

# MANUAL JAVA INTERMEDIO



## Contenido

1)	Programa oración invertida .....	4
2)	Palabras reservadas en java .....	4
3)	Comentarios en java.....	4
4)	Programacion orientada a objetos(POO) .....	5
5)	Primer programa POO .....	6
6)	Palabra reservada This .....	7
7)	Encapsulamiento en java (POO) .....	11
8)	Ejercicio de encapsulamiento .....	12
9)	Métodos Setter y Getter en Java (POO) .....	16
10)	Modificadores de acceso en Java .....	18
11)	Herencia o reutilización de código .....	19
12)	Ejemplo de herencia.....	20
13)	Polimorfismo .....	22
14)	Ejemplo polimorfismo .....	22
15)	Palabra reservada Super .....	24
16)	Ejemplo uso reservado de palabra Super .....	24
17)	Ámbito de variables en java .....	25
18)	Ejemplo ámbito de las variables .....	26
19)	Excepciones en java (try-catch-finally).....	27
20)	Ejemplo excepciones .....	27
21)	Constructores en java.....	29
22)	Ejemplo constructores en java (programa sin constructor).....	29
23)	Ejemplo constructores en java (programa con constructor) .....	30
24)	Hilos (Threads) .....	31
25)	Ejercicios hilos (threads ).....	32
26)	Hilos con parámetros .....	34
27)	Estado de los hilos (new).....	36
28)	Estado de los hilos (runnable) .....	36
29)	Estado de los hilos (Blocked) método sleep.....	37
30)	Estado de los hilos (Blocked) método sleep aplicado a la clase hija en bucle for.....	39

31)	Estado de los hilos (dead / finalizado) método stop o interrupt .....	40
32)	Sincronización de hilos .....	42
33)	Sincronización de hilos ejemplo .....	45
34)	Clase Math: Numero e y pi .....	47
35)	Clase Math: conversión de grados a radianes y viceversa .....	48
36)	Clase Math: funciones trigonométricas básicas .....	49
37)	Clase Math: Raiz cuadrada y controlando número de decimales .....	50
38)	Clase Math: números aleatorios .....	51
39)	Suma y resta de matrices .....	53
40)	Multiplicación de matrices .....	54
41)	Ejercicio multiplicación de matrices .....	55

## 1) Programa oración invertida

```
import java.util.Scanner;

public class EjemploUno {
    public static void main (String arg []) {
        Scanner entra = new Scanner(System.in);
        String nombre = "", nombreInvertido="";
        int numero=0;
        System.out.println("Escribe tu oracion:");
        nombre = entra.nextLine();
        numero= nombre.length();
        System.out.println("La oracion tiene "+numero+" caracteres");

        for(int i = numero ; i!=0 ;i=i-1 ){
            nombreInvertido =nombre.substring(i-1,i);
            System.out.print(nombreInvertido);
        }
    }
}
```

## 2) Palabras reservadas en java

Son palabras dentro del lenguaje java como lo son for, do, while, String, etc.

Por lo tanto debemos evitar usarlas como variables.

## 3) Comentarios en java

// comentarios en una linea ya se han utilizado en programas anteriores

```
/* notas
 *multi
 *linea
 */
```

#### 4) Programacion orientada a objetos(POO)

La poo, es una manera de diseñar y desarrollar software que trata de imitar la realidad tomando algunos conpectos esenciales de ella.

El primero de estos es el **objeto** cuyos rasgos son la identidad, el estado y comportameinto. Por ejemplo

Nombre objeto( <b>identificador</b> ):	Chester
Estado o características ( <b>variables</b> )	Raza: chow chow Edad: 5 años Rasgo: peludo color negro

Nombre objeto( <b>identificador</b> ):	Mailo
Estado o características ( <b>variables</b> )	Raza: pastor aleman Edad: 3 años Rasgo: enorme y orejon

Si lo comparamos ambos pueden:

- Ladrar
- Corer
- Jugar
- Comer
- Dormir

Esto es su comportamiento (**metodos**)

Si hiciéramos un programa perro.java, las variables podrían ser, raza, color y edad, y métodos: ladrar (), correr (), jugar (), comer () y dormir ()

## 5) Primer programa POO

Debemos crear 2 clases : SumaMain y suma

### SumaMain

```
import java.util.Scanner;

// creamos 2 clases la clase main donde se ejecutara el programa
principal
// y la segunda clase donde vamos a realizar los metodos

public class SumaMain {
    public static void main(String[] args) {
        Scanner entra=new Scanner(System.in);

        System.out.println("Escribe el primer valor:");
        double numUno = entra.nextDouble();

        System.out.println("Escribe el segundo valor:");
        double numDos = entra.nextDouble();

        Suma valores = new Suma (numUno,numDos); //el objeto puede tener
el nombre que quiera
        //clase_externa nombre_objeto = new clase_externa( variables a
enviar);
        valores.Impri(); //llamamos al objeto y el metodo a realizar
        //objeto.metodo();

    }
}
```

### Suma

```
public class Suma {
    private double vUno, vDos, res; //declaramos variables privadas ya
que solo se usara en esta clase
    public Suma(double numUno, double numDos){ //declaramos un
constructor publico para asignar los valores, debe llamar igual que la
clase, entre paréntesis variables definidas con su tipo de dato
        this.vUno=numUno; //asignamos primer valor con variable
privada
        this.vDos=numDos; //lo mismo con la segunda
    } // en el metodo suma no colocamos void por que no regresara ningun
valor

    public void Operacion(){ // declaramos metodo publico para la
operacion
        res = vUno + vDos;
    } //aquí si colocamos void por que regresara un valor
    public void Impri(){ // declaramos metodo publico para imprimir
        Operacion(); //llamamos el metdo operacion ya que en el esta
la variabl a imprimir
        System.out.println("El resultado de la suma es: "+res); //mensaje

    } //aquí si colocamos void por que regresara un valor
}
```

resultado

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-java
Escribe el primer valor:
5.2
Escribe el segundo valor:
0.8
El resultado de la suma es: 6.0

Process finished with exit code 0
```

## 6) Palabra reservada This

En Poo podemos acceder a variables en otras clases esto se logra con la instancia de clases, esto es cuando hacemos que 2 clases interactúen entre sí, como en el ejercicio anterior donde recibimos valores de la clase Suma que declaramos en un constructor.

Al acceder a variables de instancia de una clase, podemos encontrarlas con variables que se llamen igual que en la clase de donde estamos accediendo, cuando esto sucede utilizamos la palabra reservada this

### Ejemplo 1 area rectangulo

Clase main

```
import java.util.Scanner;

public class areaRectanguloMain {
    public static void main(String[] args) {
        Scanner entra = new Scanner(System.in);
        int base = 0, altura = 0;

        System.out.println("Write base:");
        base = entra.nextInt();

        System.out.println("Write high:");
        altura = entra.nextInt();

        areaRectangulo datos = new areaRectangulo(base, altura);
        datos.imprimir();
    }
}
```

## Clase extra

```
public class areaRectangulo {
    private int base=0, altura=0, res=0;
    public areaRectangulo (int base, int altura){ //constructor mismo
nombre que la clase
        this.base= base; // a pesar de que las variables de la clase para
el objeto datos son las mismas
        this.altura = altura; //con la palabra this podemos indicar que se
tratan de las variables de esta clase
    }

    public void operacion(){
        res= base*altura;
    }

    public void imprimir(){
        operacion();
        System.out.println("El resultado del area es" + res);
    }

}
```

## Resultado

```
Write base:
5
Write high:
17
El resultado del area es85
```

Xd ingles y español



## Ejemplo 2 area circulo

### Clase main

```
import java.util.Scanner;

public class areaCirculoMain {
    public static void main (String []args){
        Scanner ent = new Scanner(System.in);
        double diam=0;

        System.out.println("Write the diameter of circle:");
        diam= ent.nextDouble();

        areaCirculo obj1 = new areaCirculo(diam); // creamos objeto

        obj1.imprmiri(); //del objeto creado llamamos al metodo imprimir

    }
}
```

### Clase extra

```
public class areaCirculo {
    private double diam=0, rad=0, area=0, perimetro=0;

    public areaCirculo(double diam){ // realizamos constructor
        this.diam=diam;
    }

    public void radio(){ //realizamos un metodo para calcular el radio
        // no hay que llamar otro metodo ya que la
        // variable diam pertenece a esta clase
        rad= diam/2;
    }

    public void area(){ //realizamos un metodo para calcular el area
        radio();
        area=3.1416*rad*rad;
    }

    public void perimetro(){ //realizamos un metodo para calcular el
        // perimetro
        perimetro = 3.1416*diam;
    }

    public void imprmiri(){ //realizamos un metodo para imprimir
        radio();
        area();
        perimetro();
        System.out.println("Radius is:"+ rad);
    }
}
```

```
        System.out.println("Area is:"+ area);  
        System.out.println("Perimeter is:"+ perimetro);  
    }  
}
```

### Resultado

```
Write the diameter of circle:  
10  
Radius is:5.0  
Area is:78.54  
Perimeter is:31.416  
  
Process finished with exit code 0
```

## 7) Encapsulamiento en java (POO)

Conceptos previos:

**Atributos** = variables que contiene una clase

**Métodos** = funciones que va a realizar la clase

**Objeto** = instancia de una clase

**Instancia** = elemento tangible (o sea que ocupa espacio en la memoria durante la ejecución del programa)

El encapsulamiento es consiste en controlar el acceso a los datos que conformas un objeto que conforman un objeto o instancia de una clase.

- Es decir indicar que métodos y atributos son **públicos** para poder revisar su contenido e incluso ser modificados.
- Y a su vez que métodos y atributos son **privados** para evitar el acceso a su contenido o que se realice alguna modificación en ellos.

En otras palabras el encapsulamiento consiste el elegir que métodos y atributo se van a ocultar de una clase, para que no sean modificados por una segunda clase, todo esto con el fin de que otro programador utilice la primera clase que creamos, no pueda modificar su estado o comportamiento de manera imprevista o incontrolada.

Para realizar el encapsulamiento es necesario usar modificadores de acceso, estos permiten dar niveles de seguridad restringiendo acceso a diferentes atributos, métodos o constructores. Obligando al usuario seguir una ruta especificada por nosotros para acceder a la información.

Cuando usamos: `Import java.util.Scanner;`

Solo importamos la librería o bilioteca donde se encuentra la clase Scanner para poder introducir datos desde el teclado, pero en ningún momento podemos modificar la clase Scanne, solo usarla.

Modificadores de acceso:

- `public`
- `protected`
- `default`
- `private`

## 8) Ejercicio de encapsulamiento



Samsung solicita el desarrollo de una clase programada en Java, para el funcionamiento lógico de su nueva línea de lavadoras, misma que puedan implementar sus programadores de manera sencilla en los programas desarrollados dentro de Samsung, con las siguientes características:

1. Debe recibir los kilos de ropa y tipo de ropa a través de argumentos.
2. Debe de realizar las funciones de llenado de agua, lavado y secado.
3. La clase debe estar correctamente encapsulada, para evitar que los programadores de Samsung utilicen métodos o variables que no son necesarios.
4. El único método disponible para importar, debe ser CicloFinalizado().

[Palabra reservada return](#)

[Non static](#)

**Primero hacemos la clase solicitada por Samsung**

```
//clase para las funciones logicas de una lavadora
public class LLfunciones {

    private int kilos=0, llenadoCompleto=0, TipoDeRopa=0,
    LavadoCompleto=0, SecadoCompleto=0;
    // al colocar private ya encapsulamos las variables
    public LLfunciones (int kilos, int TipoDeRopa) {
        this.kilos = kilos;
        this.TipoDeRopa = TipoDeRopa;
    }
    //metodos privados por que no queremos que nadie interfiera

    private void Llenado() {
        if(kilos<=12){ //suponiendo que la lavadora SOLO SOPORTA 12
        KILOS
            llenadoCompleto = 1;
            System.out.println("Llenando");
            System.out.println("Llenado terminado");
        }else{
            System.out.println("Demasiada carga, reduce la carga");
        }
    }
}
```

```

private void Lavado() {
    Llenado();
    if (llenadoCompleto == 1) {
        if (TipoDeRopa == 1) {
            System.out.println("Ropa blanca");
            System.out.println("Lavado suave");
            System.out.println("Lavando...");
            LavadoCompleto = 1;
        }
        if (TipoDeRopa == 2) {
            System.out.println("Ropa de color");
            System.out.println("Lavado intenso");
            System.out.println("Lavando...");
            LavadoCompleto = 1;
        }
        if (TipoDeRopa > 2) {
            System.out.println("Error eleccion incorrecta, se tratara
como ropa de color");
            System.out.println("Lavado intenso");
            System.out.println("Lavando...");
            LavadoCompleto = 1;
        }
    }
}

private void Secado() {
    Lavado();
    if (LavadoCompleto == 1) {
        System.out.println("Secando....");
        System.out.println("Secado completo");
        SecadoCompleto = 1;
    }
}

// solo este metodo es publico por que los programadores de samsung
necesitan acceso
public void CicloFinalizado() {
    Secado();
    if (SecadoCompleto == 1) {
        System.out.println("Ciclo terminado");
    }
}
}

```

Simulamos ser los programadores de Samsung y hacemos la clase main

```
package LLfunciones;
import LLfunciones.LLfunciones; // importamos la clase
//import nombredelpackage.nombredelaclase;
// para este ejemplo el nombre del package y de la clase son los mismos

import java.util.Scanner;

public class Lavadora_uno {
    public static void main (String[]args){
        Scanner entra = new Scanner(System.in);
        System.out.println("¿La ropa es blanca o de color?");
        int TipoDeRopa= entra.nextInt(); // nos salen las variables de la
        clase LLfunciones por que son privadas

        System.out.println("¿Kilos de ropa?");
        int kilos= entra.nextInt();

        LLfunciones mensajero = new LLfunciones(kilos,TipoDeRopa);
        mensajero.CicloFinalizado(); // solo sale el ciclo finalizado por
        que es publico
    }
}
```

Resultado

```
¿La ropa es blanca o de color?
2
¿Kilos de ropa?
5
Llenando
Llenado terminado
Ropa de color
Lavado intenso
Lavando...
Secando....
Secado completo
Ciclo terminado
```

```
¿La ropa es blanca o de color?  
3  
¿Kilos de ropa?  
5  
Llenando  
Llenado terminado  
Error eleccion incorrecta, se tratara como ropa de color  
Lavado intenso  
Lavando...  
Secando....  
Secado completo  
Ciclo terminado
```

```
¿La ropa es blanca o de color?  
1  
¿Kilos de ropa?  
7  
Llenando  
Llenado terminado  
Ropa balanca  
Lavado suave  
Lavando...  
Secando....  
Secado completo  
Ciclo terminado
```

```
¿La ropa es blanca o de color?  
1  
¿Kilos de ropa?  
13  
Demasiada carga, reduce la carga  
  
Process finished with exit code 0
```

### Conclusión

Podemos observar que en ese ejercicio se definieron variables globales tipo privado, por lo que tuvimos que usar un constructor para asignar las variables, además de que en cada método se iba llamando al método anterior para hacer uso de alguna variable o resultado, por ejemplo del **método secado** al **método ciclo final**, también hay que notar que en el **método ciclo final**

mandamos a imprimir el mensaje, y en el **método main** solo mandamos a llamar el **metodo ciclo final**.

## 9) Métodos Setter y Getter en Java (POO)

### Setter

El método set ("establecer") sirve para asignar valor a un atributo de nuestra clase, esto se hace de manera directa con este método, como este método no retorna a nada debe contener la palabra void en su estructura, y siempre debe recibir un parámetro de entrada

### Getter

El método get ("obtener"): accede a la clase para retornaros el valor de algún atributo que queramos, este método si debe retornar un valor por lo cual la estructura de este método debe contener el tipo de valor que vamos a retornar con ese método

### Usamos el ejemplo anterior

Ahora supondremos que quitaremos opciones de tipo de ropa y solo tendrá lavado intenso, sin quitar nada del código, ni modificando el acceso a public de las variables de la clase LLfunciones, ojo esto si se puede hacer pero para el ejemplo evitaremos hacerlo.

### Primero modificamos la clase solicitada por Samsung

```
//clase para las funciones logicas de una lavadora
public class LLfunciones {

    private int kilos=0, llenadoCompleto=0, TipoDeRopa=0,
LavadoCompleto=0, SecadoCompleto=0;
    // al colocar private ya encapsulamos las variables
    public LLfunciones (int kilos, int TipoDeRopa) {
        this.kilos = kilos;
        this.TipoDeRopa = TipoDeRopa;
    }
    //metodos privados por que no queremos que nadie interfiera

    private void Llenado() {
        if(kilos<=12){ //suponiendo que la lovadora SOLO SOPORTA 12
KILOS
            llenadoCompleto = 1;
            System.out.println("Llenando");
            System.out.println("Llenado terminado");
        }else{
            System.out.println("Demasiada carga, reduce la carga");
        }
    }

    private void Lavado() {
        Llenado();
        if (llenadoCompleto==1) {
```



```

        if(TipoDeRopa==1) {
            System.out.println("Ropa blanca");
            System.out.println("Lavado suave");
            System.out.println("Lavando...");
            LavadoCompleto=1;
        }
        if(TipoDeRopa==2) {
            System.out.println("Ropa de color");
            System.out.println("Lavado intenso");
            System.out.println("Lavando...");
            LavadoCompleto=1;
        }
        if(TipoDeRopa>2) {
            System.out.println("Error eleccion incorrecta, se tratara
como ropa de color");
            System.out.println("Lavado intenso");
            System.out.println("Lavando...");
            LavadoCompleto=1;
        }
    }
}

private void Secado() {
    Lavado();
    if(LavadoCompleto==1) {
        System.out.println("Secando....");
        System.out.println("Secado completo");
        SecadoCompleto=1;
    }
}

// solo este metodo es publico por que los programadores de samsung
necesitan acceso
public void CicloFinalizado() {
    Secado();
    if(SecadoCompleto==1) {
        System.out.println("Ciclo terminado");
    }
}

//Setter y Getter
//getter debe ser publico y poner el tipo de valor a regresar

public int getTipoDeRopa() {
    return TipoDeRopa; //con esto garantizamos que retorna valor a
la clase main
    // o podemos decir que obtiene la clase main
}

//Setter con void y la variable que debemos establecer entre
parentesis
public void setTipoDeRopa(int TipoDeRopa) { //con esto establecemos
nuestro metodo
    this.TipoDeRopa = TipoDeRopa; // con esto garantizamos que se
guarde en la variable que queremos
} // en teoria los metodos getter y setter son eso, metodos pero es
importante entender su funcion
}

```

Simulamos ser los programadores de Samsung y agregamos los metodos set y get

```
import Llfunciones.Llfunciones; // importamos la clase
//import nombredelpackage.nombredelaclase;
// para este ejemplo el nombre del package y de la clase son los mismos

import java.util.Scanner;

public class Lavadora_uno {
    public static void main (String[]args){
        Scanner entra = new Scanner(System.in);
        System.out.println("¿La ropa es blanca o de color?");
        int TipoDeRopa= entra.nextInt(); // nos salen las variables de la
        clase Llfunciones por que son privadas

        System.out.println("¿Kilos de ropa?");
        int kilos= entra.nextInt();

        Llfunciones mensajero = new Llfunciones(kilos,TipoDeRopa);
        mensajero.setTipoDeRopa(2); // con esto le indicamos al programa
        que en automatico asigne (establecer) el
        // valor 2 a la variable tipo de ropa
        System.out.println("Automaticamente el tipo de ropa es
        "+mensajero.getTipoDeRopa());
        // el anterior mensaje se concatena con el nombre del
        objeto(instancia) y el metodo get para devolver el valor
        // o podemos decir obtener el valor en la clase main
        mensajero.CicloFinalizado(); // solo sale el ciclo finalizado por
        que es publico
    }
}
```

## 10) Modificadores de acceso en Java

### Particularidades de por defecto y publico

Cuando declaramos una variable o metodo digamos en la clase1, pueden ser llamados desde cualquier otra clase, por ejemplo de la clase 2, siempre y cuando dicha clase este en el mismo package, lo mismo aplica con los métodos.

Si la clase se encuentra en un **package** diferente debemos importar la clase y utilizar un **mensajero (instancia)**.

### Por defecto

Cuando declaramos una variable:

```
Int numUno=0;
String nombre="";
```

A estos se les asigna un modificador por defecto.

### **Público**

Tiene la característica que todo lo declarado, tanto atributo (variable) como método (función) declarados, podrán ser utilizados o llamados o modificados.

### **Privado**

Son lo contrario del público, lo usamos cuando no queremos que modifiquen algún punto importante del código ya sea atributo o método.

- Cualquier otra clase del mismo paquete no podrá acceder a estos miembros.
- Las clases e interfaces no se pueden declarar como privadas (private).

Si queremos modificar por ejemplo una variable podemos hacer uso de los métodos **getter y setter**, por supuesto dichos métodos deben crearse públicos y en la misma clase donde se encuentran las variables a modificar.

### **Protected**

Se caracteriza por que solo compartirá atributos y métodos con la clase que este en el mismo **package** o la clase que tenga su **herencia**

## **11) Herencia o reutilización de código**

Podemos definirla como la reutilización de métodos y atributos de una clase ya creada en una nueva. Es decir sin tener que volver a escribir el código.

Al usar herencia podemos referirnos a 2 tipos de clases:

- La clase padre o clase base
- La clase hija o clase derivada

La clase padre es la que se crea primero y tiene las variables y métodos que vamos a reutilizar, la clase hija es quien reutiliza los métodos

En programación tenemos 2 tipos de herencia, simple y múltiple, en java solo contamos con la **herencia simple**, esto consiste en que una clase hijo solo puede tener una clase padre, es decir no puede tener más de un padre, mientras que una clase padre, si puede tener múltiples clases hijas.

## 12) Ejemplo de herencia

Creamos un programa con la clase padre y 2 hijas en un mismo package, finalmente el metodo main en otro package

Definimos la clase padre en package operaciones

```
package operaciones;
import java.util.Scanner;
public class ClasePadre {
    protected int valor1, valor2, resultado;
    //utilizamos protected para que las clases hijas que estan en el
    mismo package
    //tengan acceso
    Scanner entra = new Scanner(System.in);
    //metodo para recibir datos
    public void PedirDatos() {
        System.out.print("Dame el primer valor:");
        valor1=entra.nextInt();
        System.out.print("Dame el segundo valor:");
        valor2=entra.nextInt();
    }
    //metodo para imprimir
    public void Imprimir() {
        System.out.print("Resultado: "+resultado);
    }
}
```

Clase hija suma en mismo package

```
package operaciones;

public class ClaseHija_Suma extends ClasePadre{
    //usamos extends nombre de la clase padre para que la clase hija
    tenga la herencia
    public void Sumar() {
        resultado=valor1+valor2;
    }
}
```

Clase hija resta en mismo package

```
package operaciones;

public class ClaseHija_Resta extends ClasePadre{
    //usamos extends nombre de la clase padre para que la clase hija
    tenga la herencia
    public void Restar() {
        resultado=valor1-valor2;
    }
}
```

Creamos clase main en distinto package

```
package main;
import operaciones.ClaseHija_Resta;    //importamos solo las clases hijas
import operaciones.ClaseHija_Suma;    // ya que ellas ya tienen heredado
el valor                                // o codigo

import java.util.Scanner;

public class ClasePrincipal {
    public static void main(String[] args){
        Scanner entra = new Scanner(System.in);
        System.out.println("Para Sumar escribe 1");
        System.out.println("Para restar escribe 2");
        int op= entra.nextInt();
        if(op==1) {
            ClaseHija_Suma MensajeroSuma = new ClaseHija_Suma(); //creamos
            mensajero, objeto o instancia
            MensajeroSuma.PedirDatos(); //automaticamente nos sale el
            metodo pedir datos
            //ya que la clase hija ya heredo este metodo de la clase padre
            MensajeroSuma.Sumar(); //este metodo es propio de la clase
            hija
            MensajeroSuma.Imprimir(); // lo mismo pasa con este metodo

        }if (op==2){
            ClaseHija_Resta MensajeroResta = new ClaseHija_Resta();
            MensajeroResta.PedirDatos();
            MensajeroResta.Restar();
            MensajeroResta.Imprimir();
        }
    }
}
```

Resultado

```
Para Sumar escribe 1
Para restar escribe 2
1
Dame el primer valor:5
Dame el segundo valor:3
Resultado: 8
Process finished with exit code 0
```

```
Para Sumar escribe 1
Para restar escribe 2
2
Dame el primer valor:3
Dame el segundo valor:5
Resultado: -2
Process finished with exit code 0
```

```
Para Sumar escribe 1
Para restar escribe 2
3
Process finished with exit code 0
```

## 13) Polimorfismo

El polimorfismo en POO es la capacidad que se le da a un método, de comportarse de manera diferente de acuerdo a la instancia (mensajero u objeto) creada. Es decir dependiendo de la clase con la que se esté interactuando, será la función que va a ejecutar el método.

## 14) Ejemplo polimorfismo

Hacemos el mismo ejemplo anterior pero por practicidad dejamos todo en un solo package

### Clase padre operaciones

```
package polimorfismo;
import java.util.Scanner;
//clase padre
public abstract class Operaciones { // ya que tenemos polimorfismo
    //debemos colocar la palabra abstract
    protected int valor1, valor2, resultado;
    Scanner entrada = new Scanner(System.in);

    public void PedirDatos() {
        System.out.print("Dame el primer valor: ");
        valor1 = entrada.nextInt();
        System.out.print("Dame el segundo valor: ");
        valor2 = entrada.nextInt();
    }
    //metodo que usa el polimorfismo colocando abstract
    public abstract void Poli1(); // creamos metodo pero en lugar de usar
    //llaves usamos ;

    public void Imprimir() {
        System.out.print("El resultado es: "+resultado);
    }
}
```

### Clase hija suma

```
package polimorfismo;
//clase hija
public class Suma extends Operaciones {
    @Override //para sobre escribir
    public void Poli1() { // si no colocamos esta linea y la anterior
    marcara error
        resultado=valor1+valor2;
    }
}
```

### Clase hia resta

```
package polimorfismo;

public class Resta extends Operaciones {
    @Override
    public void Poli1() {
        resultado=valor1-valor2;
    }
}
```

## Clase Main

```
package polimorfismo;
import java.util.Scanner;

public class main {
    public static void main(String[] args) {
        Scanner entra= new Scanner(System.in);
        System.out.println("Para suma escribe 1");
        System.out.println("Para resta escribe 2");
        int op = entra.nextInt();

        if(op==1) {
            Operaciones Mensajero_suma = new Suma(); //creamos instancia
            //Clasepadre NombreMesnajero = new Clasehija;
            // con esto permitimos la comunicacion con la clase padre y el uso del
            metodo
            // abstracto o con polimorfismo
            Mensajero_suma.PedirDatos();
            Mensajero_suma.Poli1(); // aqui el metodo como se usa la clase
            hija suma
            // lo que hara es sumar
            Mensajero_suma.Imprimir();
        }
        if(op==2) {
            Operaciones Mensajero_resta = new Resta();
            Mensajero_resta.PedirDatos();
            Mensajero_resta.Poli1(); // aqui el metodo como se usa la clase
            hija resta
            // lo que hara es restar
            Mensajero_resta.Imprimir();
        }
        if(op>2){
            System.out.println("Error");
        }
    }
}
```

## Resultado

```
Para suma escribe 1
Para resta escribe 2
1
Dame el primer valor: 5
Dame el segundo valor: 6
El resultado es: 11
```

```
C:\Program Files\Java\jdk-17\bin\java.exe
Para suma escribe 1
Para resta escribe 2
2
Dame el primer valor: 3
Dame el segundo valor: 5
El resultado es: -2
Process finished with exit code 0
```

```
Para suma escribe 1
Para resta escribe 2
3
Error
```

```
Process finished with exit code 0
```

## 15) Palabra reservada Super

Se utiliza para poder acceder a un elemento en la clase padre, el uso más frecuente es llamar a un método o constructor en la clase padre.

## 16) Ejemplo uso reservado de palabra Super

### Clase padre

```
package clases;

public class Padre {
    public void Saludar() {
        System.out.println("Hola yo soy el padre");
    }
}
```

### Clase hija

```
package clases;

public class Hija extends Padre {

    public void saludar() {
        System.out.println("Hola soy la hija ");
    }
}
```

### Metodo main

```
package clases;

public class Main {
    public static void main (String[]args){
        Hija mensajero_hija = new Hija();
        mensajero_hija.saludar();
    }
}
```

### Resultado

```
Hola soy la hija

Process finished with exit code 0
```

Con los códigos anteriores definimos 2 métodos, uno en la clase padre y otro en la clase hija, hay que observar que los 2 métodos tienen el mismo nombre, y en la clase main, creamos la instancia y llamamos al método de la clase hija.

Haremos un cambio en la clase hija y lo demás lo dejaremos igual



### Cambios en la clase hija

```
package clases;

public class Hija extends Padre {

    public void saludar() {
        super.Saludar();
    }
}
```

### Resultados

```
Hola yo soy el padre

Process finished with exit code 0
```

Al colocar la palabra súper, un punto y el nombre del método estamos diciendo ahora que el método saludar hija estará llamando al método saludar del padre, por lo tanto al llamar al método saludar hija desde la clase main estamos activando el método saludar del padre.

## 17) Ámbito de variables en java

El ámbito de una variable define su alcance de uso, es decir, indica en que secciones de código una variable estará disponible.

Fuera de este ámbito, una variable no podrá ser accedida.

En java tenemos 3 tipos de ámbitos que pueden aplicar a una variable:

- **Local o de bloque:** son aquellas que solo pueden ser accedidas desde el bloque de código en el que son declaradas es decir entre las llaves del método , y son necesarias inicializar
- **Global o de instancia:** son aquellas que pertenecen a la clase donde han sido declaradas, y dependiendo del modificador( default, public ,private o protected) de acceso utilizado, podrán ser accedidas únicamente desde la misma clase, puedes o no puedes inicializar.
- **Estático o variables de clase:** son aquella que pertenecen a la propia clase donde han sido declaradas, y para poder acceder a ellas no es necesario crear una instancia de clases, sin importar que sean de distintos package

## 18) Ejemplo ámbito de las variables

Clase variable estatica 1

```
package clases;

public class estatica {
    public static int var1=555;
}
```

Clase variable estatica 2

```
package package2;

public class estatica2 {
    public static int var1=666;
}
```

Main

```
package clases;
import package2.estatica2;

public class Main {
    static int global1, global2;
    public static void imprimirGlobal1() {
        global1=333; // podemos utilizar la variable global en este
metodo
        System.out.println(global1);
        global2=333;
    }
    public static void main(String[] args) {
        int local=222;
        global2 = 444; //aqui tambien podemos utilizar la variable global
y asignar otro valor
        System.out.println(local);
        imprimirGlobal1(); //llamamos al metodo para imprimir
        System.out.println(global2);
        System.out.println(estatica.var1);
        System.out.println(estatica2.var1); // se 'puede utilizar pero
debemos llamar al package
    }
}
```

Obsérvese que a las variables globales tuvimos que colocar static para poder usarlas de quitarlas aparece error como se muestra a continuación

```

public class Main {
    2 usages
    int global1, global2;

    1 usage
    public static void imprimirGlobal1(){
        global1=333; // podemos utilizar la variable global en este metodo
        System.out.println(global1);
        global2=333;
    }

    public static void main(String[] args) {
        int local=222;
        global2 = 444; //aquí también podemos utilizar la variable global y asignar otro valor
        System.out.println(local);
        imprimirGlobal1(); //llamamos al metodo para imprimir
        System.out.println(global2);
        System.out.println(estatica.var1);
        System.out.println(estatica2.var1); // se 'puede utilizar pero debemos llamar al package
    }
}

```

## 19) Excepciones en java (try-catch-finally)

Son los medios que ofrecen algunos lenguajes de programación, para tratar situaciones anómalas que pueden suceder cuando ejecutamos un programa

## 20) Ejemplo excepciones

Realizamos un programa de división

```

package CURSO_INTERMEDIO;
import java.util.Scanner;

public class Excepciones {
    public static void main (String []args) {
        Scanner input = new Scanner(System.in);
        System.out.println("Write first number:");
        double numOne = input.nextDouble();
        System.out.println("Write second number:");
        double numTwo = input.nextDouble();
        double result = numOne /numTwo;
        System.out.println("The result of division is: "+result);
    }
}

```

En este programa al querer dividir por 0 nos marcara un error o si dividimos un carácter en lugar de un double

## Programa con try-catch-finally

```
import java.util.Scanner;

public class Excepciones {
    public static void main (String []args) {
        try { // dentro de las llaves try sucede lo que seria el
comportamiento normal
            Scanner input = new Scanner(System.in);
            System.out.println("Write first number:");
            double numOne = input.nextDouble();
            System.out.println("Write second number:");
            double numTwo = input.nextDouble();
            double result = numOne / numTwo;
            System.out.println("The result of division is: " + result);
        } catch (Exception e){ // entre llaves que pasara si pasa un error
            System.out.println("Error" + e); //debemos concatenar la
variable que declaramos en Exception
        } finally { // operacion o mensaje que sucedera al terminar con
try o catch
            System.out.println("Operation finished");
        }
    }
}
```

## Resultados

```
C:\Program Files\Java\jdk-17\bin\java.exe
Write first number:
20
Write second number:
6
The result of division is: 3.333333333333333
Operation finished

Process finished with exit code 0
```

Escribimos un carácter en lugar de un numero

```
C:\Program Files\Java\jdk-17\bin\java.exe
Write first number:
L
Error: java.util.InputMismatchException
Operation finished

Process finished with exit code 0
```

## 21) Constructores en java

Es un método que se ejecuta inicialmente y de manera automática

Un constructor tiene las siguientes características:

- Tiene el mismo nombre de la clase
- Es el primer método que se ejecuta
- No puede retornar datos, no lleva void (vacío)
- Se ejecuta una única vez

## 22) Ejemplo constructores en java (programa sin constructor)

Creemos un programa sencillo que pida nombre y lo imprima

Primero sin constructor, creamos la clase sin constructor y main

### Clase sin constructor

```
package clases;
import java.util.Scanner;
public class sinConstructor {
    Scanner input = new Scanner(System.in);
    String name="";
    public void getName(){ //metodo para pedir nombre
        System.out.println("What is your name?");

        name = input.nextLine();
    }
    public void toPrint(){ // metodo para regresar nombre
        System.out.println("Your name is :" + name);
    }
}
```

### Clase main

```
package clases;
public class main {
    public static void main (String []args){
        sinConstructor carrier1 = new sinConstructor(); // instancia
        carrier1.getName(); // llamamos al metodo
        carrier1.toPrint(); //llamamos al metodo
    }
}
```

### Resultado

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaa
What is your name?
Luis
Your name is :Luis

Process finished with exit code 0
|
```

## 23) Ejemplo constructores en java (programa con constructor)

Mismo ejemplo pero ahora con una clase donde definimos el constructor

### Clase constructor

```
package clases;
import java.util.Scanner;
public class constructor {
    public constructor(){ // definimos constructor, sin void, mismo
        nombre que la clase
        Scanner input =new Scanner(System.in);
        String name ="";
        System.out.println("What is your name:");
        name = input.nextLine();
        System.out.println("Your name is:"+ name);
        // los codigos son los mismos que sin constructor pero utilizamos
2 metodos
    }
}
```

### Clase main

```
package clases;
public class main {
    public static void main (String []args){
        constructor carrier2= new constructor();
    }
}
```

Observemos que esta vez solo creamos el mensajero

### Resultados

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\I
What is your name:
Paola
Your name is:Paola

Process finished with exit code 0
```

Y en automático el programa nos permite ejecutar todo lo que está en el constructor, porque recordemos que solo con crear la instancia el constructor se ejecuta

## 24) Hilos (Threads)

Es el flujo de control dentro de un programa, el cual permite tener múltiples procesos corriendo de forma simultánea.

Ejercicio: realiza un programa que imprima el mensaje de process 1 y process 2

**Clase main**

```
package Procesos1;

public class main {
    public static void main(String[] args) {
        for(int i=0; i<=5; i++){
            System.out.println("Process 1");
        }
        for(int i=0; i<=5; i++){
            System.out.println("Process 2");
        }
    }
}
```

**Resultado**

```
"C:\Program Files\Java\jdk-19\bin\jav
Process 1
Process 1
Process 1
Process 1
Process 1
Process 1
Process 2
Process 2
Process 2
Process 2
Process 2
Process 2
Process finished with exit code 0
```

Podemos observar que primero hace el proceso 1 y después el proceso 2

## 25) Ejercicios hilos (threads )

Realiza el ejercicio anterior pero ahora que utilice 2 hilos

### Clase process 1 con herencia ( extends)

```
package Procesos1;

public class Process1 extends Thread { // aqui heredamos propiedades de
una clase que viene por default en el JDK
    @Override
    public void run() {
        for (int i = 0; i <= 5; i++) {
            System.out.println("Process 1");
        }
    }
}
```

### Clase process 2 con implementación

```
package Procesos1;

public class Process2 implements Runnable { //en este caso estamos
implementando de la clase Runnable igual del JDK
    // tambien por que Runnable es una interfaz
    @Override //usamos el metodo run que esta en la interfaz por lo tanto
hay que sobrescribir
    public void run () {
        for (int i = 0; i <= 5; i++) {
            System.out.println("Process 2");
        }
    }
}
```

### Clase main

```
package Procesos1;

public class main {
    public static void main(String[] args) {
        Process1 hilo1 = new Process1(); // al ser una extension de una
clase
        //generamos la instancia (objeto) de forma comun clase nombre =
new clase();
        Thread hilo2 = new Thread(new Process2()); //al ser una
implementacion de una interfaz
        // debemos crear una instancia (objeto) tipo Thread
        //Thread nombre= new Threas(new ClaseDondeImplementamosInterfaz()
);
        hilo1.start(); //empezamos el objeto hilo 1
        hilo2.start(); //empezamos el objeto hilo 2
    }
}
```



debemos tener cuidado donde colocamos los procesos start, por que podemos obligar a que un proceso inicie antes que otro

### Resultados

```
Process 1
Process 1
Process 2
Process 2
Process 2
Process 2
Process 2
Process 2
Process 1
Process 1
Process 1
Process 1
```

Podemos observar que ahora (al realizar varios interacciones) el proceso 1 no funciona de forma continua, si no que se intercala con el proceso 2

## 26) Hilos con parámetros

Creamos otro proyecto con solo 2 clases, pero heredaremos la clase thread

### Clase procesos

```
package Clases;
public class Procesos extends Thread{ // extendemos (herencia) clase
padre Thread viene por default en JDK
    protected int parameteer;
    public Procesos (String nameThread){ // constructor donde definimos
la variable nameThread
        super(nameThread); // llamamos a la clase padre y pedimos que
asigne el nombre y lo guarde en la
        //variable name de la clase padre
    }
    @Override
    public void run(){
        for(int i=0;i<=parameteer;i++){
            System.out.println(i+getName()); // ya que tenemos la
herencia de la clase pade podemos llamar el
            // metodo getName
        }
    }
    //getter and setter
    public int getParameteer() {
        return parameteer;
    }
    public void setParameteer(int parameteer) {
        this.parameteer = parameteer;
    }
}
```

### Clase padre thread (ya por default en el JDK)

Método llamado con super

```
Initializes a new platform Thread. This constructor has the same effect as Thread (null, null, name).
This constructor is only useful when extending Thread to override the run() method.
Params: name – the name of the new thread
See Also: Inheritance when creating threads
228 @ public Thread( @NotNull String name) {
229     this( g: null, checkName(name), characteristics: 0, task: null, stackSize: 0, acc: null);
230 }
231
Initializes a new platform Thread. This constructor has the same effect as Thread (group, null, name).
```

Metodo get para regresar nombre

```
Returns this thread's name.
Returns: this thread's name.
See Also: setName(String)
@Contract(pure = true)
2015 @ public final String getName() {
2016     return name;
2017 }
2018
```

## Clase main

```
package Clases;
import java.util.Scanner;
public class main {
    public static void main (String[] args){
        Scanner input = new Scanner(System.in);
        System.out.println("Give the first parameter:");
        int parameter = input.nextInt();
        Procesos Hilo1 = new Procesos(" Thread 1");//mensajero hilo 1
        // entre parentesis mandamos el nombre del proceso 1 incluso el
        // en la variable nameThread declarada en el constructor
        Hilo1.setParameeter(parameter);// metodo set para mandar el
        // numero guardado en la variable parametro
        System.out.println("Give the second parameter:");
        parameter = input.nextInt();
        Procesos Hilo2 = new Procesos(" Thread 2");//mensajero hilo 2
        // entre parentesis mandamos el nombre del proceso 2 incluso el
        // en la variable nameThread declarada en el constructor
        Hilo2.setParameeter(parameter);
        System.out.println("Give the third parameter:");
        parameter = input.nextInt();
        Procesos Hilo3 = new Procesos(" Thread 3");//mensajero hilo 3
        // entre parentesis mandamos el nombre del proceso 3 incluso el
        // en la variable nameThread declarada en el constructor
        Hilo3.setParameeter(parameter);
        //observemos que solo usamos una unica variable "parameter" y que
        //definimos 3 instancias, cada instancia se
        //encarga de guardar el numero y ya no la variable

        Hilo1.start(); // pedimos que inicie hilo 1
        Hilo2.start(); // pedimos que inice hilo 2
        Hilo3.start(); // pedimos que inice hilo 3

    }
}
```

## Resultado

```
Give the first parameter:
Give the second parameter:
Give the third parameter:
0 Thread 1
1 Thread 1
2 Thread 1
3 Thread 1
4 Thread 1
5 Thread 1
0 Thread 2
1 Thread 2
2 Thread 2
3 Thread 2
4 Thread 2
5 Thread 2
6 Thread 1
7 Thread 1
8 Thread 1
9 Thread 3
1 Thread 3
2 Thread 3
3 Thread 3
```

## 27) Estado de los hilos (new)

Los hilos presentan distintas etapas o etapas:

**New (nuevo):** El hilo, ha sido creado pero no inicializado, es decir no se ha ejecutado todavía el método start(). Se producirá un mensaje de error (IllegalThreadStateException) si se intenta ejecutar cualquier método de la clase Thread, Excepto con el método start().

**Clase hilo\_procesos (clase hija de la clase padre Thread)**

```
package Clases;
public class Hilo_Procesos extends Thread{
    @Override
    public void run(){
        for (int i=0;i<=5;i++){
            System.out.println(i);
        }
    }
}
```

**Clase main**

```
package Clases;
public class main {
    public static void main (String []args){
        Hilo_Procesos Hilo1 = new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo_Procesos Hilo2= new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
    }
}
```

## 28) Estado de los hilos (runnable)

**Runnable(ejecutable):** cuando el método start() crea los recursos del sistema necesarios para ejecutar el hilo, programa el thread para ejecutarse, y llama al método run() del thread. En este punto el hilo está en el estado ejecutable

**Clase main**

```
package Clases;
public class main {
    public static void main (String []args) {
        Hilo_Procesos Hilo1 = new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo_Procesos Hilo2= new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo1.start(); //a partir de aqui esta en el estado runnable
        Hilo2.start(); //a partir de aqui esta en el estado runnable
    }
}
```

## 29) Estado de los hilos (Blocked) método sleep

**Blocked o not runnable (bloqueado):** En este estado, el hilo se encuentra en ejecución, pero existe una tarea o actividad del mismo hilo que lo impide

### Clase main primera forma

```
package Clases;
public class main {
    public static void main (String []args) throws InterruptedException {
// en automatico el programa nos puede poner throws para el método sleep
        Hilo_Procesos Hilo1 = new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo_Procesos Hilo2= new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo1.start(); //apartir de aqui esta en el estado runnable
        Hilo1.sleep(1000); //metodo sleep para bloquear aqui declaramos que se
        atrasara el proceso 1 segundo
        Hilo2.start(); //apartir de aqui esta en el estado runnable
    }
}
```

### Clase main segunda forma

```
package Clases;
public class main {
    public static void main (String []args){
        Hilo_Procesos Hilo1 = new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo_Procesos Hilo2= new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo1.start(); //apartir de aqui esta en el estado runnable
        try { // saldra error si solo ponemos la linea de abajo, debemos
            poner try y catch
            // el metodo en un try y catch
            Hilo1.sleep(1000); //metodo sleep para bloquear aqui declaramos
            que se atrasara el proceso 1 segundo
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        Hilo2.start(); //apartir de aqui esta en el estado runnable
    }
}
```

Con esto hacemos que se atarse por 1 segundo el hilo 1

Ahora haremos que se atarse 3 segundos el hilo 1 y 2 segundos el hilo 2, agregaremos el método getName en el ciclo for de la clase hilo\_ procesos, para ver que en efecto se atrase cada uno

### Clase hilo\_procesos (clase hija de la clase padre Thread)

```
package Clases;
public class Hilo_Procesos extends Thread{
    @Override
    public void run(){
        for (int i=0;i<=5;i++){
            System.out.println(i +" "+ getName() );
        }
    }
}
```

### Clase main

```
package Clases;
public class main {
    public static void main (String []args){
        Hilo_Procesos Hilo1 = new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo_Procesos Hilo2= new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo1.start(); //apartir de aqui esta en el estado runnable
        try { // saldra error si solo ponemos la linea de abajo, debemos
            poner try y catch
            Hilo1.sleep(3000); //metodo sleep para bloquear aqui declaramos
            que se
            // atrasara el proceso 1 segundo
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        Hilo2.start(); //apartir de aqui esta en el estado runnable
        try {
            Hilo2.sleep(2000); // que se detenga en 2 segundos
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

### Resultado

```
"C:\Program Fi
0 Thread-0
1 Thread-0
2 Thread-0
3 Thread-0
4 Thread-0
5 Thread-0
0 Thread-1
1 Thread-1
2 Thread-1
3 Thread-1
4 Thread-1
5 Thread-1
```

### 30) Estado de los hilos (Blocked) método sleep aplicado a la clase hija en bucle for

Ahora causaremos un atraso en el proceso del bucle

**Clase hilo\_procesos (clase hija de la clase padre Thread)**

```
package Clases;
public class Hilo_Procesos extends Thread{
    @Override
    public void run(){
        for (int i=0;i<=5;i++){
            System.out.println(i + " " + getName() );
            try { // tambien necesitamos try
                Hilo_Procesos.sleep(1000); // indicamos que el metodo
                sleep se usara en la clase Hilo_proceso
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

**Clase main**

```
package Clases;
public class main {
    public static void main (String []args){
        Hilo_Procesos Hilo1 = new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo_Procesos Hilo2= new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo1.start(); //apartir de aqui esta en el estado runnable
        try { // saldra error si solo ponemos la línea de abajo, debemos
        poner try y catch
            Hilo1.sleep(3000); //metodo sleep para bloquear aqui declaramos
            que se
            // atrasara el proceso 1 segundo
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        Hilo2.start(); //apartir de aqui esta en el estado runnable
        try {
            Hilo2.sleep(2000); // que se detenga en 2 segundos
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

### Resultado

```
0 Thread-0
1 Thread-0
2 Thread-0
0 Thread-1
3 Thread-0
1 Thread-1
4 Thread-0
2 Thread-1
5 Thread-0
3 Thread-1
4 Thread-1
5 Thread-1
```

## 31) Estado de los hilos (dead / finalizado) método stop o interrupt

Un hilo puede morir de dos formas: por causas naturales o siendo asesino (parado). Una muerte natural se produce cuando su método run() termina normalmente (lo que ya trabajamos), mientras que una muerte provocada se produce cuando existe una instrucción que obligue al hilo a finalizar sin haber concluido su tarea por completo

Ahora dejaremos morir de forma natural el hilo 1 y mataremos al hilo 2

### Clase main

```
package Clases;
public class main {
    public static void main (String []args){
        Hilo_Procesos Hilo1 = new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo_Procesos Hilo2= new Hilo_Procesos(); // creamos la instancia
        aqui tenemos el estado new
        Hilo1.start(); //apartir de aqui esta en el estado runnable
        try { // saldra error si solo ponemos la linea de abajo, debemos
            poner try y catch
            Hilo1.sleep(3000); //metodo sleep para bloquear aqui declaramos
            que se
            // atrasara el proceso 1 segundo
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
        Hilo2.start(); //apartir de aqui esta en el estado runnable
        Hilo2.stop(); // con esto interrumpimos el hilo
        try {
            Hilo2.sleep(2000); // que se detenga en 2 segundos
```



```
    } catch (InterruptedException e) {  
        throw new RuntimeException(e);  
    }  
}  
}
```

### Resultado

```
0 Thread-0  
1 Thread-0  
2 Thread-0  
3 Thread-0  
4 Thread-0  
5 Thread-0
```

```
Process finished with exit code 0
```

## 32) Sincronización de hilos

La sincronización de hilos permite controlar el tiempo y forma en que se ejecutan los hilos sin provocar que se entorpezcan entre sí o establecer un orden de ejecución

Haremos un programa que imprima la palabra LUIS con distintos hilos

### Clase hilo1

```
package Clases;
public class Hilo1 extends Thread{
    @Override public void run(){
        for(int i=0; i<=5;i++){
            System.out.println("L");
            try {
                Hilo1.sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

### Clase hilo2

```
package Clases;
public class Hilo2 extends Thread{
    @Override public void run(){
        for(int i=0; i<=5;i++){
            System.out.println("U");
            try {
                Hilo2.sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

### Clase hilo3

```
package Clases;
public class Hilo3 extends Thread{
    @Override public void run(){
        for(int i=0; i<=5;i++){
            System.out.println("I");
            try {
                Hilo3.sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

### Clase hilo4

```
package Clases;
public class Hilo4 extends Thread{
    @Override public void run(){
        for(int i=0; i<=5;i++){
            System.out.println("S");
            try {
                Hilo4.sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

### Clase main

```
package Clases;
public class main {
    public static void main(String []args){
        Hilo1 mHilo1 = new Hilo1();
        Hilo2 mHilo2 = new Hilo2();
        Hilo3 mHilo3 = new Hilo3();
        Hilo4 mHilo4 = new Hilo4();

        mHilo1.start();
        try {
            mHilo1.sleep(1000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        mHilo2.start();
        try {
            mHilo2.sleep(1000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        mHilo3.start();
        try {
            mHilo3.sleep(1000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        mHilo4.start();
        try {
            mHilo4.sleep(1000);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

## Resultado

L  
L  
L  
L  
L  
L  
U  
U  
U  
U  
U  
U  
I  
I  
I  
I  
I  
I  
S  
S  
S  
S  
S

Hasta aquí hemos utilizado lo que hemos aprendido pero aun no muestra la palabra Luis en orden

### 33) Sincronización de hilos ejemplo

Modifica println por print y agregamos un println en el Hilo4, y reducimos el tiempo de 1000 a 10 en el método main

#### Clase Hilo1

```
package Clases;
public class Hilo1 extends Thread{
    @Override public void run(){
        for(int i=0; i<=5;i++){
            System.out.print("L");
            try {
                Hilo1.sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

#### Clase Hilo2

```
package Clases;
public class Hilo2 extends Thread{
    @Override public void run(){
        for(int i=0; i<=5;i++){
            System.out.print("U");
            try {
                Hilo2.sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

#### Clase Hilo3

```
package Clases;
public class Hilo3 extends Thread{
    @Override public void run(){
        for(int i=0; i<=5;i++){
            System.out.print("I");
            try {
                Hilo3.sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

### Clase Hilo4

```
package Clases;
public class Hilo4 extends Thread{
    @Override public void run(){
        for(int i=0; i<=5;i++){
            System.out.print("S");
            System.out.println("");

            try {
                Hilo4.sleep(100);
            } catch (InterruptedException e) {
                throw new RuntimeException(e);
            }
        }
    }
}
```

### Clase main

```
package Clases;
public class main {
    public static void main(String []args){
        Hilo1 mHilo1 = new Hilo1();
        Hilo2 mHilo2 = new Hilo2();
        Hilo3 mHilo3 = new Hilo3();
        Hilo4 mHilo4 = new Hilo4();

        mHilo1.start();
        try {
            mHilo1.sleep(10);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        mHilo2.start();
        try {
            mHilo2.sleep(10);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        mHilo3.start();
        try {
            mHilo3.sleep(10);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }

        mHilo4.start();
        try {
            mHilo4.sleep(10);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }
}
```

## Resultado

```
"C:\Program Files\Java\jdk-19\bin\ja
LUIS
LUIS
LUIS
LUIS|
LUIS
LUIS
LUIS

Process finished with exit code 0
```

## 34) Clase Math: Numero e y pi

```
package ClaseMath;

public class Constantes {
    public static void main (String[]args){
        System.out.println("The number e : "+Math.E);//llamar num euler
        System.out.println("The number PI : "+Math.PI);//llamar numero pi
    }
}
```

## Resultado

```
"C:\Program Files\Java\jdk-19\bin\java.exe
The number e : 2.718281828459045
The number PI : 3.141592653589793

Process finished with exit code 0
```

### 35) Clase Math: conversión de grados a radianes y viceversa

```
package ClaseMath;

public class ConversionesGrados {
    public static void main (String[]args){
        double angleDegrees=180;
        double angleRadian = Math.toRadians(angleDegrees); // convertimos a
radian
        System.out.println("The angle in Degrees "+angleDegrees+ "° is
"+angleRadian+" in radian");
        angleRadian = Math.toRadians(angleDegrees) / Math.PI; // al radian le
quitamos el numero pi
        System.out.println("The angle in Degrees "+angleDegrees+ "° is
"+angleRadian+"π in radian");
    }
}
```

#### Resultado

```
The angle in Degrees 180.0° is 3.141592653589793 in radian
The angle in Degrees 180.0° is 1.0 π in radian

Process finished with exit code 0
```



## 36) Clase Math: funciones trigonométricas básicas

```
package ClaseMath;

public class FuncionesTrigonometricas {
    public static void main(String[] args) {
        double angleDegrees = 45, result = 0;
        // para que funcionen adecuadamente los calculos necesitamos
        // convertir gradianes a radianes
        double angleRadian = Math.toRadians(angleDegrees);
        //Seno
        result = Math.sin(angleRadian);
        System.out.println("The sine of the angle " + angleDegrees + "is
" + result);
        //coseno
        result = Math.cos(angleRadian);
        System.out.println("The cosine of the angle " + angleDegrees +
"is " + result);
        //tangente
        result = Math.tan(angleRadian);
        System.out.println("The tangent of the angle " + angleDegrees +
"is "+ result);

        //calculo de angulos
        double data = 0.906;
        // sen^-1 o arcoseno
        angleRadian= Math.asin(data);
        angleDegrees=Math.toDegrees(angleRadian);
        System.out.println("Sin^-1("+data+")= "+angleDegrees);
        // cos^-1 o arcocoseno
        angleRadian= Math.acos(data);
        angleDegrees=Math.toDegrees(angleRadian);
        System.out.println("Cos^-1("+data+")= "+angleDegrees);
        // sen^-1 o arcotang
        angleRadian= Math.atan(data);
        angleDegrees=Math.toDegrees(angleRadian);
        System.out.println("Tan^-1("+data+")= "+angleDegrees);
    }
}
```

### Resultado

```
The sine of the angle 45.0is 0.7071067811865475
The cosine of the angle 45.0is 0.7071067811865476
The tangent of the angle 45.0is 0.9999999999999999
Sin^-1(0.906)= 64.95830480700477
Cos^-1(0.906)= 25.041695192995235
Tan^-1(0.906)= 42.17657760838539

Process finished with exit code 0
```

### 37) Clase Math: Raiz cuadrada y controlando número de decimales

```
package ClaseMath;
import java.text.DecimalFormat; // se debe importar para la primer forma
de controlar el numero de decimales
import java.math.BigDecimal; // se debe importar para la cuarta forma de
controlar el numero de decimales
import java.math.RoundingMode; // se debe importar para la cuarta forma de
controlar el numero de decimales

public class RaizyNumDecimales {
    public static void main (String[] args) {
        double data=15.6;
        int data2=0;
        double result= Math.sqrt(data);
        System.out.println("The square root of "+data+" is "+result);

        // primera forma para poner menos decimales clase DecimalFormat
        DecimalFormat df = new DecimalFormat("#.00"); // creamos objeto
        DecimalFormat
        //DecimalFormat Nombreobjeto = new DecimalFormat
        ("#.numeroDeDecimales);
        System.out.println("The square root of "+data+" whit 2 decimal
        places is "+df.format(result));

        //df.format(variable)
        // Segunda form para poner menos decimales String.format
        System.out.println("The square root of "+data+" whit 3 decimal
        places is "+String.format("%.3f",result));

        //String.format("%.numeroDecimalesf,variable)

        //tercer metodo para poner menos decimales Math.round
        System.out.println("The square root of "+data+" whit 4 decimal
        places is "+(double)Math.round(result*10000d)/10000);
        //el numero de ceros despues del 1 son la cantidad de decimales
        se pone en ambos 1___d)/1___

        //cuarto metodo para poner menos decimales BigDecimal
        BigDecimal BD = new BigDecimal(result); // creamos onjeto
        BigDecimal y entre parentesis la variable
        BD = BD.setScale(5, RoundingMode.HALF_UP); //parentesis para
        numero de decimales
        System.out.println("The square root of "+data+" whit 5 decimal
        places is "+BD.doubleValue());
        // al colocar esta sentencia le decimos al sistema qe convierta
        numero entero y en automatico redondea al numero menor
        data2= (int) (Math.sqrt(data));
        System.out.println("The rounded square root of "+data+" is
        "+data2);

    }
}
```

## Resultado

```
"C:\Program Files\Java\jdk-19\bin\java.exe" "-javaagent:C:\Pr
The square root of 15.6 is 3.9496835316262997
The square root of 15.6 whit 2 decimal places is 3.95
The square root of 15.6 whit 3 decimal places is 3.950
The square root of 15.6 whit 4 decimal places is 3.9497
The square root of 15.6 whit 5 decimal places is 3.94968
The rounded square root of 15.6 is 3

Process finished with exit code 0
```

## 38) Clase Math: números aleatorios

### Clase main

```
package ClaseMath;
import java.util.Random;

public class NumeroAleatorio {
    public static void main (String[]args){
        int random1=0;
        //primera forma
        for(int i =0 ; i<=5;i++) {
            random1 = (int) (Math.random() * 100);
            //variable = (int) (Math.random()*rango de numero aleatorio);
            System.out.println(random1);
        }
        System.out.println("");
        //segunda forma
        Random r = new Random();
        for(int i =0 ; i<=5;i++) {
            random1= (int) (r.nextDouble()*100);
            //variable = (int) (objetoTipoRandom.nexDouble *rango de numero
            aleatorio);
            // podemos cambiar nextDouble por el tipo de dato que necesitemos
            por ejemplo nextFloat()
            System.out.println(random1);
        }
    }
}
```

Tambien podemos utilizar en el segundo método nextGaussian()

## Resultado

```
"C:\Pro  
41  
68  
33  
68  
55  
18  
  
4  
22  
96  
37  
47  
34
```

### 39) Suma y resta de matrices

Hay que recordar que cuando sumas y restas una matriz solo se pueden sumar y restar matrices de las mismas dimensiones (2x2, 3x3, 4x4, etc.) y el resultado también tendrá las mismas dimensiones.

#### Clase main

```
package ClaseMath;

public class SumaRestaMatriz {
    public static void main (String[] args) {
        int matrix1 [][] = new int[3][3];
        int matrix2 [][] = new int[3][3];
        int matrix3 [][] = new int[3][3];
        //generando matriz aleatoria 1
        for(int i = 0; i<matrix1.length; i++){
            for(int j=0; j<matrix1.length;j++){
                matrix1[i][j]= (int) (Math.random()*5);
            }
        }
        //generando matriz aleatoria 2
        for(int i = 0; i<matrix2.length; i++){
            for(int j=0; j<matrix2.length;j++){
                matrix2[i][j]= (int) (Math.random()*5);
            }
        }
        //generando sumatoria y guardando en matriz 3
        for(int i = 0; i<matrix3.length; i++){
            for(int j=0; j<matrix3.length;j++){
                matrix3[i][j]= matrix1[i][j]+matrix2[i][j];
            }
        }
        // imprimiendo matrices
        for(int i = 0; i<matrix1.length; i++) { //inicia for 1
            for (int j = 0; j < matrix1.length; j++) {
                System.out.print "[" + matrix1[i][j] + " ";
            }
            if (i == 1) {
                System.out.print(" + ");
            }else {
                System.out.print(" ");
            }

            for (int j = 0; j < matrix2.length; j++) {
                System.out.print "[" + matrix2[i][j] + " ";
            }
            if (i == 1) {
                System.out.print(" = ");
            }else {
                System.out.print(" ");
            }

            for (int j = 0; j < matrix3.length; j++) {
                System.out.print "[" + matrix3[i][j] + " ";
            }
            System.out.println("");
        } // termina for 1
    }
}
```

```

}
}

```

#### Resultado

```

0. Program Files (x86)\jdk-17\bin\java.exe
[0][1][3]    [4][3][3]    [4][4][6]
[1][0][3] + [2][0][0] = [3][0][3]
[2][2][2]    [4][3][2]    [6][5][4]

Process finished with exit code 0

```

## 40) Multiplicación de matrices

En el caso de la multiplicación el número de filas de la matriz1 debe tener el número de columnas de la matriz 2, y el número de columnas de la matriz 1 debe ser el número de filas de la matriz 2, y la matriz resultante debe tener el mismo número de filas de la matriz1 y el mismo número de columnas de la matriz2

Ejemplo

Matriz\_uno

5	2	→ 3
9	4	1

**X**

Matriz\_dos

4	7
5	2
→ 2	1

**=**

Matriz\_resultante


**(5 \* 4) + (2 \* 5) + (3 \* 2) = 36**

Matriz\_uno

5	2	3
9	4	→ 1

**X**

Matriz\_dos

4	7
5	2
→ 2	1

**=**

Matriz\_resultante

<b>36</b>	

**(9 \* 4) + (4 \* 5) + (1 \* 2) = 58**

Matriz\_uno                      Matriz\_dos                      Matriz\_resultante

5	2	→ 3
9	4	1

×

4	7
5	2
2	→ 1

=

36	
58	

(5 \* 7) + (2 \* 2) + (3 \* 1) = 42

Matriz\_uno                      Matriz\_dos                      Matriz\_resultante

5	2	3
9	4	→ 1

×

4	7
5	2
2	→ 1

=

36	42
58	

(9 \* 7) + (4 \* 2) + (1 \* 1) = 72

Matriz\_resultante

36	42
58	72

#### 41) Ejercicio multiplicación de matrices

Realiza un programa donde introduzcas el valor de las filas y columnas de las matrices, coloque el valor de cada posición de la matriz en aleatorio (entre 0 y 5), realice la multiplicación e imprima las 3 matrices

## Clase main

```
package ClaseMath;
import java.util.Scanner;

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

        Scanner input = new Scanner(System.in);
        System.out.println("please, enter the number of rows of matrix 1
and the number");
        int data1=input.nextInt();
        System.out.println("please, enter the number of columns of matrix
1 and the number");
        int data2=input.nextInt();
        System.out.println("then the number of rows of matrix 2 is:"+
data2);
        System.out.println("then the number of columns of matrix 2 is "+
data1);

        int matrix1 [][] = new int[data1][data2];
        int matrix2 [][] = new int[data2][data1];
        int matrix3 [][] = new int[data1][data1];

        //generando matriz aleatoria 1
        for(int i = 0; i<data1; i++){
            for(int j=0; j<data2;j++){
                matrix1[i][j]= (int) (Math.random()*5);
            }
        }
        //generando matriz aleatoria 2
        for(int i = 0; i<data2; i++){
            for(int j=0; j<data1;j++){
                matrix2[i][j]= (int) (Math.random()*5);
            }
        }
        //realizando operaciones
        for(int i=0; i<data1;i++){
            for(int j=0; j< data1;j++ ){
                for(int k=0; k<data2;k++){
                    matrix3[i][j] += matrix1[i][k]*matrix2[k][j];
                }
            }
        }
        // imprimiendo matriz 1
        System.out.println("Matrix 1");
        for(int i = 0; i<data1; i++) {
            for (int j = 0; j < data2; j++) {
                System.out.print "[" + matrix1[i][j] + " ";
            }
            System.out.println("");
        }
        System.out.println("");
        //imprimiendo matriz 2
        System.out.println("Matrix 2");
        for(int i = 0; i<data2; i++) {
            for (int j = 0; j < data1; j++) {
                System.out.print "[" + matrix2[i][j] + " ";
            }
        }
    }
}
```



```

        }
        System.out.println("");
    }
    System.out.println("");
    //imprimiendo matriz 3
    System.out.println("Matrix 3");
    for(int i = 0; i<data1; i++) {
        for (int j = 0; j < data1; j++) {
            System.out.print "[" + matrix3[i][j] + " " );
        }
        System.out.println("");
    }
}
}

```

## Resultado

```

please, enter the number of rows of matrix 1 and the number
2
please, enter the number of columns of matrix 1 and the number
3
then the number of rows of matrix 2 is:3
then the number of columns of matrix 2 is 2
Matrix 1
[4][1][1]
[4][2][0]

Matrix 2
[3][1]
[0][2]
[2][2]

Matrix 3
[14][8]
[12][8]

```

```

please, enter the number of rows of matrix 1 and the number
5
please, enter the number of columns of matrix 1 and the number
5
then the number of rows of matrix 2 is:3
then the number of columns of matrix 2 is 3
Matrix 1
[0][0][4]
[3][4][1]
[4][2][4]

Matrix 2
[1][4][1]
[1][1][0]
[2][1][4]

Matrix 3
[8][4][16]
[9][17][7]
[14][22][20]

```

## 42) Palabra reservada return

La palabra return se utiliza en los métodos donde debemos regresar un valor siempre y cuando al método se le coloque el tipo de valor a regresar int, float, double, etc.

Ejemplo crea un programa que realice la suma de 2 valores, crea 2 clases

### Clase main

```
package My;
public class ReturnMain {
    public static void main(String[] args) {
        ReturnSuma obj1 = new ReturnSuma(); // creamos objeto
        System.out.println("The result is:"+obj1.sumar(5,4)); // mandamos
a imprimir y al mismo tiempo
        // llamamos al metodo sumar y mandamos los valores que se
declaran en el metodo sumar
        // incluso el programa muestra que valor se asigna en a y en b
    }
}
```

### Clase ReturnSuma

```
package My;
public class ReturnSuma {
    public int sumar(int a, int b){ // ponemos el tipo de valor a
regresar
        int suma =a+b; // declaramos nuestra variable del mismo valor
        return suma; //y declaramos que vamos a regresar ese valor
    }
}
```

### Resultado

```
"C:\Program Files\Java\jdk-19\bin\java.exe
The result is:9

Process finished with exit code 0
```

### Conclusión

Si analizamos el [Ejercicio de encapsulamiento](#), podemos observar que en ese ejercicio se definieron variables globales tipo privado, por lo que tuvimos que usar un constructor para asignar las variables, además de que en cada método se iba llamando al método anterior para hacer uso de alguna variable o resultado, por ejemplo del **método secado** al **método ciclo final**, también hay que notar que en el **metodo ciclo final** mandamos a imprimir el mensaje, y en el **método main** solo mandamos a llamar el **metodo ciclo final**.

También observemos que los métodos solo tienen la palabra **Void**, (vacío) por lo que no tenemos que usar **return**, ya que no regresaremos ningún valor.

En este caso no necesitamos las variables globales porque en el método estamos declarando **variables locales** y regresando un valor y en el método main solo imprimimos el valor regresado

### 43) Palabra reservada static

Ahora vamos a completar el problema anterior, esta vez crearemos un método en la misma clase donde se encuentra el método main

**Clase main**

```
public class ReturnMain {
    // ahora lo hacemos creando el metodo en la misma clase pero fuera
    // del metodo main observe que el metodo
    // puede estar antes del metodo main
    public static int sumar2(int a, int b){ // ponemos el tipo de valor a
    regresar y ademas static
        // con static podemos decir que el metodo puede ser usado en esta
    misma clase
        int suma =a+b; // declaramos nuestra variable del mismo valor
        return suma; //y declaramos que vamos a regresar ese valor
    }
    public static void main(String[]args){
        ReturnSuma obj1 = new ReturnSuma(); // creamos objeto
        System.out.println("The result is :"+obj1.sumar(5,4)); //
    mandamos a imprimir y al mismo tiempo
        // llamamos al metodo sumar y mandamos los valores que se
    declaran en el metodo sumar
        // incluso el programa muestra que valor se asigna en a y en b
        System.out.println("The result is :"+ sumar2(2,8));
        // como el metodo esta en la misma clase y tiene la palabra
    static podemos imprimir
        // solo llamando el metodo, pero al igual que con la instancia
    mandamos los valores
        // ahora veamos que podemos utilizar el valor que nos manda el
    metodo
        int c=5, resultado=0; // definimos 2 nuevas variables
        resultado= sumar2(2,8)+c; // sumamos
        System.out.println("the new result is :"+resultado);// imprimimos
    }
}
```

**Resultado**

```
The result is :9
The result is :10
the new result is :15
```

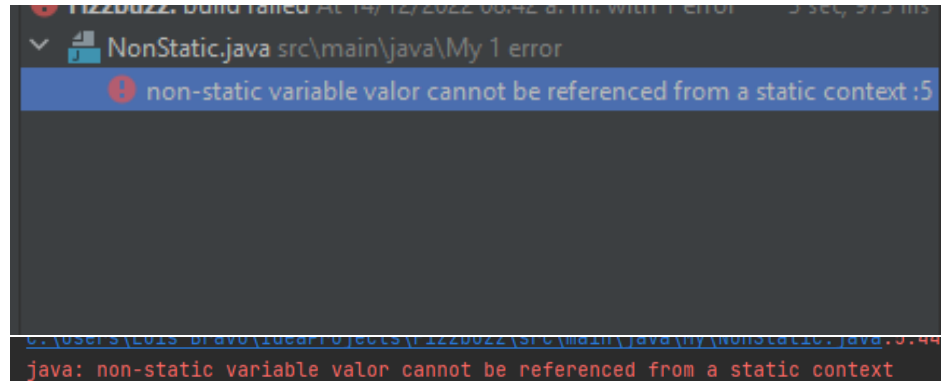
### 44) Exception Non static

El Non Static resutla cuando nosotros tenemos una variable (atributo o campo) o metodo que no son estáticos y por ende no se pueden usar en los metodo de la misma clase un ejemplo es el siguiente

### Clase main

```
package My;
public class NonStatic {
    int valor = 5;
    public static void main(String[] args) {
        System.out.println("el valor es: "+valor);
    }
}
```

### Resultado



Build failed At 14/12/2022 00:42:0. With 1 error 0.0 sec, 975 ms

NonStatic.java src/main/java/My 1 error

non-static variable valor cannot be referenced from a static context :5

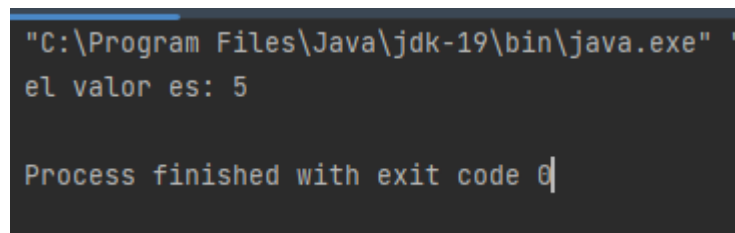
C:\Users\Luis Bravo\IdeaProjects\1220022\src\main\java\My\NonStatic.java:5:44  
java: non-static variable valor cannot be referenced from a static context

Podemos observar que nos indica el error ahora si colocamos static

### Clase main

```
package My;
public class NonStatic {
    static int valor = 5;
    public static void main(String[] args) {
        System.out.println("el valor es: "+valor);
    }
}
```

### Resultado



```
"C:\Program Files\Java\jdk-19\bin\java.exe" "  
el valor es: 5  
  
Process finished with exit code 0
```

### Conclusión

Si analizamos el [Ejercicio de encapsulamiento](#) podemos observar que en la clase LLfunciones colocamos el modificador de acceso private con lo que indicamos que las variables y método pertenecen a la misma clase, por lo que no tuvimos que utilizar static.

## 45) Ejercicio return

Realiza un programa que realice suma, resta, multiplicación y división solo introduciendo 2 valores

### Clase main

```
package My;
import java.util.Scanner;
public class Return2Main {
    static int a=0, b=0;
    public static void main(String[]arg) {
        Scanner input = new Scanner(System.in);
        System.out.print("Please enter first number: ");
        a= input.nextInt();
        System.out.print("Please enter second number: ");
        b= input.nextInt();
        Return2 obj1 = new Return2();
        System.out.println("the result of addition is "
+obj1.sumar(a,b)+"");
        System.out.println("the result of subtraction is "
+obj1.restar(a,b)+"");
        System.out.println("the result of multiplication is "
+obj1.multiplicar(a,b)+"");
        System.out.println("the result of division is "
+obj1.dividir(a,b)+"");
    }
}
```

### Clase Return 2

```
public class Return2 {

    public int sumar(int c, int d){
        int resultado =c+d;
        return resultado;
    }
    public int restar(int c, int d){
        int resultado =c-d;
        return resultado;
    }
    public int multiplicar(int c, int d){
        int resultado =c*d;
        return resultado;
    }
    public int dividir(int c, int d){
        int resultado =c/d;
        return resultado;
    }
}
```

### Resultado

```
Please enter first number: 10  
Please enter second number: 5  
the result of addition is 15  
the result of subtraction is 5  
the result of multiplication is 50  
the result of division is 2
```