

**GUÍA DE LABORATORIO 03**
Modificadores y Encapsulación

Asignatura	Datos del alumno	Fecha y Firma
Algoritmos y solución de problemas	Apellidos:	
	Nombre:	

Instrucciones:

Desarrollar las actividades que indica el docente en base a la guía de trabajo que se presenta.

1. Objetivos:

- Aprender sobre modificadores
- Aprender sobre encapsulación.

2. Equipos, Herramientas o Materiales

- Computador
- Software: JDK – VSCode - Extension Pack for Java.

3. Fundamento Teórico**3.1. Modificadores de Java**

La palabra clave **public** es un modificador de acceso, lo que significa que se utiliza para establecer el nivel de acceso para clases, atributos, métodos y constructores.

Dividimos los modificadores en dos grupos:

- Modificadores de acceso: controlan el nivel de acceso
- Modificadores sin acceso: no controlan el nivel de acceso, pero proporcionan otras funciones.

Modificadores de acceso

Modificador	Descripción
Public	La clase es accesible por cualquier otra clase
default	La clase solo es accesible por clases en el mismo paquete. Esto se usa cuando no especificas un modificador. Aprenderás más sobre paquetes en el capítulo de Paquetes

Para atributos, métodos y constructores, puede utilizar uno de los siguientes:

Modificador	Descripción
Public	El código es accesible para todas las clases.
private	El código solo es accesible dentro de la clase declarada.
default	El código solo es accesible en el mismo paquete.
protected	El código es accesible en el mismo paquete y en subclases.



Modificadores sin acceso

Para las clases, puedes utilizar uno de los siguientes:

Modificador	Descripción
Final	Esta palabra clave indica que la clase no puede ser heredada por otras clases.
abstract	Esta palabra clave significa que la clase no puede ser utilizada para crear objetos directamente

Para los atributos y métodos, puede utilizar uno de los siguientes:

Modificador	Descripción
Final	Indica que los atributos y métodos no pueden ser sobrescritos o modificados en las clases derivadas (subclases)
static	Los atributos y métodos estáticos pertenecen a la clase en sí, no a instancias específicas (objetos) de la clase. Se puede acceder a ellos sin crear un objeto de la clase
abstract	Solo se puede usar en una clase abstracta. Solo se puede aplicar a métodos. El método no tiene un cuerpo (implementación), solo una declaración. La subclase que hereda de la clase abstracta debe proporcionar la implementación del método.
Transient	Los atributos y métodos marcados como transient se omiten cuando se serializa el objeto (se convierte en una secuencia de bytes para almacenamiento o transmisión)
synchronized	Los métodos sincronizados solo pueden ser accedidos por un hilo (subproceso de ejecución) a la vez. Esto evita problemas de concurrencia cuando varios hilos intentan acceder al mismo método simultáneamente
Volatile	El valor de un atributo volátil no se almacena en caché a nivel de hilo y siempre se lee desde la "memoria principal". Esto garantiza que todos los hilos vean el valor más actualizado del atributo

Final

Si no desea tener la capacidad de anular los valores de atributos existentes, declare los atributos como final:

```
Main.java 5
Semana02 > Main.java > ...
1  public class Main {
2      final int x = 10;
3      final double PI = 3.14;
4
5      Run | Debug | Run main | Debug main
6      public static void main(String[] args) {
7          Main myObj = new Main();
8          myObj.x = 50; // will generate an error: cannot assign a value to a final variable
9          myObj.PI = 25; // will generate an error: cannot assign a value to a final variable
10         System.out.println(myObj.x);
11     }
12 }
```

Static

Un método static significa que se puede acceder a él sin crear un objeto de la clase, a diferencia de public:



```
Main.java
Semana02 > Main.java > Language Support for Java(TM) by Red Hat > Main
1 public class Main {
2     // Static method
3     static void myStaticMethod() {
4         System.out.println(x:"Static methods can be called without creating objects");
5     }
6
7     // Public method
8     public void myPublicMethod() {
9         System.out.println(x:"Public methods must be called by creating objects");
10    }
11
12    // Main method
13    Run | Debug | Run main | Debug main
14    public static void main(String[] args) {
15        myStaticMethod(); // Call the static method
16        // myPublicMethod(); This would output an error
17
18        Main myObj = new Main(); // Create an object of Main
19        myObj.myPublicMethod(); // Call the public method
20    }
```

Abstract

Un método abstract pertenece a una clase abstract y no tiene un cuerpo. El cuerpo lo proporciona la subclase:



```
Main.java 1
Semana02 > Main.java > Language Support for Java(TM) by Red Hat > Second
1 // Code from filename: Main.java
2 // abstract class
3 abstract class Main {
4     public String fname = "John";
5     public int age = 24;
6     public abstract void study(); // abstract method
7 }
8
9 // Subclass (inherit from Main)
10 class Student extends Main {
11     public int graduationYear = 2018;
12     public void study() { // the body of the abstract method is provided here
13         System.out.println(x:"Studying all day long");
14     }
15 }
16 // End code from filename: Main.java
17
18 // Code from filename: Second.java
19 class Second {
20     Run | Debug | Run main | Debug main
21     public static void main(String[] args) {
22         // create an object of the Student class (which inherits attributes and methods from Main)
23         Student myObj = new Student();
24
25         System.out.println("Name: " + myObj.fname);
26         System.out.println("Age: " + myObj.age);
27         System.out.println("Graduation Year: " + myObj.graduationYear);
28         myObj.study(); // call abstract method
29     }
```



3.2. Encapsulación de java

Encapsulación

El significado de la encapsulación es asegurarse de que los datos "sensibles" estén ocultos a los usuarios. Para lograrlo, debe:

-  declarar variables/atributos de clase como **private**
-  Proporcionar métodos públicos de obtención y configuración para acceder y actualizar el valor de una variable **private**.

Obtener y configurar

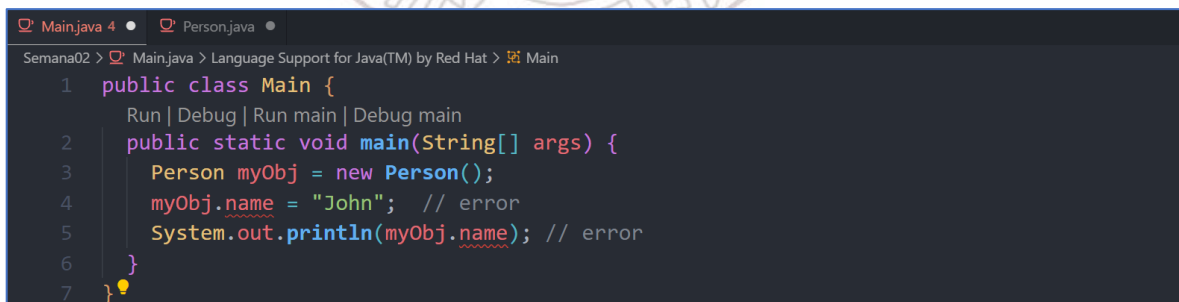
Sabemos que **private** solo se puede acceder a las variables dentro de la misma clase (una clase externa no tiene acceso a ellas). Sin embargo, es posible acceder a ellas si proporcionamos métodos públicos **get** y **set**.

El método **get** devuelve el valor de la variable y el método **set** establece el valor.

La sintaxis para ambos es que comienzan con **get** o **set**, seguido del nombre de la variable, con la primera letra en mayúscula:



Sin embargo, como la variable **name** se declara como **private**, **no podemos** acceder a ella desde fuera de esta clase:



Si la variable se declaró como **public**, esperaríamos el siguiente resultado:





Sin embargo, cuando intentamos acceder a una variable **private**, obtenemos un error:

En su lugar, utilizamos los métodos **getName()** y **setName()** para acceder y actualizar la variable:



```
1 public class Main {
2     public static void main(String[] args) {
3         Person myObj = new Person();
4         myObj.setName(newName:"John"); // Set the value of the name variable to "John"
5         System.out.println(myObj.getName());
6     }
7 }
```

¿Por qué encapsulación?

-  Mejor control de los atributos y métodos de clase
-  Los atributos de clase se pueden hacer de solo lectura (si solo usa el getmétodo) o de solo escritura (si solo usa el setmétodo)
-  Flexible: el programador puede cambiar una parte del código sin afectar otras partes.
-  Mayor seguridad de los datos.

4. Ejercicios

Ejercicio parte 01:

- 1) Diseña una clase Persona con los atributos nombre, edad y correo Electronico. Implementa la encapsulación para proteger estos atributos y proporciona métodos get y set para acceder y modificarlos de forma controlada.
- 2) Crea una clase CuentaBancaria con los atributos numeroCuenta (público), saldo (privado) y titular (protegido). Implementa métodos para depositar y retirar dinero, asegurando que el saldo no se vuelva negativo.
- 3) Diseña una clase Circulo con un atributo radio que sea final. Implementa un método para calcular el área del círculo.
- 4) Crea una clase Utilidades con un método static llamado convertirCelsiusAFahrenheit que tome una temperatura en grados Celsius y la convierta a Fahrenheit.