

## GUÍA DE PRÁCTICA DE LABORATORIO Nro 05

EXPERIENCIA CURRICULAR: Metodología de Programación

DOCENTE: Mg. Ing. Ivan Petrlik Azabache / Ing. Eric Gustavo Coronel Castillo

SESIÓN: 5

### DESARROLLO DE LA PRÁCTICA

#### CAPACIDAD A TRABAJAR:

Construye Programas básicos usando clases, objetos, estructuras selectivas y repetitivas

#### CONCEPTOS:

### Estructuras de control en Java

Las estructuras de control determinan la secuencia de ejecución de las sentencias de un programa.

Las estructuras de control se dividen en tres categorías:

Secuencial

Condicional o Selectiva

Iterativa o Repetitiva.

#### 1. ESTRUCTURA SECUENCIAL

El orden en que se ejecutan por defecto las sentencias de un programa es secuencial. Esto significa que las sentencias se ejecutan en secuencia, una después de otra, en el orden en que aparecen escritas dentro del programa.

La estructura secuencial está formada por una sucesión de instrucciones que se ejecutan en orden una a continuación de la otra.

Cada una de las instrucciones están separadas por el carácter punto y coma (;).

Las instrucciones se suelen agrupar en bloques.

El bloque de sentencias se define por el carácter llave de apertura ({) para marcar el inicio del mismo, y el carácter llave de cierre (}) para marcar el final.

Ejemplo:

```
{  
instrucción 1;  
instrucción 2;  
instrucción 3;  
}
```

En Java si el bloque de sentencias está constituido por una única sentencia no es obligatorio el uso de las llaves de apertura y cierre ({ }), aunque sí recomendable.

Ejemplo de programa Java con estructura secuencial: Programa que lee dos números por teclado y los muestra por pantalla.

```
/* Programa que lea dos números por teclado y los muestre por pantalla.  
 */  
import java.util.*;  
public class Main {  
    public static void main(String[] args){  
        //declaración de variables  
        int n1, n2;  
        Scanner sc = new Scanner(System.in);  
        //leer el primer número  
        System.out.println("Introduce un número entero: ");  
        n1 = sc.nextInt();        //lee un entero por teclado  
        //leer el segundo número  
        System.out.println("Introduce otro número entero: ");  
        n2 = sc.nextInt();        //lee un entero por teclado  
  
        //mostrar resultado  
        System.out.println("Ha introducido los números: " + n1 + " y " + n2);  
    }  
}
```

Ejemplo de programa Java con estructura secuencial: Programa que lee dos números de tipo double por teclado y calcula y muestra por pantalla su suma, resta y multiplicación.

```
/*
```

```

* Programa que lee dos números de tipo double por teclado
* y muestra su suma, resta y multiplicación.
*/
import java.util.*;
public class Main {
    public static void main(String[] args){
        Scanner sc = new Scanner(System.in);
        double numero1, numero2;
        System.out.println("Introduce el primer número:");
        numero1 = sc.nextDouble();
        System.out.println("Introduce el segundo número:");
        numero2 = sc.nextDouble();
        System.out.println("Números introducido: " + numero1 + " " + numero2);
        System.out.println
            (numero1 + " + " + numero2 + " = " + (numero1+numero2));
        System.out.println
            (numero1 + " - " + numero2 + " = " + (numero1-numero2));
        System.out.println
            (numero1 + " * " + numero2 + " = " + numero1*numero2);
    }
}

```

Para modificar el orden de ejecución de las instrucciones de un programa Java se utilizan las estructuras condicionales y repetitivas.

## 2. ESTRUCTURA CONDICIONAL, ALTERNATIVA O SELECTIVA

La estructura condicional determina si se ejecutan unas instrucciones u otras según se cumpla o no una determinada condición.

En java la estructura condicional se implementa mediante:

- Instrucción if.
- Instrucción switch.
- Operador condicional ? :

### 2.1 INSTRUCCION if

Puede ser del tipo:

- Condicional simple: if
- Condicional doble: if ... else ...
- Condicional múltiple: if .. else if ..

La condición debe ser una **expresión booleana** es decir debe dar como resultado un valor booleano (**true ó false**).

**Condicional simple:** se evalúa la condición y si ésta se cumple se ejecuta una determinada acción o grupo de acciones. En caso contrario se saltan dicho grupo de acciones.

```

    if(expresión_booleana){
        instrucción 1
        instrucción 2
        .....
    }

```

Si el bloque de instrucciones tiene **una sola instrucción** no es necesario escribir las llaves { } aunque para evitar confusiones se recomienda escribir las llaves siempre.

Ejemplo de programa Java con estructura condicional: Programa que pide por teclado la nota obtenida por un alumno y muestra un mensaje si el alumno ha aprobado.

```

/*
 * Programa que pide una nota por teclado y muestra un mensaje si la nota es
 * mayor o igual que 5
 */
import java.util.*;
public class Ejemplo0If {
    public static void main( String[] args ){
        Scanner sc = new Scanner( System.in );
        System.out.print("Nota: ");
        int nota = sc.nextInt();
        if (nota >= 5 ){
            System.out.println("Enorabuena!!");
            System.out.println("Has aprobado");
        }
    }
}

```

**Condicional doble:** Se evalúa la condición y si ésta se cumple se ejecuta una determinada instrucción o grupo de instrucciones. Si no se cumple se ejecuta otra instrucción o grupo de instrucciones.

```

    if(expresión booleana){
        instrucciones 1
    }
    else{
        instrucciones 2
    }

```

Ejemplo de programa Java que contiene una estructura condicional doble: Programa que lee la nota de un alumno y muestra si el alumno ha aprobado o no.

```

/*
 * Programa que pide una nota por teclado y muestra si se ha aprobado o no
 */
import java.util.*;
public class Ejemplo0If {
    public static void main( String[] args ){
        Scanner sc = new Scanner( System.in );
        System.out.print("Nota: ");
        int nota = sc.nextInt();
        if (nota >= 5 ){
            System.out.println("Enorabuena!!");
            System.out.println("Has aprobado");
        }
        else
            System.out.println("Lo Siento, has suspendido");
    }
}

```

```
}  
}
```

Otro ejemplo de programa Java que contiene una estructura condicional doble: Calcular si un número es par. El programa lee un número por teclado y muestra un mensaje indicando si es par o impar.

```
/*  
 * programa que pide un número por teclado y calcula si es par o impar  
 */  
import java.util.*;  
public class EjemploIf {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        int num;  
        System.out.println("Introduzca numero: ");  
        num = sc.nextInt();  
        if ((num%2)==0)  
            System.out.println("PAR");  
        else  
            System.out.println("IMPAR");  
    }  
}
```

**Condicional múltiple:** Se obtiene anidando sentencias if ... else. Permite construir estructuras de selección más complejas.

```
if (expresion_booleana1)  
    instruccion1;  
else if (expresion_booleana2)  
    instruccion2;  
else  
    instruccion3;
```

Cada else se corresponde con el if más próximo que no haya sido emparejado.

Una vez que se ejecuta un bloque de instrucciones, la ejecución continúa en la siguiente instrucción que aparezca después de las sentencias if .. else anidadas.

Ejemplo de programa Java que contiene una estructura condicional múltiple: Programa que lee una hora (número entero) y muestra un mensaje según la hora introducida.

```
/*  
 * Programa que muestra un saludo distinto según la hora introducida  
 */  
import java.util.*;  
public class Ejemplo2If {  
    public static void main(String[] args) {  
        Scanner sc = new Scanner(System.in);  
        int hora;  
        System.out.println("Introduzca una hora (un valor entero): ");  
        hora = sc.nextInt();  
        if (hora >= 0 && hora < 12)  
            System.out.println("Buenos días");  
        else if (hora >= 12 && hora < 21)  
            System.out.println("Buenas tardes");  
        else if (hora >= 21 && hora < 24)  
            System.out.println("Buenas noches");  
        else  
            System.out.println("Hora no válida");  
    }  
}
```

```
}
```

Ejemplo de programa Java que contiene una estructura condicional múltiple: Programa que lee una nota (número entero entre 0 y 10) y muestra la calificación equivalente en forma de texto.

```
/*
 * programa que lee una nota y escribe la calificación correspondiente
 */
import java.util.*;
public class Ejemplo3If {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        double nota;
        System.out.println("Introduzca una nota entre 0 y 10: ");
        nota = sc.nextDouble();
        System.out.println("La calificación del alumno es ");
        if(nota < 0 || nota > 10)
            System.out.println("Nota no válida");
        else if(nota==10)
            System.out.println("Matrícula de Honor");
        else if (nota >= 9)
            System.out.println("Sobresaliente");
        else if (nota >= 7)
            System.out.println("Notable");
        else if (nota >= 6)
            System.out.println("Bien");
        else if (nota >= 5)
            System.out.println("Suficiente");
        else
            System.out.println("Suspenso");
    }
}
```

### Comparar String en Java

Para comparar el contenido de dos Strings en Java se usa el método **equals**:

```
if ((cadena1.equals(cadena2))
```

En caso de que una cadena coincida exactamente con una constante se puede usar ==

```
String nombre = "Lucas";
```

```
if (nombre == "Lucas")
```

Para comparar Strings en el orden alfabético se usa el método **compareTo**

```
if (cadena1.compareTo(cadena2) < 0) // cadena1 antes que cadena2
```

```
if (cadena1.compareTo(cadena2) > 0) // cadena1 después que cadena2
```

```
if (cadena1.compareTo(cadena2) == 0) // cadena1 igual que cadena2
```

### 2.2 INSTRUCCION switch

Se utiliza para seleccionar una de entre múltiples alternativas.

La forma general de la instrucción switch en Java es la siguiente:

```
switch (expresión){
    case valor 1:
        instrucciones;
        break;
    case valor 2:
        instrucciones;
        break;
    . . .
    default:
```

instrucciones;

}

La instrucción switch se puede usar con datos de tipo byte, short, char e int. También con tipos enumerados y con las clases envolventes Character, Byte, Short e Integer. A partir de Java 7 también pueden usarse datos de tipo String en un switch.

Funcionamiento de la instrucción switch:

Se evalúa la expresión y salta al case cuya constante coincida con el valor de la expresión. Se ejecutan las instrucciones que siguen al case seleccionado hasta que se encuentra un break o hasta el final del switch. El break produce un salto a la siguiente instrucción a continuación del switch.

Si ninguno de estos casos se cumple se ejecuta el bloque default (si existe). No es obligatorio que exista un bloque default y no tiene por qué ponerse siempre al final, aunque es lo habitual.

Ejemplo de programa Java que contiene una instrucción switch: Programa que lee por teclado un mes (número entero) y muestra el nombre del mes.

```
/*
 * Programa que pide un número de mes y muestra el nombre correspondiente
 */
import java.util.*;
public class Ejemplo0Switch {
    public static void main(String[] args) {
        int mes;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un numero de mes: ");
        mes = sc.nextInt();
        switch (mes)
        {
            case 1: System.out.println("ENERO");
                    break;
            case 2: System.out.println("FEBRERO");
                    break;
            case 3: System.out.println("MARZO");
                    break;
            case 4: System.out.println("ABRIL");
                    break;
            case 5: System.out.println("MAYO");
                    break;
            case 6: System.out.println("JUNIO");
                    break;
            case 7: System.out.println("JULIO");
                    break;
            case 8: System.out.println("AGOSTO");
                    break;
            case 9: System.out.println("SEPTIEMBRE");
                    break;
            case 10: System.out.println("OCTUBRE");
                    break;
            case 11: System.out.println("NOVIEMBRE");
                    break;
            case 12: System.out.println("DICIEMBRE");
                    break;
            default : System.out.println("Mes no válido");
        }
    }
}
```

### 3. ESTRUCTURA ITERATIVA O REPETITIVA

Permiten ejecutar de forma repetida un bloque específico de instrucciones.

Las instrucciones se repiten mientras o hasta que se cumpla una determinada condición. Esta condición se conoce como **condición de salida**.

Tipos de estructuras repetitivas:

ciclo while

ciclo do – while

ciclo for

#### 3.1 CICLO WHILE

Las instrucciones se repiten mientras la condición sea cierta. La condición **se comprueba al principio** del bucle por lo que las acciones se pueden ejecutar **0 ó más veces**.

La ejecución de un bucle while sigue los siguientes pasos:

1. Se evalúa la condición.

Si el resultado es false las instrucciones no se ejecutan y el programa sigue ejecutándose por la siguiente instrucción a continuación del while.

Si el resultado es true se ejecutan las instrucciones y se vuelve al paso 1

Ejemplo de programa Java que contiene una instrucción while:

Programa que lee números por teclado. La lectura acaba cuando el número introducido sea negativo. El programa calcula y muestra la suma de los números leídos.

```
/*
 * Programa que lee números hasta que se lee un negativo y muestra la
 * suma de los números leídos
 */
import java.util.*;
public class Ejemplo1While {
    public static void main(String[] args) {
        int suma = 0, num;
        Scanner sc = new Scanner(System.in);
        System.out.print("Introduzca un número: ");
        num = sc.nextInt();
        while (num >= 0){
            suma = suma + num;
            System.out.print("Introduzca un número: ");
            num = sc.nextInt();
        }
        System.out.println("La suma es: " + suma );
    }
}
```

Ejemplo de programa Java que contiene una instrucción while:

Programa que lee un número entero N y muestra N asteriscos.

```
/*
 * programa que lee un número n y muestra n asteriscos
 */
import java.util.*;
public class Ejemplo2While {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        int n, contador = 0;
        System.out.print("Introduce un número: ");
        n = sc.nextInt();
        while (contador < n){
            System.out.println(" * ");
        }
    }
}
```



```

        contador++;
    }
}
}

```

Ejemplo de programa Java con una instrucción while:

```

/*
 * programa que muestra una tabla de equivalencias entre
 * grados Fahrenheit y grados celsius
 */
public class Ejemplo3While {
    public static void main(String[] args) {
        final int VALOR_INICIAL = 10; // limite inf. tabla
        final int VALOR_FINAL = 100; // limite sup. tabla
        final int PASO = 10 ; // incremento
        int fahrenheit;
        double celsius;
        fahrenheit = VALOR_INICIAL;
        System.out.printf("Fahrenheit \t Celsius \n");
        while (fahrenheit <= VALOR_FINAL ){
            celsius = 5*(fahrenheit - 32)/9.0;
            System.out.printf("%7d \t %8.3f \n", fahrenheit, celsius);
            fahrenheit += PASO;
        }
    }
}

```

### 3.2 CICLO DO - WHILE

Las instrucciones se ejecutan mientras la condición sea cierta.

La condición **se comprueba al final** del bucle por lo que el bloque de instrucciones se ejecutarán **al menos una vez**. Esta es la diferencia fundamental con la instrucción while. Las instrucciones de un bucle while es posible que no se ejecuten si la condición inicialmente es falsa.

La ejecución de un bucle do - while sigue los siguientes pasos:

1. se ejecutan las instrucciones a partir de do{

2. se evalúa la condición.

3. si el resultado es false el programa sigue ejecutándose por la siguiente instrucción a continuación del while.

4. si el resultado es true se vuelve al paso 1

Ejemplo de programa Java que contiene una instrucción do while:

Programa que lee un número entero N. El número debe ser menor que 100.

```

/*
 * Programa que obliga al usuario a introducir un número menor que 100
 */
import java.util.*;
public class Ejemplo1DoWhile {
    public static void main(String[] args) {
        int valor;
        Scanner in = new Scanner( System.in );
        do {
            System.out.print("Escribe un entero < 100: ");
            valor = in.nextInt();
        }while (valor >= 100);
        System.out.println("Ha introducido: " + valor);
    }
}

```

Ejemplo de programa Java con una instrucción do while:

```
/*
 * Programa que lee un número entre 1 y 10 ambos incluidos
 */
import java.util.*;
public class Ejemplo2DoWhile {
    public static void main(String[] args) {
        int n;
        Scanner sc = new Scanner( System.in );
        do {
            System.out.print("Escribe un número entre 1 y 10: ");
            n = sc.nextInt();
        }while (n<1 || n >10);
        System.out.println("Ha introducido: " + n);
    }
}
```

### 3.3 CICLO FOR

Hace que una instrucción o bloque de instrucciones se repitan un **número determinado de veces mientras se cumpla la condición**.

La estructura general de una instrucción for en Java es la siguiente:

```
for(inicialización; condición; incremento/decremento){
instrucción 1;
.....
instrucción N;
}
```

A continuación de la palabra for y entre paréntesis debe haber siempre **tres zonas separadas por punto y coma**:

zona de inicialización.

zona de condición

zona de incremento ó decremento.

Si en alguna ocasión no es necesario escribir alguna de ellas se pueden dejar en blanco, pero los dos punto y coma deben aparecer.

**Inicialización** es la parte en la que la variable o variables de control del bucle toman su valor inicial. Puede haber una o más instrucciones en la inicialización, separadas por comas. La inicialización se realiza solo una vez.

**Condición** es una expresión booleana que hace que se ejecute la sentencia o bloque de sentencias mientras que dicha expresión sea cierta. Generalmente en la condición se compara la variable de control con un valor límite.

**Incremento/decremento** es una expresión que decrementa o incrementa la variable de control del bucle.

La ejecución de un bucle for sigue los siguientes pasos:

1. Se inicializa la variable o variables de control (inicialización)
2. Se evalúa la condición.
3. Si la condición es cierta se ejecutan las instrucciones. Si es falsa, finaliza la ejecución del bucle y continúa el programa en la siguiente instrucción después del for.
4. Se actualiza la variable o variables de control (incremento/decremento)
5. Se vuelve al punto 2.

Ejemplo de programa Java que contiene una instrucción for:

```
/*
 * programa que muestra los números del 1 al 10
 */
public class Ejemplo0For {
    public static void main(String[] args) {
        int i;
        for(i=1; i<=10;i++)
            System.out.println(i + " ");
    }
}
```

```
}  
}
```

La instrucción for del ejemplo anterior la podemos interpretar así:

Asigna a i el valor inicial 1, mientras que i sea menor o igual a 10 muestra i + " ", a continuación incrementa el valor de i y comprueba de nuevo la condición.

Ejemplo de programa Java con una instrucción for:

```
/*  
 * programa que muestra los números del 10 al 1  
 */  
public class Ejemplo2For {  
    public static void main(String[] args) {  
        int i;  
        for(i=10; i>0;i--)  
            System.out.println(i + " ");  
    }  
}
```

Ejemplo de programa Java con una instrucción for:

```
/*  
 * programa que muestra una tabla de equivalencias entre  
 * grados Fahrenheit y grados celsius  
 */  
public class Ejemplo1For {  
    public static void main(String[] args) {  
        final int VALOR_INICIAL = 10; // limite inf. tabla  
        final int VALOR_FINAL = 100; // limite sup. tabla  
        final int PASO = 10 ; // incremento  
        int fahrenheit;  
        double celsius;  
        fahrenheit = VALOR_INICIAL;  
        System.out.printf("Fahrenheit \t Celsius \n");  
        for (fahrenheit = VALOR_INICIAL; fahrenheit <= VALOR_FINAL;  
            fahrenheit+= PASO) {  
            celsius = 5*(fahrenheit - 32)/9.0;  
            System.out.printf("%7d \t %8.3f \n", fahrenheit, celsius);  
        }  
    }  
}
```

En las zonas de inicialización e incremento/decremento puede aparecer más de una variable. En ese caso deben ir separadas por comas.

Ejemplo:

```
/*  
 * programa que muestra el valor de a, b y su suma mientras que la suma de  
 * ambas es menor de 10. En cada iteración el valor de a se incrementa en  
 * 1 unidad y el de b en 2  
 */  
public class Ejemplo3For {  
    public static void main(String[] args) {  
        int a, b;  
        for(a = 1, b = 1; a + b < 10; a++, b+=2){  
            System.out.println("a = " + a + " b = " + b + " a + b = " + (a+b));  
        }  
    }  
}
```

La salida de este programa es:

a = 1 b = 1 a + b = 2

$a = 2$   $b = 3$   $a + b = 5$

$a = 3$   $b = 5$   $a + b = 8$

Aunque la instrucción repetitiva for, al igual que las instrucciones while y do- while, se puede utilizar para realizar repeticiones cuando no se sabe a priori el número de pasadas por el bucle, esta instrucción es especialmente indicada para bucles donde se conozca el número de pasadas. Como regla práctica podríamos decir que las instrucciones while y do-while se utilizan generalmente cuando no se conoce a priori el número de pasadas, y la instrucción for se utiliza generalmente cuando sí se conoce el número de pasadas.

Se ha de tener cuidado con escribir el punto y coma (;) después del paréntesis final del bucle for. Un bucle for generalmente no lleva punto y coma final.

Por ejemplo el bucle:

```
int i;  
for (i = 1; i <= 10; i++);  
{  
    System.out.println("Elementos de Programación");  
}
```

no visualiza la frase "Elementos de Programación" 10 veces, ni produce un mensaje de error por parte del compilador.

En realidad lo que sucede es que se visualiza una vez la frase "Elementos de Programación", ya que aquí la sentencia for es una sentencia vacía al terminar con un punto y coma (;).

La sentencia for en este caso hace que i empiece en 1 y acabe en 11 y tras esas iteraciones, se ejecuta la sentencia

```
System.out.println("Elementos de Programación");
```

#### **MATERIAL Y/O EQUIPO A UTILIZAR:**

- ☐ Computadora personal
- ☐ Programa JAVA Netbeans instalado
- ☐ Cuaderno de clases, donde están los ejercicios resueltos en clase

## EJERCICIO #01

Desarrollar un programa en java que me permita simular el movimiento de un almacén de bolsas de cemento.

Al comenzar la ejecución del programa, inicializar el stock de bolsas de cemento en 2000 bolsas y la capacidad del almacén en 3500 bolsas.

Luego, el programa permitirá efectuar operaciones de depósito y de retiro, mostrando en todo momento:

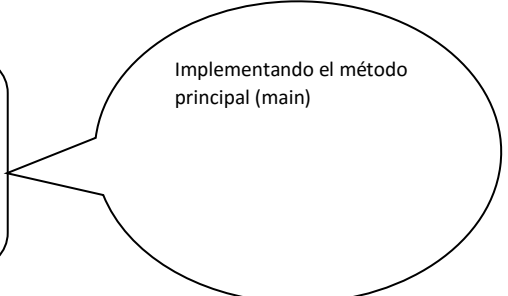
- El stock actual del almacén
- La capacidad libre del almacén
- El numero de depósitos y de retiros efectuados
- La cantidad total de bolsas depositadas y retiradas
- La cantidad máxima de bolsas retiradas y depositadas
- La cantidad mínima de bolsas retiradas y depositadas.

### Solución:

- a) Primeramente tenemos que crear un programa en java de nombre **Ejercicio1\_1** que implemente el método principal ( main )

```
public class Ejercicio1_1
{
    public static void main(String[] args)
    {

    }
}
```

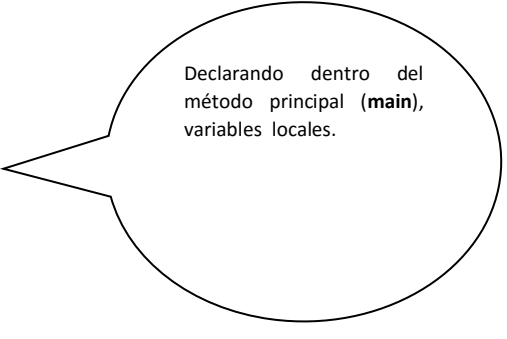


Implementando el método principal (main)

- b) Ahora vamos a declarar dentro del método principal (**main**) tres variables( op, cantidad , seguir,stock,capacidad ) de tipo de dato **int** , una variable (cantCad,opCad)de tipo de dato **String**.

```
public class Ejercicio1_1
{
    public static void main(String[] args)
    {
        int op=0 , cantidad=0 , seguir=0;
        String cantCad,opCad;
        int stock =2000 , capacidad =3500;

    }
}
```



Declarando dentro del método principal (main), variables locales.

- c) En el programa importamos un paquete javax.swing y además implementamos una estructura repetitiva (do - while) que me permite que el programa siga funcionando en el momento que el usuario decida.

```
import javax.swing.*;
public class Ejercicio1_1
{
    public static void main(String[] args)
    {
        int op=0 , cantidad=0 , seguir=0;
        String cantCad,opCad;
        int stock =2000 , capacidad =3500;

        do
        {
            // COLOCAR MAS ADELANTE EL CODIGO REQUERIDO

            seguir= JOptionPane.showConfirmDialog(null,"¿Deseas continuar?");
        }
        while(seguir!=1);
    }
}
```

1) Importando la librería Javax.swing

2) Implementando la estructura repetitiva do-while

- d) Dentro de la estructura repetitiva, agregamos líneas de código que me permitan ingresar una opción, eligiendo el depósito o el retiro.

Esta opción después se tiene que convertir a un valor entero.

```
import javax.swing.*;
public class Ejercicio1_1
{
    public static void main(String[] args)
    {
        int op=0 , cantidad=0 , seguir=0;
        String cantCad,opCad;
        int stock =2000 , capacidad =3500;

        do
        {
            opCad=JOptionPane.showInputDialog(null,"Elija la Operacion\n"
                                                + "DEPOSITAR [ 1 ]\n"
                                                + "RETIRAR [ 2 ]"
                                                , "",2);

            op=Integer.parseInt(opCad);

            seguir= JOptionPane.showConfirmDialog(null,"¿Deseas continuar?");
        }
        while(seguir!=1);
    }
}
```

Línea de código que me permite digitar una opción (**opcionCad**) entre el depósito y el retiro, luego se convierte la opción a un número entero (**op**)

- e) A continuación agregamos una serie de líneas de códigos que me permita ingresar por teclado la cantidad (cantCad) , a través de una caja de dialogo.

La cantidad (cantCad) se tendrá que convertir en un numero entero, donde será almacenado en una variable cuyo identificador es cantidad.

Luego tenemos que implementar una condicional múltiple (switch), que lea la opción que hemos escogido.

```
import javax.swing.*;
public class Ejercicio1_1
{
    public static void main(String[] args)
    {
        int op=0 , cantidad=0 , seguir=0;
        String cantCad,opCad;
        int stock =2000 , capacidad =3500;
        do
        {
            opCad=JOptionPane.showInputDialog(null,"Elija la Operacion\n"
                                                + "DEPOSITAR [ 1 ]\n"
                                                + "RETIRAR [ 2 ]"
                                                , "",2);

            op=Integer.parseInt(opCad);
            cantCad=JOptionPane.showInputDialog(null,"Ingrese la cantidad", "",2);
            cantidad=Integer.parseInt(cantCad);

            switch(op)
            {
                case 1 :
                {

                    break;
                }

                case 2 :
                {

                    break;
                }
            }

            JOptionPane.showMessageDialog(null,"STOCK ACTUAL DEL ALMACEN: "+stock);

            seguir= JOptionPane.showConfirmDialog(null,"¿Deseas continuar?");
        }
        while(seguir !=1);
    }
}
```

**1)** Línea de código que me permite digitar una cantidad (cantCad), luego se convierte la variable a un número entero (cantidad)

**2)** Implementando una estructura condicional múltiple con la finalidad de realizar las operaciones de depósito y retiro

**3)** Línea de código que me permite mostrar por pantalla el número de stock

- f) Los respectivos casos de la condicional múltiple(switch), se tiene que colocar condicionales dobles con la finalidad de validar la cantidad ingresada en el cálculo del stock en las diferentes opciones de retiro y deposito.

```
import javax.swing.*;
public class Ejercicio1_1
{
    public static void main(String[] args)
    {
        int op=0 , cantidad=0 , seguir=0;
        String cantCad,opCad;
        int stock =2000 , capacidad =3500;
        do
        {
            opCad=JOptionPane.showInputDialog(null,"Elija la Operacion\n"
                                                + "DEPOSITAR [ 1 ]\n"
                                                + "RETIRAR [ 2 ]"
                                                , "",2);

            op=Integer.parseInt(opCad);
            cantCad=JOptionPane.showInputDialog(null,"Ingrese la cantidad", "",2);
            cantidad=Integer.parseInt(cantCad);
            switch(op)
            {
                case 1 :
                {
                    if(cantidad<= capacidad -stock)
                    {
                        stock=stock +cantidad ;
                    }
                    else
                    {
                        JOptionPane.showMessageDialog(null,"ESPACIO LIBRE INSUFICIENTE");
                    }
                    break;
                }

                if(cantidad<=stock)
                {
                    stock=stock -cantidad ;
                }
                else
                {
                    JOptionPane.showMessageDialog(null,"STOCK INSUFICIENTE");
                }
                break;
            }

            JOptionPane.showMessageDialog(null,"STOCK ACTUAL DEL ALMACEN: "+stock);
            seguir= JOptionPane.showConfirmDialog(null,"¿Deseas continuar?");
        }
        while (seguir!=1);
    }
}
```

**1)** Implemen  
tando  
dentro  
del case 1  
la  
condicion  
al doble  
con la  
finalidad  
de validar  
la

**2)** Implemen  
tando  
dentro  
del case  
2 la  
condicion  
al doble  
con la  
finalidad  
de



g) Ahora declaramos en la clase una serie de variables de clase y además dentro de los respectivos case de la condicional múltiple(switch) implementamos 2 contadores que me permiten calcular la cantidad de depósitos efectuados y la cantidad de retiros efectuados.

```
import javax.swing.*;
public class Ejercicio1_1
{
    public static void main(String[] args)
    {
        int op=0 , cantidad=0 , seguir=0;
        String cantCad,opCad;
        int stock =2000 , capacidad =3500;
        int cantdepositosefectua=0,cantretirosefectua=0;
        do
        {
            opCad=JOptionPane.showInputDialog(null,"Elija la Operacion\n"
                                                + "DEPOSITAR [ 1 ]\n"
                                                + "RETIRAR [ 2 ]"
                                                + ":",2);

            op=Integer.parseInt(opCad);
            cantCad=JOptionPane.showInputDialog(null,"Ingrese la cantidad");
            switch(op)
            {
                case 1 :
                {
                    if(cantidad<= capacidad -stock)
                    {
                        stock=stock +cantidad ;
                        cantdepositosefectua=cantdepositosefectua+1 ;
                    }
                    else
                    {
                        JOptionPane.showMessageDialog(null,"ESPACIO LIBRE");
                        break;
                    }
                }
                case 2 :
                {
                    if(cantidad<=stock)
                    {
                        stock=stock -cantidad ;
                        cantretirosefectua=cantretirosefectua+1;
                    }
                    else
                    {
                        JOptionPane.showMessageDialog(null,"STOCK INSUFICIENTE");
                    }
                }
            }

            JOptionPane.showMessageDialog(null,"STOCK ACTUAL DEL ALMACEN: " +stock);
            JOptionPane.showMessageDialog(null,"NUMERO DE DEPOSITOS EFECTUADOS : " +cantdepositosefectua);
            JOptionPane.showMessageDialog(null,"NUMERO DE RETIROS EFECTUADOS : " +cantretirosefectua);

            seguir= JOptionPane.showConfirmDialog(null,"¿Deseas continuar?");
        }
        while(seguir!=1);
    }
}
```

**1) Declarando las variables**

**2) Implementando un contador de la cantidad del depósito efectuado**

**3) Implementando un contador de la cantidad del retiro efectuado**

**4) Mostrando por pantalla el número de depósitos efectuados y el número de retiros efectuados**

h) En la clase declaramos tres variables enteras (maximodeposito ,minimodeposito ,maximoretiro ,minimoretiro), además en el case 1 de la condicional múltiple(switch) implementamos dentro de la condicional doble un código que me permite calcular el máximo y mínimo deposito.

```
import javax.swing.*;
public class Ejercicio1_1
{
    public static void main(String[] args)
    {
        int op=0 , cantidad=0 , seguir=0;
        String cantCad,opCad;
        int stock =2000 , capacidad =3500;
        int cantdepositosefectua=0,cantretirosefectua=0;

        int maximodeposito=0 , minimodeposito=0 , maximoretiro =0, minimoretiro=0;
        do
        {
            opCad=JOptionPane.showInputDialog(null,"Elija la Operacion\n"
                                                + "DEPOSITAR [ 1 ]\n"
                                                + "RETIRAR [ 2 ]"
                                                , "",2);

            op=Integer.parseInt(opCad);
            cantCad=JOptionPane.showInputDialog(null,"Ingrese la cantidad", "",2);
            cantidad=Integer.parseInt(cantCad);
            switch(op)
            {
                case 1 :
                {
                    if(cantidad<= capacidad -stock)
                    {
                        stock=stock +cantidad ;
                        cantdepositosefectua=cantdepositosefectua+1 ;

                        if(cantdepositosefectua==1)
                        {
                            maximodeposito=cantidad;
                            minimodeposito=cantidad;
                        }
                        else
                        {
                            if(cantidad>maximodeposito)
                                maximodeposito=cantidad;
                            if(cantidad<minimodeposito)
                                minimodeposito=cantidad;
                        }
                    }
                    else
                    case 2 :
                    {
                        if(cantidad<=stock)
                        {
                            stock=stock -cantidad ;
                            cantretirosefectua=cantretirosefectua+1;
                        }
                        else
                        {
                            JOptionPane.showMessageDialog(null,"STOCK INSUFICIENTE");
                        }
                        break;
                    }
                }
            }
            JOptionPane.showMessageDialog(null,"STOCK ACTUAL DEL ALMACEN: " +stock);
            JOptionPane.showMessageDialog(null,"NUMERO DE DEPOSITOS EFECTUADOS : " +cantdepositosefectua);
            JOptionPane.showMessageDialog(null,"NUMERO DE RETIROS EFECTUADOS : " +cantretirosefectua);
            JOptionPane.showMessageDialog(null,"CANTIDAD MAXIMA DESPOSITADA : " +maximodeposito);
            JOptionPane.showMessageDialog(null,"CANTIDAD MINIMA DESPOSITADA : " +minimodeposito);

            seguir= JOptionPane.showConfirmDialog(null,"¿Deseas continuar?");
        } while(seguir!=1);
    }
}
```

1) Declarando variables enteras

2) Aquí hemos logrado implementar este código que me permite en primer lugar inicializar las variables maximodeposito y minimodesposito, cuando la cantidad de depósitos es igual a uno.

Además calculamos el máximo y mínimo deposito

3) Mostramos por pantalla la cantidad máxima depositada y cantidad mínima depositada

- h) Ahora dentro de la condicional múltiple en su respectivo case 2, implementamos dentro de la condicional doble un código que me permite calcular el máximo y mínimo retiro.

```
import javax.swing.*;

public class Ejercicio1_1
{
    public static void main(String[] args)
    {
        int op=0 , cantidad=0 , seguir=0;
        String cantCad,opCad;
        int stock =2000 , capacidad =3500;
        int cantdepositosefectua=0,cantretirosefectua=0;

        int maximodeposito=0 , minimodeposito=0 , maximoretiro =0, minimoretiro=0;
        do
        {
            opCad=JOptionPane.showInputDialog(null,"Elija la Operacion\n"
                                                + "DEPOSITAR [ 1 ]\n"
                                                + "RETIRAR [ 2 ]"
                                                , "",2);

            op=Integer.parseInt(opCad);
            cantCad=JOptionPane.showInputDialog(null,"Ingrese la cantidad","",2);
            cantidad=Integer.parseInt(cantCad);
            switch(op)
            {
                case 1 :
                {
                    if(cantidad<= capacidad -stock)
                    {
                        stock=stock +cantidad ;
                        cantdepositosefectua=cantdepositosefectua+1 ;

                        if(cantdepositosefectua==1)
                        {
                            maximodeposito=cantidad;
                            minimodeposito=cantidad;
                        }
                        else
                        {
                            if(cantidad>maximodeposito)
                                maximodeposito=cantidad;
                            if(cantidad<minimodeposito)
                                minimodeposito=cantidad;
                        }
                    }
                }
                else
                {
                    JOptionPane.showMessageDialog(null,"ESPACIO LIBRE INSUFICIENTE");
                }
                break;
            }

            case 2 :
            {
                if(cantidad<=stock)
                {
                    stock=stock -cantidad ;
                    cantretirosefectua=cantretirosefectua+1;

                    if(cantretirosefectua==1)
                    {
                        maximoretiro=cantidad;
                        minimoretiro=cantidad;
                    }
                    else
                    {
                        if(cantidad>maximoretiro)
                            maximoretiro=cantidad;
                        if(cantidad<minimoretiro)
                            minimoretiro=cantidad;
                    }
                }
                else
                {
                    JOptionPane.showMessageDialog(null,"STOCK INSUFICIENTE");
                }
                break;
            }
        }
    }
}
```

**1)** Aquí hemos logrado implementar este código que me permite en primer lugar inicializar las variables maximoretiro y minimoretiro, cuando la cantidad de depósitos es igual a uno.

Además calculamos el máximo y mínimo retiro

```

JOptionPane.showMessageDialog(null,"STOCK ACTUAL DEL ALMACEN:      "+stock);

JOptionPane.showMessageDialog(null,"NUMERO DE DEPOSITOS EFECTUADOS :  "+cantdepositosefectua);

JOptionPane.showMessageDialog(null,"NUMERO DE RETIROS  EFECTUADOS :  "+cantretirosefectua);

JOptionPane.showMessageDialog(null,"CANTIDAD MAXIMA DESPOSITADA :  "+maximodeposito);

JOptionPane.showMessageDialog(null,"CANTIDAD MINIMA DESPOSITADA :  "+minimodeposito);

JOptionPane.showMessageDialog(null,"CANTIDAD MAXIMA RETIRADA :      "+maximoretiro);

JOptionPane.showMessageDialog(null,"CANTIDAD MINIMA RETIRADA :      "+minimoretiro);

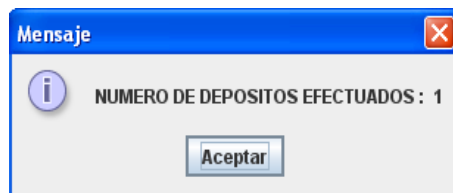
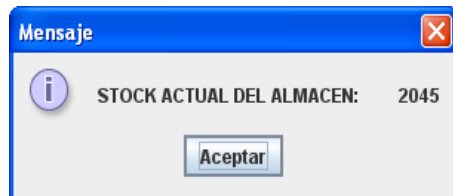
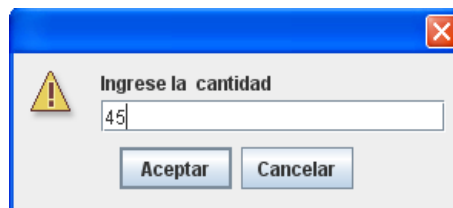
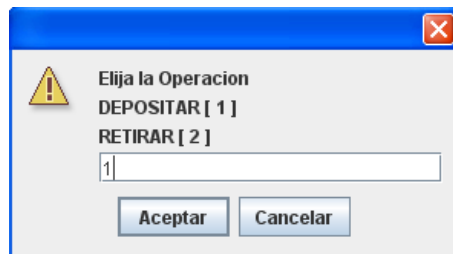
seguir= JOptionPane.showConfirmDialog(null,"¿Deseas continuar?");
}
while(seguir!=1);
}
}


```


2) Mostrando por pantalla la cantidad máxima retirada y la cantidad mínima retirada


## i) Compilando y ejecutando el programa


### SALIDA POR PANTALLA





**Mensaje** 


 NUMERO DE RETIROS EFECTUADOS : 0


**Mensaje** 


 CANTIDAD MAXIMA DESPOSITADA : 45


**Mensaje** 


 CANTIDAD MINIMA DESPOSITADA : 45


**Mensaje** 

 CANTIDAD MAXIMA RETIRADA : 0

**Mensaje** 

 CANTIDAD MINIMA RETIRADA : 0

**Seleccionar una opción** 

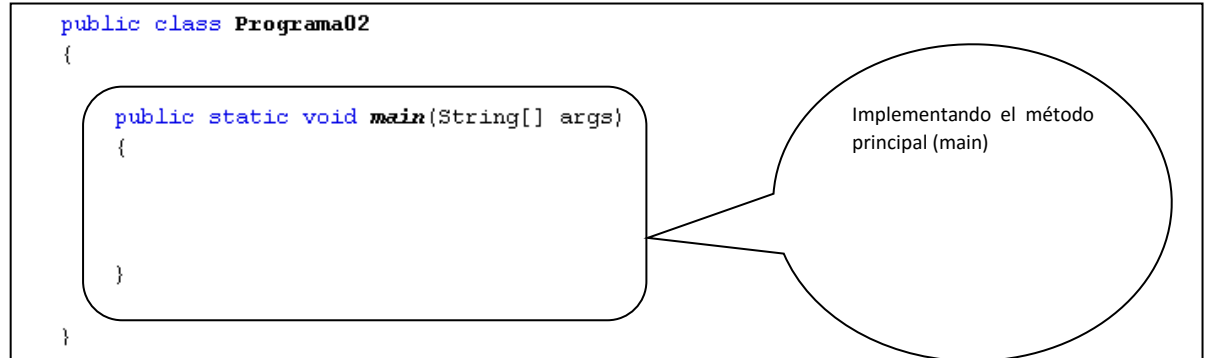
 ¿Deseas continuar?

## EJERCICIOS #02

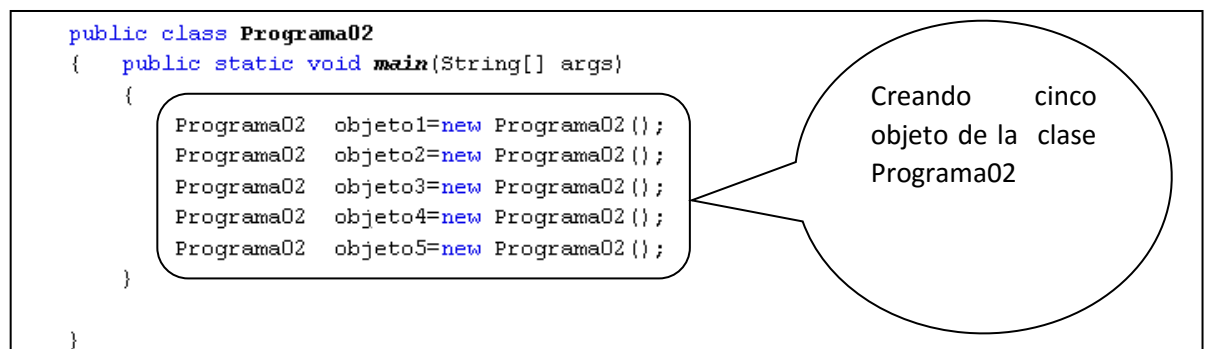
Desarrollar un programa en Java que me permita crear 5 objetos de una misma clase, que muestre un mensaje en el constructor.

### Solución:

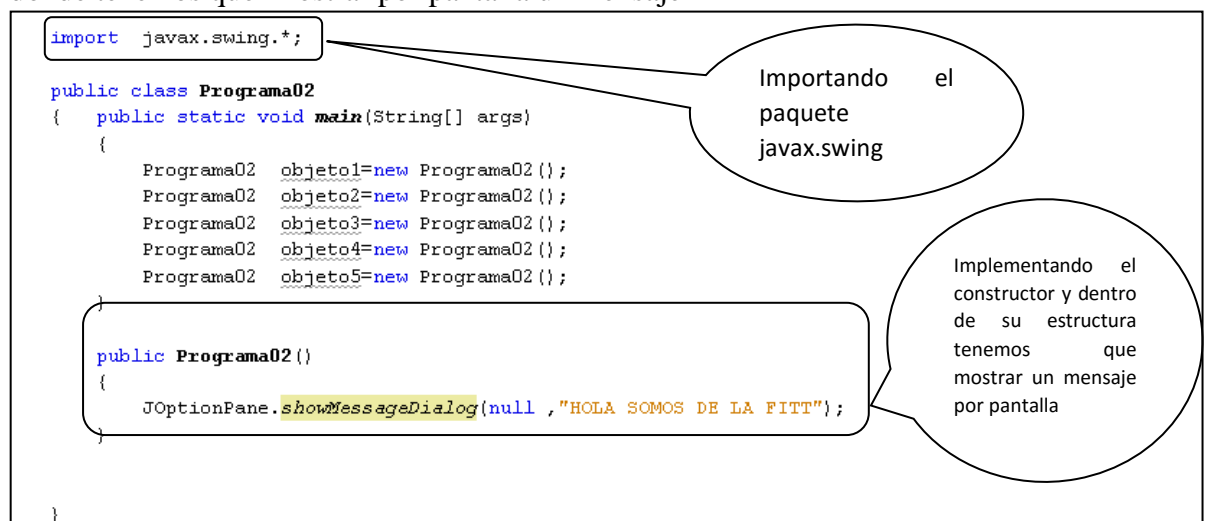
- a) Primeramente tenemos que crear un programa en java de nombre **Programa02** que implemente el método principal (main).



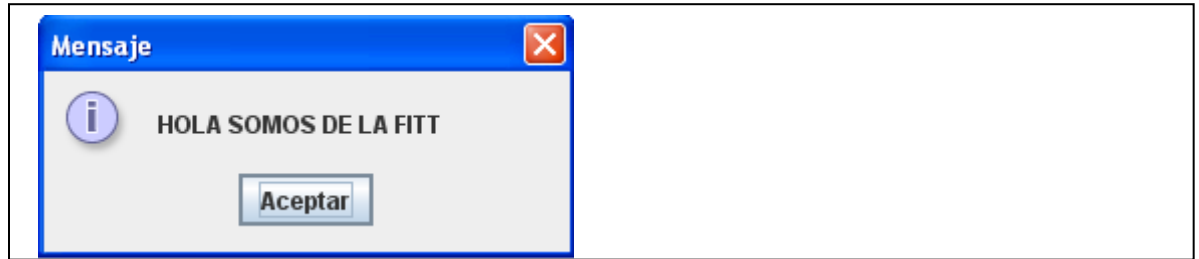
- b) Ahora dentro del método principal(main) creamos cinco objetos de la misma clase



- c) Luego importamos el paquete javax.swing e implementamos el constructor donde tenemos que mostrar por pantalla un mensaje



d) Compilando y ejecutando

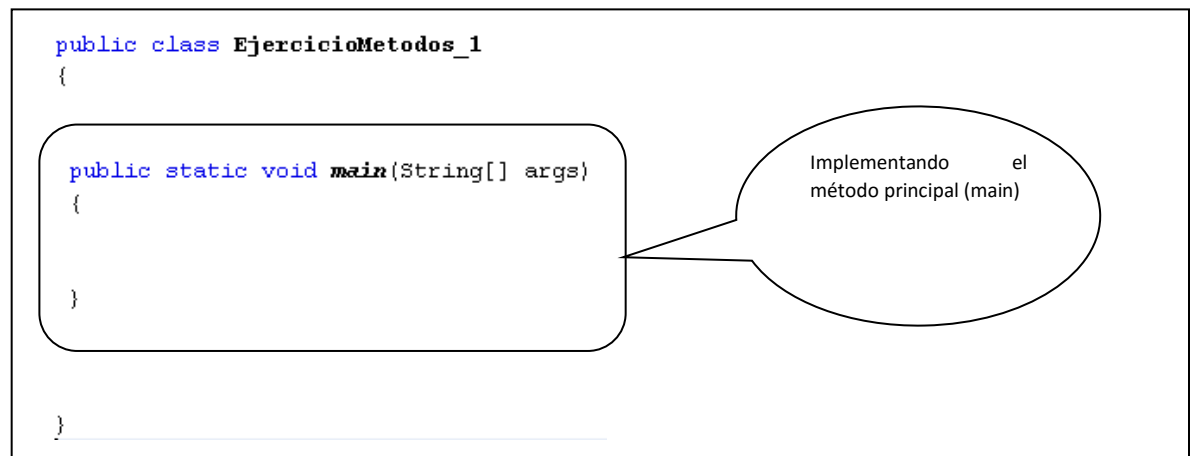


### EJERCICIO # 03

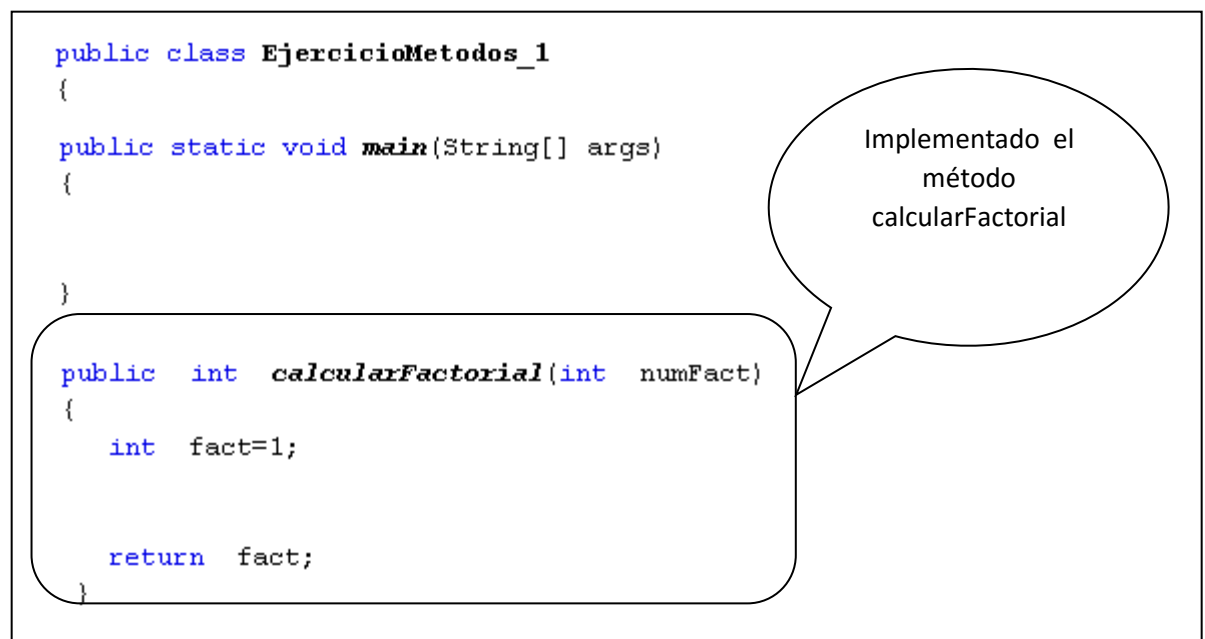
Desarrollar un programa en Java que me permita calcular la factorial de un número dado a través de un método estático (Método de Clase) con retorno de valor.

#### Solución:

- a) Primeramente tenemos que crear una clase de nombre EjercicioMetodos\_1 e implementar el método main()



- b) Después que hemos creado la clase e implementado el método main(), debemos ahora añadir dentro de la estructura de esta misma un método de nombre calcularFactorial que retorna valor y con un parámetro que representa el número del factorial.



c) Agregándole el contenido al método que retorna valor

```
import javax.swing.*;

public class EjercicioMetodos_1
{

    public static void main(String[] args)
    {

    }

    public int calcularFactorial(int numFact)
    {
        int fact=1;

        for(int i=1;i<=numFact;i++)
        {
            fact=fact+i;
        }

        return fact;
    }
}
```

Dentro de este método estamos primeramente declarando una variable fact de tipo de dato entero, inicializado en uno, Además implementamos un bucle repetitivo (for) que me permite realizar un conjunto de iteraciones, El numero de Iteraciones es igual al número del factorial(límite de las iteraciones)

Dentro del bucle repetitivo calculamos la factorial del número dado.

d) Ahora para invocar al método calcularFactorial tenemos que primeramente crear un objeto de la misma clase (EjercicioMetodos1) dentro del método main ().

```
import javax.swing.*;

public class EjercicioMetodos_1
{
    public static void main(String[] args)
    {
        int factorial=0;

        EjercicioMetodos_1 objEjerM1=new EjercicioMetodos_1();

        factorial = objEjerM1.calcularFactorial(5);

        System.out.println("EL FACTORIAL ES :"+factorial);

        JOptionPane.showMessageDialog(null,"EL FACTORIAL ES :"+factorial);
    }

    public static int calcularFactorial(int numFact)
    {
        int fact=1;

        for(int i=1;i<=numFact;i++)
        {
            fact=fact*i;
        }

        return fact;
    }
}
```

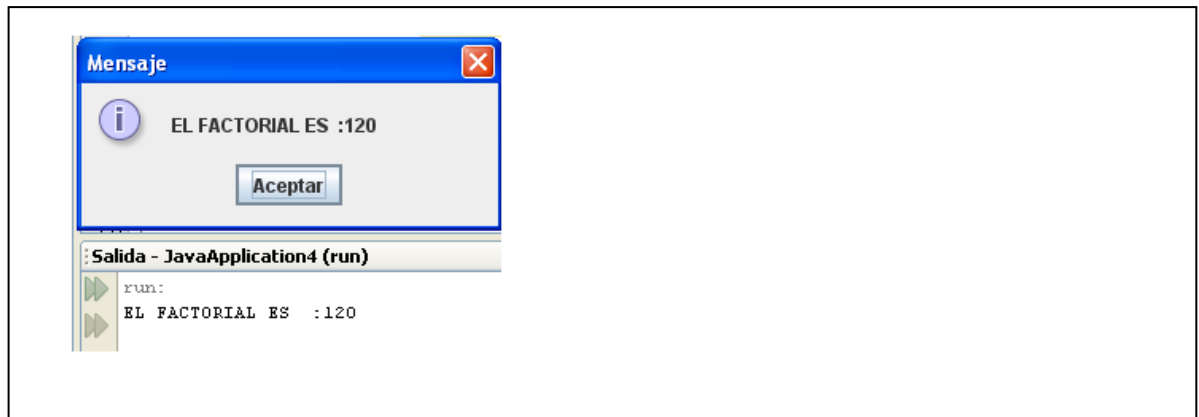
Para invocar al método **calcularFactorial**, primeramente tenemos que crear un objeto de la misma clase(**EjercicioMetodos1**).

El método para ser invocado se debe de anteponer el objeto al método de la siguiente manera:

**objEjerM1.calcularFactorial( 5 );**



- e) Ahora para invocar al método calcularFactorial tenemos que primeramente crear un objeto



#### EJERCICIO #04

Desarrollar un programa en Java que me permita crear una Método recursivo que acumule números consecutivos desde el 1 hasta el numero que usted elija como ultimo termino .

Si usted tiene los números  $1+2+3+4+5$  , significa que los numero acumulado empieza desde uno y termina en el 5 .

La acumulación se realiza recursivamente

El resultado de la acumulación los números es: 15

#### Solución:

- a) Primeramente tenemos que crear una Método que retorna valor , que acumule los números desde el uno hasta el término dado.

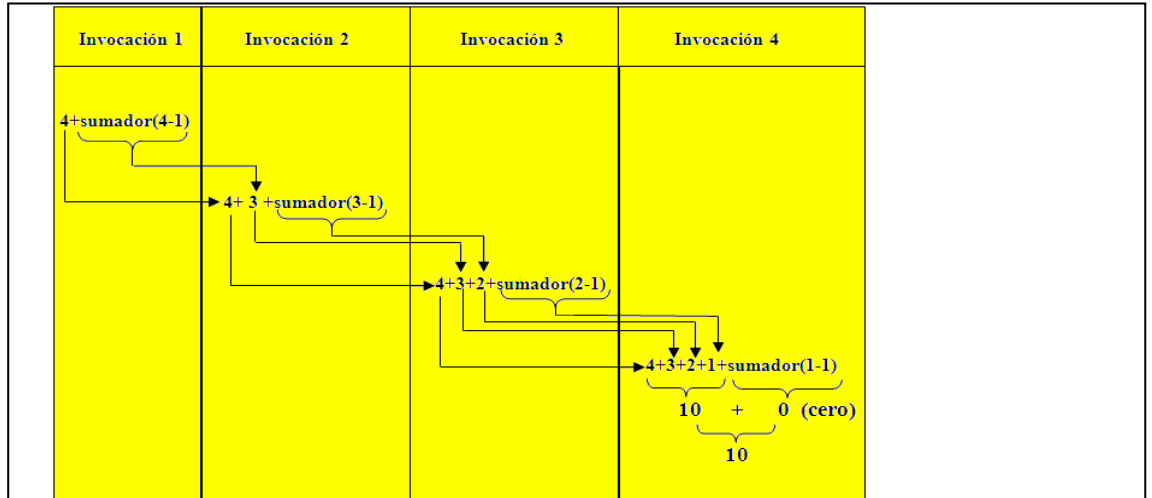
```
public int Sumador (int ultimotermينو)
{
    if(ultimotermينو==0)
    {
        return 0;
    }
    else
    {
        return ultimotermينو +Sumador(ultimotermينو-1);
    }
}
```

Este Método recursivo que se ha construido, Suponiendo que al invocarse se envía un parámetro de valor 4.

La función cuando se ejecuta recibe como parámetro el valor (4) que se ha enviado

La función empieza a acumular los valores recursivamente.

El funcionamiento de esta función recursiva es de la siguiente manera:



b) Ahora como sabemos el funcionamiento de este Método recursivo , lo implementaremos a la estructura de nuestro programa en Java.

```
import javax.swing.*;

public class EjerciciosMetodos_2
{
    public static void main(String[] args)
    {
        EjerciciosMetodos_2 objEjerM2=new EjerciciosMetodos_2();
        JOptionPane.showMessageDialog(null, objEjerM2.Sumador (4) );
    }

    public int Sumador (int ultimotermينو)
    {
        if(ultimotermينو==0)
        {
            return 0;
        }
        else
        {
            return  ultimotermينو +Sumador(ultimotermينو-1);
        }
    }
}
```

Primeramente creamos un objeto e Invocamos el metodo recursivo y enviamos un parámetro de

Implementando la función recursiva

c) Ejecutando el programa en Java ,donde tenemos la siguiente salidas de pantalla



### EJERCICIO #05

Desarrollar un programa en Java que permita clasificar el tipo de temperatura de un horno microondas, la temperatura se tiene que ingresar por teclado.

El horno microondas puede variar desde 0 hasta 100 grados centígrados y se clasifica de acuerdo a lo siguiente:

CLASIFICACION	RANGO
MUY ALTA	90°C y 100°C
ALTA	80°C y 89°C
NORMAL	40°C y 79°C
BAJA	0°C y 39°C

Utilizar en la implementación un método que no retorna valor

#### Solución:

a) Primeramente tenemos que crear un Método que no retorna valor, que determine la clasificación del tipo de temperatura que hemos ingresado por teclado.

```
public void DeterminarTipoTemperatura(int temperatura)
{
    if(temperatura>=0 && temperatura<=39)
    {
        JOptionPane.showMessageDialog(null,"BAJA");
    }
    else
    {
        if(temperatura>=40 && temperatura<=79)
        {
            JOptionPane.showMessageDialog(null,"NORMAL");
        }
        else
        {
            if(temperatura>=80 && temperatura<=89)
            {
                JOptionPane.showMessageDialog(null,"ALTA");
            }
            else
            {
                JOptionPane.showMessageDialog(null,"MUY ALTA");
            }
        }
    }
}
```

El Método que hemos construido no retorna valor, esto significa que el resultado no retorna al lugar donde es invocado.

El procedimiento tiene un parámetro que me permite la recepción de la variable temperatura.

El Método posee una sentencia que se ubica a la mano izquierda superior de nombre void.

El Método se le ha dado un nombre personalizado (flexible)

b) Ahora que hemos creado el Método, vamos a implementarlo en el programa e invocarlo para su respectiva ejecución

```
import javax.swing.*;

public class EjerciciosMetodos_3
{
    public static void main(String[] args)
    {
        EjerciciosMetodos_3 objEjerM3=new EjerciciosMetodos_3();
        objEjerM3.DeterminarTipoTemperatura( 50 );

        public void DeterminarTipoTemperatura(int temperatura)
        {
            if(temperatura>=0 && temperatura<=39)
            {
                JOptionPane.showMessageDialog(null,"BAJA");
            }
            else
            {
                if(temperatura>=40 && temperatura<=79)
                {
                    JOptionPane.showMessageDialog(null,"NORMAL");
                }
                else
                {
                    if(temperatura>=80 && temperatura<=89)
                    {
                        JOptionPane.showMessageDialog(null,"ALTA");
                    }
                    else
                    {
                        JOptionPane.showMessageDialog(null,"MUY ALTA");
                    }
                }
            }
        }
    }
}
```

Objeto creado  
y método  
invocado

Implementando  
el Método en el  
programa

## ***Bibliografía:***

- THOMAS WU C. Introducción a la programación orientada a objetos con Java. 1ª Edición. España. McGraw-Hill Interamericana de España. 2008. 214-324pp. ISBN: 978-0-07-352339-2
- LEOBARDO LOPEZ. Román. Metodología de la programación orientada a objetos. 1ª Edición. México. Alafomega grupo editor de México. 2006. 257-342pp ISBN: 970-15-1173-5
- HERBERT SHILDT. JAVA 2 v5.0. España. Ediciones Anaya multimedia.2005. 131-138pp, 834pp ISBN: 131-138-1865-3.

## ***REFERENCIAS EN INTERNET***

- <http://gcoronelc.blogspot.pe>
- <http://gcoronelc.blogspot.pe/2016/11/programando-pensando-en-servicios-parte.html>
- <http://gcoronelc.blogspot.pe/2016/11/prog-pensando-en-servicios-parte-2.html>
- <http://gcoronelc.blogspot.pe/2016/06/separata-java-orientado-objetos.html>
- <http://gcoronelc.blogspot.pe/p/java.html>
- <http://gcoronelc.blogspot.pe/2017/01/java-fundamentos-01-introduccion.html>
- <http://gcoronelc.blogspot.pe/2013/09/java-poo-leccion-01.html>
- <http://gcoronelc.blogspot.pe/2013/09/java-poo-leccion-02.html>