerlo declarando otro tipo especial de método conocido como "**setter**". Al igual que con el getter, se declarará público, pero esta vez tendrá un tipo de retorno de void y tomará un parámetro del mismo tipo que el atributo a modificar. Cuando llamemos a este método setter, le pasaremos el valor que queremos asignar al atributo. Dentro del cuerpo del método, podemos declarar la asignación.

```
public void setDenominator(double denominator){
	if(denominator != 0){
		this.denominator = denominator;
	}
}
```

El siguiente ejemplo es de una clase con un solo atributo y sus correspondientes métodos getter y setter:

```
public void Character(){
	private int healthPoints;

	public int getHealthPoints(){
		return healthPoints;
```

```
	}
	public void setHealthPoints(int healthPoints){
		this.healthPoints = healthPoints;
	}
}
```

Práctica:

1. Utilizar los conceptos obtenidos en esta práctica para crear un programa que simule operaciones bancarias.
2. Haga una clase banco que pueda registrar nuevas cuentas.
3. La clase “Cuenta”, a su vez, tendrá diferentes atributos que, por la naturaleza de esta información, deberán permanecer todos privados y solo ser accesibles para el “Banco”.
4. Una "Cuenta" se compone del "nombre" del titular de la cuenta, el "saldo" que tiene esta cuenta y un "PIN" que se utiliza para acceder al saldo de la cuenta.
5. Crear un menú en el que el usuario pueda crear una cuenta o acceder a una ya existente.
6. Al crear una cuenta se ingresará la información necesaria por parte del usuario para realizar el primer depósito.
7. Para una cuenta ya existente, primero debe pedirle al usuario su PIN y luego dejar que retire o deposite como mejor le parezca.
8. Recuerda validar las transacciones para que tu banco no pierda ninguno de sus recursos.

Actividades:

- Hacer un reporte con portada de hoja completa que incluya el código fuente, el link de su código en GitHub y una captura de pantalla de la práctica funcionando.
- En caso de contener menús será necesario agregar en el reporte una captura de pantalla por cada opción seleccionada del menú.
- Subir a blackboard código fuente (archivos java) y reporte como archivos separados, no se aceptarán archivos en carpetas comprimidas.