4. INSTRUCCIONES DE CONTROL

4.1 CONDICIONALES

4.1.1 Instrucción if

Una vez evaluada una condición determina que instrucción o instrucciones deben ejecutarse, dependiendo del resultado obtenido en la evaluación, el cual debe ser un booleano true/false. La instrucción ***if*** tiene diferentes estructuras sintácticas para su uso.

Estructura 1: Se usa cuando posterior a la evaluación de la condición se ejecuta una única instrucción.

```
if(condicion)
        Instuccion1;
```

Estructura 2: Se usa cuando posterior a la evaluación de la condición se ejecuta un único bloque de instruccioness.

```
if(condicion)
{
        Instruccion1;
        Instruccion2;
        Instruccion3;
        ……
        Instruccionn;
}
```

Estructura 3: Se usa cuando posterior a la evaluación de la condición y dependiendo del resultado de esta se ejecuta una u otra instrucción.

```
if(condicion)
        Instuccion1;
else
        Instruccion2;
```

Estructura 4: Se usa cuando posterior a la evaluación de la condición es posible ejecutar dos bloques de instrucciones.

```
if(condicion)
{
        Instuccion1;
        Instruccion2;
        ……..
        Instruccionn;
}
else
{
       Instruccion3;
        Instrucción4;
        ……..
        Instruccionq;
}
```

Estructura 5: Se usa cuando posterior a la evaluación de varias condiciones es posible ejecutar más de dos instrucciones por separado.

```
if(condicion1)
        Instruccion1;
else if(condicion2)
        Instruccion2;
else if(condicion3)
        Instruccion3;
else
        instruccionn;
```

Estructura 6: Se usa cuando posterior a la evaluación de varias condiciones es posible ejecutar más de dos bloques de instrucciones.

```
if(condicion1)
{
        Instruccion1;
        Instruccion2;
        ……..
        Instruccionn;
}
else if(condicion2)
{
        Instruccion3;
        Instruccion4;
        ……..
        Instruccionm;
}
else
{
        Instruccion5;
        Instruccion6;
        ……..
        Instruccionp;
}
```

Estructura 7: Una vez evaluada una condición a través de una instrucción ***if*(condicion)** es posible que una de las alternativas **(true/false)** a seguir sea la evaluación de otra condición.

```
if(condicion1)
{
        if(condicion2)
                Instruccion1;
        Instruccion2;
}
```

4.1.2 Operador ?

Es un operador que interactúa con tres operandos: Una **condición** que se evalúa, **Instruccion_t** que se ejecuta en caso que el operador retorne true y una **instruccion_f** que se ejecuta en caso que el operador retorne false.

```
condicion ? instruccion_t : instruccion_f;
```

Este operador se convierte en una forma de simplificar un operador if, cosa que podemos observar a continuación:

```
if (auxgaraje.equals("SI"))
      garaje = true;
else
      garaje = false;;
```

es equivalente a:

```
garaje = auxgaraje.equals("SI")? true : false;
```

4.1.3 Operador switch

El operador ***switch*** permite la simplificación de un operador ***if*** en el que se tenga que seguir la ejecución de diferentes bloques de instrucciones* una vez evaluada una condición.

```
switch(lectura)
{
        case 1 :
                instruccion1;
                instruccion2;
                break;
        case 2:
                instruccion3;
                instruccion4;
                break;
        default:
                instruccion5;
                instruccion6;
}
```

* También aplica para cuando se quiera seguir una única instrucción para cada caso.

La sentencia ***break*** que nos sirve para interrumpir la ejecución de un bloque iterativo en cualquier momento y el lenguaje Java permite esta posibilidad, no es muy recomendable ya que hace difícil la comprensión del código [7]. Dentro de la estructura de un ***swich*** la instrucción ***break*** permite separar cada uno de los casos, es decir, que una vez sea cumplida

la condición en un caso, no se siguen revisando los demás. La opción ***default*** se utiliza para ejecutar una serie de instrucciones una vez no se haya cumplido ningún caso.

El ejemplo que se da a continuación utiliza el operador ***switch*** para la construcción de una sencilla calculadora con las cuatro operaciones básicas (suma, resta, multiplicación y división).

```
import java.io.*;

public class Menu
{
	public static void main(String[]args)throws IOException
	{
		BufferedReader BufferedReader1 = new BufferedReader
		(new InputStreamReader(System.in));
		Calculadora Calculadora1 = new Calculadora();
		Calculadora1.leerTerminos(BufferedReader1);
		Calculadora1.hacerOperacion();
	}
}

class Calculadora
{
	double operando1;
	double operando2;
	char operador;
	double resultado;
	void leerTerminos(BufferedReader Lectura)throws
	IOException
	{
		System.out.println("--CALCULADORA--");
		System.out.print("Digite el primer término: ");
		System.out.flush();
		operando1 = Double.valueOf(Lectura.readLine()).
		doubleValue();
		System.out.println();
		System.out.print("Digite el segundo término: ");
		System.out.flush();
		operando2 = Double.valueOf(Lectura.readLine()).
		doubleValue();
		System.out.println();
		System.out println("=========================");
		System.out.println("SUMA              +       ");
```

```
            System.out.println("RESTA              -        ");
            System.out.println("MULTIPLICACION     *        ");
            System.out.println("DIVISION           /        ");
            System.out.println("==========================");
            System.out.print("  OPCION:  ");
            System.out.flush();
            operador = (char)System.in.read();
      }

      void hacerOperacion()
      {
            switch(operador)
            {
            case '+':
                  resultado = operando1 + operando2;
                  break;
            case '-':
                  resultado = operando1 - operando2;
                  break;
            case '*':
                  resultado = operando1 * operando2;
                  break;
            case '/':
                  resultado = operando1 / operando2;
                  break;
            default:
                  System.out.println("OPERACION NO VALIDA!");
            }
            System.out.println("RESULTADO = "+ resultado);
      }
}
```

4.2 BUCLES REPETITIVOS

En java, así como en otros lenguajes de programación, los bucles repetitivos permiten que un bloque de instrucciones sea ejecutado mientras se cumpla una condición.

4.2.1 Bucle for

En el bucle ***for*** se ejecutan una o varias instrucciones mientras se cumpla una condición que ha sido predeterminada dentro de su estructura. La estructura general de un bucle ***for*** es la siguiente:

```
for(inicializacion;condicion;incremento)
{
        instruccion_1;
        instruccion_2;
        ……..
        instruccion_n;
}
```

La inicialización corresponde al valor inicial que tomará la variable encargada de llevar el control del bucle. La condición es la expresión que se evaluará en cada iteración y determina el momento en que el bucle debe terminar. El incremento determina la variación de la variable de control cada vez que se ejecuta o itera el bucle.

El siguiente ejemplo imprime en una línea los números del 1 al 10.

```
public class Buclefor
{
      public static void main(String[]args)
      {
            for(int i = 1;i<=10;i++)
            {
                  System.out.print(i+" ");
            }
      }
}
```

Así como es posible que la variación sea de incremento, también se puede hacer que esta vaya en decremento:

```
public class BucleforDecremento
{
      public static void main(String[]argas)
      {
            int i;
            for(i=10;i>0;i--)
                  System.out.print(i+" ");
      }
}
```

4.2.1.1 Uso de la instrucción break en un blucle for

Java también permite utilizar la instrucción ***break*** para devolver el control del programa a la línea siguiente a un bucle ***for***. De esta forma se garantiza que no existan bucles infinitos. El siguiente ejemplo en el que el programa efectúa la lectura de caracteres desde el teclado y que termina cuando pulsamos el carácter 'z', ilustra el uso de la instrucción ***break***:

```
import java.io.*;
public class LeerCaracteres
{
      public static void main(String[]args)throws IOException
      {
            Entrada Entrada1 = new Entrada();
            Entrada1.leerdesdeteclado();
            System.out.println("FIN DE LA LECTURA!");
      }
}

class Entrada
{
      char caracter;
      void leerdesdeteclado()throws IOException
      {
            for(;;)
            {
                  System.out.println("Digite un caracter!");
                  caracter = (char)System.in.read();
                  if(caracter == 'z')
                        break;
            }
      }
}
```

4.2.2 Bucle while

En el bucle ***while*** las iteraciones se repiten mientras se cumpla una condición. La estructura general de un bucle ***while*** es la siguiente:

```
while(condicion)
{
        instruccion_1;
        instruccion_2;
        ……..
        instruccion_n;
}
```

Su construcción es más sencilla que la del bucle ***for***. La condición puede ser cualquier expresión o valor que pueda ser evaluado a través de un valor booleano. En el siguiente ejemplo se hace uso de un bucle ***while*** para construir una diagonal con los números desde el 10 al 1.

```
public class Diagonal
{
      public static void main(String[]args)
      {
            int i = 10;
            String espacio ="";
            while(i>0)
            {
                  System.out.println(espacio+ i);
                  espacio = espacio+" ";
                  i--;
            }
      }
}
```

4.2.3 Bucle do-while

A diferencia del bucle ***while***, el bucle ***do-while*** no hace la evaluación de la condición al inicio del bucle, sino que lo hace al final de este. La estructura general del bucle ***do-while*** es la siguiente:

```
do
{
        instruccion_1;
        instruccion_2;
        ……..
        instruccion_n;
}while(condicion);
```

El bucle ***do-while*** se repite hasta que la condición se hace falsa. Se puede observar su uso en el siguiente ejemplo, en el que una vez leído un número entero, se hace la sumatoria de cada uno de los números enteros desde el 1 hasta el número leído.

```
import java.io.*;
public class Ejemplodowhile
{
      public static void main(String[]args)throws IOException
      {
            BufferedReader BufferedReader1 = new BufferedReader
            (new InputStreamReader(System.in));
            Sumatoria Sumatoria1 = new Sumatoria();
            Sumatoria1.leerNumTerminos(BufferedReader1);
            Sumatoria1.hacerSumatoria();
            System.out.println("La sumatoria es:
            "+Sumatoria1.hacerSumatoria());
      }
}

class Sumatoria
{
      int total = 0;
      int numero;
      void leerNumTerminos(BufferedReader Lectura)throws
      IOException
      {
            System.out.println("Terminos de la sumatoria: ");
            System.out.flush();
            numero = Integer.valueOf(Lectura.readLine()).
            intValue();
      }
      int hacerSumatoria()
      {
            do
            {
                  total = total + numero;
                  numero--;
            }while(numero>0);
            return total;
      }
}
```

4.2.4 Uso de bucles anidados

En muchas ocasiones el programador debe incluir un bucle dentro de otro bucle. A este tipo de operación se conoce como anidación de bucles. Cuando esto se presenta, el compilador primero ejecuta el bucle interno, luego si sigue cumpliéndose la condición del bucle externo, entonces se vuelve a ejecutar el bucle interno hasta que su condición sea válida. El programa siguiente, que imprime la tabla que se presenta a continuación, contiene un bucle ***for*** anidado dentro de otro bucle ***for***:

```
1 2 3 4 5 6 7
2 3 4 5 6 7
3 4 5 6 7
4 5 6 7
5 6 7
6 7
7
```

```
import java.io.*;
public class MediaMatriz
{
      public static void main(String[]args)throws IOException
      {
            BufferedReader BufferedReader1 = new BufferedReader
            (new InputStreamReader(System.in));
            Tablaimpresa tablaimpresa1 = new
            Tablaimpresa();
            tablaimpresa1.leerNumero(BufferedReader1);
            tablaimpresa1.imprimirtabla();
      }
}

class Tablaimpresa
{
      int n;
      void leerNumero(BufferedReader Lectura)throws IOException
      {
            System.out.println("Digite un número entero: ");
            System.out.flush();
            n = Integer.valueOf(Lectura.readLine()).
            intValue();
      }
      void imprimirtabla()
      {
```

```
            int i, j;
            for(i=1;i<=n;i++)
            {
                  for(j=i;j<=n;j++)
                  {
                        System.out.print(j+" ");
                  }
                  System.out.println();
            }
      }
}
```

4.2.5 Instrucciones de salto

Java también proporciona instrucciones que le permiten transferir el control de un punto a otro del programa. Estas instrucciones son: ***break, continue y return***.

4.2.5.1 Instrucción break

El uso de la instrucción ***break*** tiene especial importancia en la separación de casos dentro de una instrucción ***switch***, en la que break permite separar cada uno de sus casos (Ver apartado 4.1.3) y como instrucción para forzar la salida de un bucle (Ver apartado 4.2.1.1) .

El siguiente ejemplo ilustra el uso de la instrucción break para forzar el rompimiento de un bucle ***for***. El bucle ***for*** perteneciente al método *imprimirOriginal( )* se ejecuta en su totalidad dando como resultado la impresión consecutiva de los números del 1 hasta el 50. El bucle ***for*** perteneciente al método *romperBucle( )*puede ser interrumpido al digitar un nuevo número que corresponde al valor en el cual se rompe el bucle y devuelve el control a la línea siguiente al bucle, aun cuando el límite superior del bucle sigue siendo 50.

```
import java.io.*;
public class InterrumpeBucles
{
      public static void main(String[]args) throws IOException
      {
            BufferedReader BufferedReader1 = new BufferedReader
            (new InputStreamReader(System.in));
            Ciclo CicloUno = new Ciclo();
```

```
            CicloUno.imprimirOriginal();
            CicloUno.romperBucle(BufferedReader1);
      }
}

class Ciclo
{
      int i,rompe;
      void imprimirOriginal()
      {
            for(i=1;i<=50;i++)
            {
                  System.out.println(i);
            }
            System.out.println("Fin del bucle");
      }
      void romperBucle(BufferedReader Lectura)throws IOException
      {
            System.out.println("Introduzca el número de
            rompimiento! ");rompe = Integer.valueOf
            (Lectura.readLine()).intValue();
            for(i=1;i<=50;i++)
            {
                  if(i==rompe+1)break;
                  System.out.println(i);
            }
            System.out.println("Fin del bucle");
      }
}
```

EJERCICIOS PROPUESTOS PARA ESTA UNIDAD

1. Elaborar un programa que permita leer un número entero **x** y lo eleve a la **y** potencia (x^y). Ejemplo: 3^4 = 3*3*3*3= 81

2. Elaborar un programa que genere la tabla de multiplicar para cualquiera de los números enteros entre 1 y 9. Debe contener la clase Tabla, que a su vez tendrá la variable número de tipo entero; para almacenar el número de la tabla que se quiere construir, y los métodos leerTabla( ) y construirTabla( ).

3. Elaborar un programa que genere las tablas de multiplicar para los números pares entre 1 y 20.

4. Cuál es la salida por pantalla del siguiente programa:

```
{public class Diagonal
{
      public static void main(String[]args)
      {
            int i = 10;
            String espacio ="";
            while(i>0)
            {
                  System.out.print(espacio+ i);
                  espacio = espacio+" ";
                  i--;
            }
      }
}
```

5. Elabore un programa en el que con los números del 0 al 9 y haciendo uso de un bucle while genere una figura en forma de V.

6. Haciendo uso de un bucle do-while, elabore un programa que permita calcula el factorial de un número.

7. Elabore un programa que calcule la suma de los múltiplos de tres comprendidos entre 0 y 100.

8. Elabore un programa que permita calcular la suma de los números primos comprendidos entre 0 y 100.