

## 2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### ESTRUCTURAS DE CONTROL

- Las estructuras de control determinan el comportamiento de un programa; permiten combinar instrucciones o sentencias individuales en una simple unidad lógica con un punto de entrada y otro de salida, se organizan en tres tipos que sirven para controlar el flujo de la ejecución: **secuencia, selección o decisión y repetición**. Además, ejecutan una sentencia simple o compuesta; esta última es un **conjunto de sentencias** encerradas **entre llaves** ({} ) que se utiliza para especificar un flujo secuencial; así se representa:

```
{  
sentencia 1;  
sentencia 2;  
...  
sentencia n;  
}
```



## 2.1. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. SELECCIÓN. SENTENCIA IF

- La **estructura de control de selección** principal es una sentencia **if**; la cual, tiene **dos alternativas** o formatos posibles, el más sencillo tiene la sintaxis siguiente:

*if (expresión) Acción*

- La sentencia **if** funciona de la siguiente manera: cuando se alcanza, se evalúa la siguiente expresión entre paréntesis; si expresión es verdadera se ejecuta *Acción*, en caso contrario no se efectúa y sigue la ejecución en la siguiente sentencia. *Acción* es una sentencia simple o compuesta.



## 2.1. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN. SENTENCIA IF

- Ejemplo: prueba de divisibilidad. Programa en el que se introducen dos números enteros y mediante una sentencia de selección se determina si son divisibles.

```
import java.util.Scanner;

public class EsDivisible{
    public static void main(String[] a) {
        int n, d;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca dos enteros:");
        n = entrada.nextInt();
        d = entrada.nextInt();
        if (n % d == 0)
            System.out.println(n + " es divisible por " + d);
    }
}
```



## 2.1. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. SELECCIÓN. SENTENCIA IF

- Ejemplo 2: Representar la superación de un examen considerando  $\geq 5$ , aprobado.

```
import java.util.Scanner;

public class NotaAprobado{
    public static void main(String[] a) {
        int nota;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca nota a analizar:");
        nota = entrada.nextInt();
        if (nota > 5)
            System.out.println("Prueba superada ");
    }
}
```



## 2.1. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN. SENTENCIA IF

- Ejemplo 3: El programa selecciona el signo que tiene un número real.

```
import java.util.Scanner;
public class SignoNumero{
    public static void main(String[] a) {
        float numero;
        Scanner entrada = new Scanner(System.in);
        System.out.println("Introduzca un número real");
        numero = entrada.nextFloat();
        // comparar número con cero
        if (numero > 0)
            System.out.println(numero + " es mayor que cero");
        if (numero < 0)
            System.out.println(numero + " es menor que cero");
        if (numero == 0)
            System.out.println(numero + " es igual que cero");
    }
}
```



## 2.2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN. SENTENCIA IF DE DOS ALTERNATIVAS: IF-ELSE

- Un segundo formato de *if* es *if-else*, cuyo formato tiene la siguiente sintaxis:

*if (expresión)*

*acción 1*

*else*

*acción 2*

- En este formato *acción 1* y *acción 2* son, de forma individual, una única sentencia que termina en un punto y coma, o un grupo de sentencias entre llaves; *expresión* se evalúa cuando se ejecuta la sentencia: si es *verdadera*, se efectúa *acción 1*; en caso contrario se ejecuta *acción 2*.



## 2.2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN. SENTENCIA IF DE DOS ALTERNATIVAS: IF-ELSE

- Ejemplo1: Si salario es mayor que 100 000, se calcula el salario neto restándole los impuestos; en caso contrario (else), el salario neto es igual al salario bruto.

```
if (salario > 100000)
    salario_netto = salario - impuestos;
else
    salario_netto = salario;
```

Si salario es mayor que 100.000, se calcula el salario neto restándole los impuestos; en caso contrario (else), el salario neto es igual al salario bruto.




## 2.2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN. SENTENCIA IF DE DOS ALTERNATIVAS: IF-ELSE

- Ejemplo2: Prueba de divisibilidad.

```
import java.util.Scanner;

public class Divisible{
    public static void main(String[] a) {
        int n, d;
        Scanner entrada = new Scanner(System.in);
        System.out.print("Introduzca primer valor ");
        n = entrada.nextInt();
        System.out.print("Introduzca segundo valor ");
        d = entrada.nextInt();
        if (n % d == 0)
            System.out.println(n + " es divisible entre " + d);
        else
            System.out.println(n + " no es divisible entre " + d);
    }
}
```






## 2.2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN. SENTENCIA IF DE DOS ALTERNATIVAS: IF-ELSE

- Ejemplo3: determina el mayor de dos números ingresados y lo visualiza en pantalla; la entrada de los datos enteros se realiza de la misma forma que en el ejemplo anterior; por último, la selección del mayor se realiza con el operador `>` y la sentencia `if`.
- La condición es `(n1 > n2)`; si `n1` es mayor que `n2` la condición es verdadera; en caso de que `n1` sea menor o igual que `n2`, la condición es falsa; así se imprime `n1` cuando es mayor que `n2`, como en el ejemplo de la ejecución.

```
import java.util.Scanner;

public class MayorNumero{
    public static void main(String[] a) {
        int n1, n2;
        Scanner entrada = new Scanner(System.in);
        System.out.print("Introduzca primer entero: ");
        n1 = entrada.nextInt();
        System.out.print("Introduzca segundo entero: ");
        n2 = entrada.nextInt();
        if (n1 > n2)
            System.out.println(" El mayor es " + n1);
        else
            System.out.println(" El mayor es " + n2);
    }
}
```



## 2.3. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN. SENTENCIAS IF-ELSE ANIDADAS

- Una sentencia *if* es anidada cuando alguna de las ramas, sin importar si es verdadera o falsa, **también es if**; entonces se puede utilizar para tomar decisiones con *varias opciones o multiopciones*.

*if (condición 1)*

*sentencia 1*

*else if (condición 2)*

*sentencia 2*

...

*else if (condición n)*

*sentencia n*

*else*

*sentencia e*



## 2.3. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN. SENTENCIAS IF-ELSE ANIDADAS

- Ejemplo: Se incrementan contadores de números positivos, números negativos o ceros.

La sentencia if que está anidada tiene tres variables (num\_pos, num\_neg y num\_ceros), incrementa una de ellas en 1, dependiendo de si x es mayor que cero, menor que cero o igual a cero, respectivamente; su ejecución se realiza de la siguiente forma: se comprueba la primera condición ( $x > 0$ ) y, si es verdadera, num\_pos se incrementa en 1 y se omite el resto de la sentencia; si es falsa, se comprueba la segunda condición ( $x < 0$ ) y, si ésta es verdadera, num\_neg se incrementa en uno; en caso contrario num\_ceros se aumenta en uno.

Observe que la segunda condición sólo se comprueba si la primera condición es falsa.

```
if (x > 0)
    num_pos = num_pos + 1;
else if (x < 0)
    num_neg = num_neg + 1;
else
    num_ceros = num_ceros + 1;
```



## 2.4. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN. SENTENCIA DE CONTROL SWITCH

- En Java, **switch** es una sentencia que se utiliza para **elegir una de entre múltiples opciones**; es especialmente útil cuando la selección se basa en el valor de una variable simple o de una expresión simple denominada *expresión de control o selector*; *el valor de dicha expresión* puede ser de tipo **int** o **char** pero **no de tipo double**.

```
switch (selector)
{
  case etiqueta1 : sentencias1 ;
  break;
  case etiqueta2 : sentencias2 ;
  break;
  ...
  case etiquetan : sentencias n ;
  break;
  default: sentenciasd ; // opcional
}
```



## 2.4. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN. SENTENCIA DE CONTROL SWITCH

- La *expresión de control* o *selector* se **evalúa y compara con cada una de las etiquetas de case**; además, **debe ser un tipo ordinal**, por ejemplo, int, char, boolean pero no float o string; cada etiqueta es un valor único, constante y debe tener un valor diferente de los otros. Si el valor de la expresión es igual a una de las etiquetas case, por ejemplo *etiqueta1*, entonces la ejecución comenzará con la primera sentencia de *sentencias1* y **continuará hasta encontrar break o el final de switch**.
- El **tipo** de cada etiqueta debe ser el mismo que la expresión de selector.
- Si el valor del selector no está listado en ninguna etiqueta case no se ejecutará ninguna de las opciones a menos que se especifique una **acción predeterminada**. La omisión de una etiqueta **default** puede crear un error lógico difícil de prever; aunque ésta es *opcional*, se recomienda su uso, a menos de estar absolutamente seguro de que todos los valores de selector están incluidos en las etiquetas case.


## 2.5. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN/REPETICIÓN. SENTENCIA BREAK

- Para alterar el flujo de control de una sentencia de selección múltiple o de los bucles, existe la **sentencia break la cual termina el bucle.**

***break;***

***break*** *etiqueta;*

- Una sentencia *break* consta de la misma palabra reservada seguida por un punto y coma; cuando la computadora ejecuta las sentencias siguientes a una etiqueta case, continúa hasta que se alcanza una sentencia *break*; al hacerlo, la computadora termina la sentencia *switch*. Si se omiten las sentencias break después de ejecutar el código case, la computadora ejecutará el código siguiente hasta el próximo case.
- 

## 2.5. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. SELECCIÓN/REPETICIÓN. SENTENCIA BREAK

- Ejemplo: menú selector opciones del cero al dos.

```
Scanner teclado = new Scanner(System.in);  
int opcion = teclado.nextInt();  
switch (opcion) {  
    case 0:  
        System.out.println("Cero!");  
        break;  
    case 1:  
        System.out.println("Uno!");  
        break;  
    case 2:  
        System.out.println("Dos!");  
        break;  
    default:  
        System.out.println("Fuera de rango!");
```



## 2.5. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN/REPETICIÓN. SENTENCIA BREAK

- Ejemplo 2: agrupo varias opciones del case donde tienen el mismo bloque de sentencias.

```
Scanner teclado = new Scanner(System.in);  
int opcion = teclado.nextInt();  
switch (opcion) {  
    case 0:  
    case 1:  
    case 2:  
        System.out.println("Menor que 3!");  
        break;  
    case 3:  
        System.out.println("Igual a 3!");  
        break;  
    default:  
        System.out.println("Mayor que 3!");  
}
```





## 2.5. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.


### SELECCIÓN/REPETICIÓN. SENTENCIA BREAK

- Ejemplo 3: Considerando un rango entre 1 y 10 para asignar la nota de un curso, el programa ilustra la selección múltiple con la sentencia *switch*. Si no existiera *break* también se ejecutarían las sentencias restantes de *switch*.

```
import java.util.Scanner;

public class PruebaCompuesta{
    public static void main(String[] a) {
        int nota;
        Scanner entrada = new Scanner(System.in);
        System.out.print("Introduzca calificación "
            + "(1 - 10), pulse Intro:");
        nota = entrada.nextInt();
    }
}
```

```
switch (nota) {
    case 10:
    case 9:
        System.out.println("Excelente.");
        break;
    case 8:
    case 7:
        System.out.println("Notable.");
        break;
    case 6:
    case 5:
        System.out.println("Aprobado.");
        break;
    case 4:
    case 3:
    case 2:
    case 1:
    case 0:
        System.out.println("Suspendido.");
        break;
    default:
        System.out.println("no es posible esta nota.");
}
}
```



## 2.5. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### SELECCIÓN/REPETICIÓN. SENTENCIA BREAK

- Ejemplo 4: Este ejemplo selecciona tipo de vehículo y, en concordancia, asigna peaje y salta la ejecución a la sentencia que sigue switch. Completa el código y ejecútalo.

```
int tipo_vehiculo;
System.out.println("Introduzca tipo de
vehículo:");
tipo_vehiculo = entrada.nextInt();
switch (tipo_vehiculo) {
case 1:
System.out.println("turismo");
peaje = 500;
break;
case 2:
System.out.println("autobus");
peaje = 3000;
break;
case 3:
System.out.println("motocicleta");
peaje = 300;
break;
default:
System.out.println("vehículo no autorizado");
}
```

Si se omite el primer *break*, el primer vehículo será turismo y luego, autobús.

Cuando la computadora comienza a ejecutar case no se detiene hasta que encuentra una sentencia *break* o bien termina *switch* y sigue en secuencia.



## 2.5. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. SELECCIÓN. EJERCICIOS

1. Antes de realizar los ejercicios de selección, veamos la clase *String*. Se trata de una clase que nos permite guardar objetos de cadenas de texto con métodos muy útiles. Revisa la documentación ***Clase String*** del aula virtual.
2. Realiza los ejercicios **1,4,6,7,8** de la hoja de ***Ejercicios con String***.
3. Realiza la hoja de ejercicios ***Ejercicios de selección***.



## 2.6. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. SENTENCIA WHILE

- Un **bucle** es cualquier construcción de programa que **repite una sentencia o secuencia de sentencias** determinado **número de veces**; cuando ésta se menciona varias veces en un bloque se denomina *cuerpo del bucle*; cada vez que éste se repite se denomina *iteración del bucle*.
- Un bucle **while** tiene una condición, una expresión lógica que controla la secuencia de repetición; su posición es delante del cuerpo del bucle y significa que *while* es un bucle *pretest*, de modo que cuando éste se ejecuta, se evalúa la condición antes de ejecutarse el cuerpo del bucle; la sentencia o sentencias expresadas se repite mientras la condición del bucle permanece verdadera y termina al volverse falsa. La condición se examina antes de ejecutarse el cuerpo y, por consiguiente, si aquélla es inicialmente falsa, éste no se ejecutará; es decir, el cuerpo de un bucle *while* se ejecutará cero o más veces.

## 2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. SENTENCIA WHILE

- Una de las aplicaciones más usuales del operador de incremento ++ es la de controlar la iteración de un bucle. Ejemplo: cálculo de calorías:

```
import java.util.Scanner;
public class Calorias{
    public static void main(String[] a) {
        int num_de_elementos, cuenta, calorías_por_alimento, calorías_total;
        Scanner entrada = new Scanner(System.in);
        System.out.print("¿Cuántos alimentos ha comido hoy? ");
        num_de_elementos = entrada.nextInt();
        System.out.println("Introducir el número de calorías de cada uno de los " +
            num_de_elementos + " alimentos tomados:");
        calorías_total = 0;
        cuenta = 1;
        while (cuenta++ <= num_de_elementos) {
            calorías_por_alimento = entrada.nextInt();
            calorías_total += calorías_por_alimento;
        }
        System.out.println("Las calorías totales consumidas hoy son = " + calorías_total);
    }
}
```

## 2.6. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. SENTENCIA WHILE

- Ejemplo: visualizar n asteriscos.
- La variable que representa la condición del bucle también se denomina *variable de control* debido a que su valor determina si el cuerpo se repite; ésta debe ser: 1) inicializada, 2) comprobada y 3) actualizada para que aquél se ejecute adecuadamente; cada etapa se resume así:
  1. **Inicialización.** Se establece contador a un valor inicial antes de que se alcance la sentencia while, aunque podría ser cualquiera, generalmente es 0.
  2. **Prueba/condición.** Se comprueba el valor de contador antes de la iteración; es decir, el comienzo de la repetición de cada bucle.
  3. **Actualización.** Durante cada iteración, contador actualiza su valor incrementándose en uno mediante el operador ++.

```
int contador = 0; // inicialización
int n=5; //número máximo de iteraciones
// prueba/condición
while (contador < n) {
    System.out.print(" * ");
    // actualización (incrementa en 1 contador)
    contador++;
} // fin de while
```



## 2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. SENTENCIA WHILE

- Ejemplo: Cálculo de la media de 6 números.

```
import java.util.Scanner;
public class MediaSeis{
    public static void main(String[] a) {
        Scanner entrada = new Scanner(System.in);
        final int TotalNum = 6;
        int contadorNum = 0;
        double sumaNum = 0.0;
        double media;
        System.out.println("/nIntroduzca %d números" + TotalNum);
        while (contadorNum < TotalNum) {
            // valores a procesar
            double numero;
            numero = entrada.nextDouble();
            sumaNum += numero; // añadir valor a Acumulador
            ++contadorNum; // incrementar números leídos
        }
        media = sumaNum / contadorNum;
        System.out.println("Media: \n" + media);
    }
}
```

Actualiza el programa, en vez de hacer la media de 6 número, solicita al usuario de cuántos números quiere hacer la media y úsalo. Formatea la salida con 2 decimales.



## 2.6.1. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. SENTENCIA WHILE. BUCLE INFINITO

- Si la variable de control no se actualiza se ejecutará un **bucle infinito**; en otras palabras esto sucede cuando su condición permanece sin hacerse falsa en alguna iteración.

Ejemplo:

```
// bucle infinito
contador = 1;
while (contador < 100)
{
    System.out.println(contador);
    contador--; //decrementa en 1 contador
}
```

Se inicializa contador a 1 (menor que 100), como contador-- decrementa en 1 el valor de contador en cada iteración su valor nunca llegará a 100, número necesario para que la condición sea falsa; por consiguiente, contador < 100 siempre será verdadera, resultando un bucle infinito.

- Crea y prueba un bucle infinito en tu programa. ¿Cómo lo solucionarías?



## 2.6.2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. SENTENCIA WHILE. BUCLES CONTROLADOS POR CENTINELAS

- Por lo general, no se conoce con exactitud cuántos elementos de datos se procesarán antes de comenzar su ejecución.
- Para manejar esta situación el usuario introduce, al final, **un dato único**, definido y específico en el llamado *valor centinela*; al hacerlo, *la condición del bucle* comprueba cada dato y **termina** cuando lee dicho valor. El valor *centinela* se selecciona con mucho cuidado pues no debe haber forma de que se produzca como dato.
- Es decir, **el centinela sirve para terminar el proceso del bucle.**



## 2.6.2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. SENTENCIA WHILE. BUCLES CONTROLADOS POR CENTINELAS

- Ejemplo: se introducen notas mientras sean distintas al centinela (valor -1). Actualiza el programa para que funcione correctamente y ejecútalo.

```
public static void main(String[] a) {  
    final int centinela = -1;  
    System.out.print("Introduzca primera nota:");  
    nota = entrada.nextInt();  
    while (nota != centinela) {  
        cuenta++;  
        suma += nota;  
        System.out.print("Introduzca siguiente nota: ");  
        nota = entrada.nextInt();  
    } // fin de while  
    System.out.println("Final");  
}
```



## 2.6.3. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. SENTENCIA WHILE. BUCLES CONTROLADOS POR INDICADORES O BANDERAS

- Con frecuencia se utilizan variables de tipo *boolean* como indicadores o ***banderas*** de estado para controlar la ejecución de un bucle; su valor se inicializa normalmente como falso y, antes de la entrada al bucle, se redefine cuando un suceso específico ocurre dentro del bucle. **Un bucle controlado por un indicador o bandera se ejecuta hasta que se produce el suceso anticipado y se cambia el valor del indicador.**
- Ejemplo: Se desea leer diversos datos de tipo carácter introducidos por teclado mediante un bucle *while*; el cual *terminará cuando se lea un dato tipo dígito con rango entre 0 y 9*. Veamos el ejemplo:



## 2.6.3 ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. SENTENCIA WHILE. BUCLES CONTROLADOS POR INDICADORES O BANDERAS

```
digito_leido = false; // no se ha leído ningún dato
while (!digito_leido)
{
    System.out.print("Introduzca un carácter: ");
    car = System.in.read(); // lee siguiente carácter
    System.in.skip(1); // salta 1 carácter(fin de línea)
    digito_leido = (('0' <= car) && (car <= '9'));
    ...
} // fin de while
```

La variable bandera *digito\_leido* se utiliza para indicar cuando un dígito se introduce por medio del teclado; su valor es **falso** antes de entrar al bucle y mientras el dato leído sea un carácter, pero es **verdadero** si se trata de un dígito. Al comenzar, la ejecución del bucle deberá continuar mientras el dato leído sea un carácter y, en consecuencia, la variable *digito\_leido* tenga un valor falso.

- El bucle se **detendrá** cuando dicho dato sea un dígito y, en este caso, el valor de la variable *digito\_leido* cambiará a **verdadero**. En consecuencia, la condición del bucle debe ser *!digito\_leido* ya que es verdadera cuando *digito\_leido* es falso.
- Crea un programa que pida número enteros pares y pare cuando se introduzca un número impar. Puedes utilizar la clase Scanner para pedir los números.

## 2.6.4. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. SENTENCIA WHILE. SENTENCIA BREAK EN BUCLES

- La sentencia ***break*** a veces se utiliza para realizar una terminación anormal del **bucle** o antes de lo previsto; su sintaxis es:

***break*;**

- La sentencia *break* se utiliza para la salida de un bucle *while*, *do-while* o *for*, aunque su uso más frecuente es dentro de una sentencia *switch*.

```
while (condición1)
{
    if (condición2)
        break;
    // sentencias
}
```



## 2.6.4. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. SENTENCIA WHILE. SENTENCIA BREAK EN BUCLES

- Ejemplo: lee y visualiza los valores de entrada hasta que se encuentra el valor clave especificado:

```
public static void main(String[] a) {  
    Scanner entrada= new Scanner(System.in);  
    int clave = -9;  
    boolean activo = true;  
    while (activo) {  
        int dato;  
        dato = entrada.nextInt();  
        if (dato != clave)  
            System.out.println(dato);  
        else  
            break;  
    }  
}
```

El método `nextInt()` lee un número entero desde el dispositivo de entrada; si su condición siempre fuera *true*, se ejecutaría indefinidamente; sin embargo, cuando hay un *dato==clave*, la ejecución sigue por *else* y, por su parte, *break* hace que la ejecución continúe en la sentencia siguiente a *while*.



## 2.6.5. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. SENTENCIA WHILE. LA SENTENCIA BREAK CON ETIQUETA

- Para transferir el control a la siguiente sentencia de una estructura de bucles anidados, se utiliza la sentencia **break con etiqueta**; cuya sintaxis es:

*break etiqueta*

- Ejemplo: pide un día de la semana y calcula cuántos quedan para terminar la semana. Veamos el ejemplo.



## 2.6.5 ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. SENTENCIA WHILE. LA SENTENCIA BREAK CON ETIQUETA

```
public static void main(String[] a) {  
    Scanner entrada = new Scanner(System.in);  
    int dia = 0, i = 1;  
    System.out.println("Escribe día de la semana del 1 al 7:");  
    dia = entrada.nextInt();  
    seguir: while (i <= 7) {  
        if (dia == i) {  
            System.out.println("Hoy es el " + i + "º día de la semana." +  
                "Quedan " + (7 - i) + " días para terminar la semana.");  
            i++;  
            break seguir;  
        }  
        System.out.println("Día " + i);  
        i++;  
    }  
    System.out.println("Seguimos después...");  
}
```





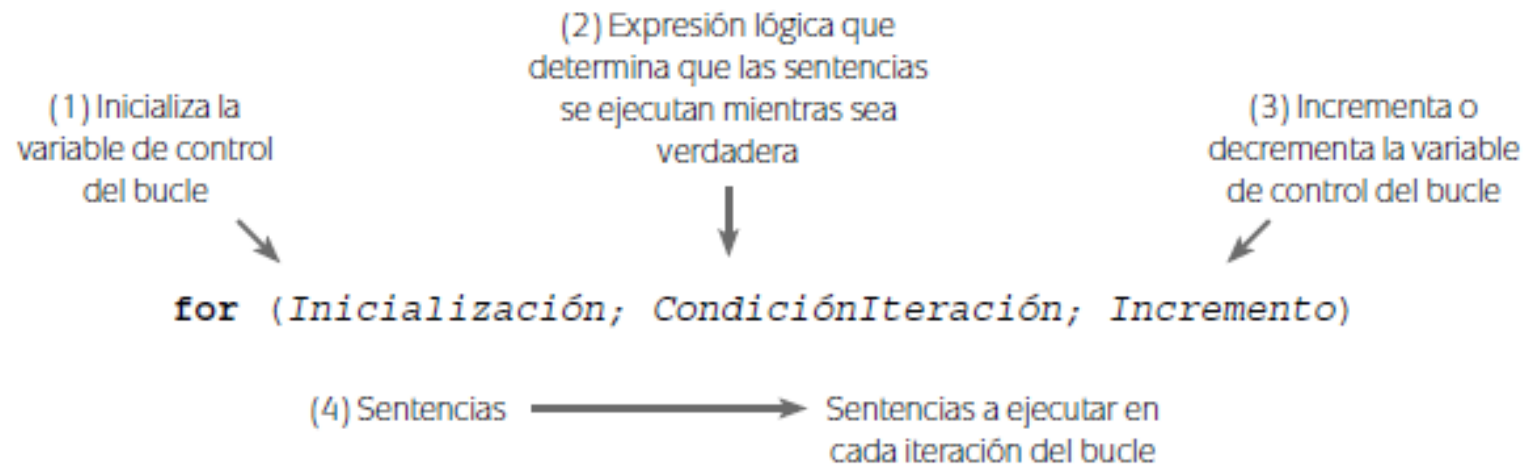
## 2.7. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. BUCLE FOR

- Java proporciona otros dos tipos de bucles: **for** y **do**; el primero es el más adecuado para implementar conjuntos de sentencias que se ejecutan una vez por cada valor de un rango especificado; a éstos se les llama *bucles controlados por contador*, y su algoritmo es:

por cada valor de una variable\_contador de un rango específico:  
ejecutar sentencias

- La sentencia o bucle **for** es la mejor forma de programar la ejecución de un bloque de sentencias **un número fijo de veces**; éste sitúa las operaciones de control del bucle en la cabecera de la sentencia.



## 2.7. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLE FOR


- El bucle **for** se compone de:
  - Una parte de inicialización que comienza la variable o variables de control; pueden definirse en esta parte y ser simples o múltiples.
  - Una parte de condición que contiene una expresión lógica y que itera las sentencias mientras la expresión sea verdadera.
  - Una parte que incrementa o decrementa la variable o variables de control del bucle (separadas por comas).
  - Sentencias o acciones que se ejecutarán por cada iteración del bucle.



## 2.7. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLE FOR

○ Ejemplo:

```
public static void main(String[] a) {  
    // imprimir Hola 10 veces  
    for (int i = 0; i < 10; i++)  
        System.out.println("HoLa!");  
    // Otra forma con varios incrementos:  
    int i,j = 0;  
    for (i = 0; i < 10; i++,j++) {  
        System.out.println("HoLa!");  
        System.out.println("El valor de i es: " + i+" el valor de j: "+j);  
    }  
}
```



## 2.7. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLE FOR

- Ejemplo: con formato descendente

```
for (int n = 10; n > 5; n--)  
    System.out.println("\t" + n + "\t" + n * n);
```

- Su salida es:

10	100
9	81
8	64
7	49
6	36

- Debido a que el valor inicial de la variable de control es 10, y el límite que se ha puesto es  $n > 5$ ; es decir, es verdadera cuando  $n = 10, 9, 8, 7, 6$ ; la expresión de decremento es  $n--$  que disminuye en 1 el valor de  $n$  tras la ejecución de cada iteración.



## 2.7. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. BUCLE FOR

- Los rangos de incremento/decremento de la variable o expresión de control del bucle pueden tener cualquier valor y no siempre 1, es decir 5, 10, 20, 4, etc, dependiendo de los intervalos necesarios;
- Ejemplo:

```
//utiliza la expresión de incremento n += 20
```

```
for (int n = 0; n < 100; n += 20)
```

```
System.out.println("\t" + n + "\t" + n * n );
```

Aumenta el valor de n en 20, puesto que equivale a  $n = n + 20$ ; por tanto, la salida que producirá la ejecución del bucle es:

0	0
20	400
40	1600
60	3600
80	6400

## 2.7. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. BUCLE FOR

- Ejemplo: Inicializa la variable de control del bucle `c` al carácter `'A'`, lo cual equivale a inicializar al entero 65 porque éste es el código ASCII de `A` e itera mientras el valor de la variable `c` sea menor o igual que el ordinal del carácter `'Z'`. La parte de incremento del bucle aumenta el valor de la variable `c` en 1; por consiguiente, el bucle se realiza tantas veces como letras mayúsculas se necesiten.

```
public static void main(String[] a) {  
    for (int c = 'A'; c <= 'Z'; c++)  
        System.out.print(c + " ");  
        System.out.println();  
}
```

Salida: ¿Qué salida aparece?  
¿Por qué?



## 2.7. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. BUCLE FOR

- Ejemplos: realiza un programa que haga lo que indica cada ejemplo y responde a la pregunta: ¿cuántas veces se ejecuta el bucle? ¿con qué valores?
- 1. Muestra un bucle descendente que inicializa la variable de control a 9, el cual se realiza mientras i no sea negativo; la variable disminuye en 3.
- 2. La variable de control i se inicializa a 1 y se incrementa en múltiplos de 2.
- 3. Declara 2 variables de control i y j y las inicializa a 0 y la constante MAX a 25; el bucle se ejecutará mientras i sea menor que j; además, i se incrementa en 1, mientras j se decrementa en 1.




## 2.7. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLE FOR

- Ejemplo: Suma de los M primeros números pares. Con M=12,

```
public static void main(String[] a) {  
    final int M = 12;  
    int suma = 0;  
    for (int n = 1; n <= M; n++)  
        suma += 2 * n;  
    System.out.println("La suma de los " + M + " primeros números pares: " + suma);  
}
```

- El bucle puede diseñarse con un incremento de 2:

```
public static void main(String[] a) {  
    final int M = 12;  
    int suma = 0;  
    for (int n = 2; n <= 2*M; n += 2)  
        suma += n;  
    System.out.println("La suma de los " + M + " primeros números pares: " + suma);  
}
```





## 2.7.1. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLE FOR. PRECAUCIONES EN EL USO DE FOR

- Un bucle **for** se debe construir con precaución, asegurándose que la expresión de inicialización y la de incremento harán que la condición se convierta en false en algún momento; en particular **“si el cuerpo de un bucle de conteo modifica los valores de cualquier variable implicada en la condición, entonces el número de repeticiones se puede modificar”**;
- Malos ejemplos de programación:

```
final int LIM = 50;
int iter, tope;
for (iter = tope = 0; tope <= LIM; iter++) {
    System.out.println("Iteración: " + iter);
    tope = entrada.nextInt();
}
```

```
int limite = 11;
for (int i = 0; i <= limite; i++) {
    System.out.println(i);
    limite++;
}
```

## 2.7.2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLE FOR. BUCLES INFINITOS

- La sentencia se ejecuta **indefinidamente** a menos que se utilice **return** o **break**; aunque normalmente es una combinación **if-break** o **if-return**; la razón de la ejecución indefinida es que se eliminó la expresión de inicialización, la condición y la expresión de incremento, y al no existir una condición específica para terminar la repetición de sentencias, se asume que la condición es verdadera;
- Para evitar esto, se requiere que el diseño del bucle *for* sea de la forma siguiente:
  1. El cuerpo del bucle debe contener todas las sentencias que se desean ejecutar repetidamente.
  2. Una sentencia terminará la ejecución del bucle cuando se cumpla determinada condición.
- La sentencia de terminación suele ser **if-break**. Veamos su sintaxis:

## 2.7.2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLE FOR. BUCLES INFINITOS

*for (;;) // bucle*

*{*

*lista\_sentencias1*

*if (condición\_terminación)*  
*break;*

*lista\_sentencias2*

*} // fin del bucle*

*lista\_sentencias puede ser*  
*vacía, simple o compuesta.*

`if (condición) break;`

o bien:

`if (condición) break label;`

`condición`

es una expresión lógica

`break`

termina la ejecución del bucle y transfiere el control a la sentencia siguiente al bucle.

`break label;`

termina la ejecución del bucle y transfiere el control a la sentencia siguiente al bucle con esa etiqueta.



## 2.7.2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLE FOR. BUCLES INFINITOS.

- Ejemplo: bucle indefinido cuya ejecución se termina al teclear un valor equivalente a CLAVE.

```
final int CLAVE = -999;  
for (;;)   
{  
    System.out.println("Introduzca un número " + CLAVE +  
        " para terminar");  
    num = entrada.nextInt();  
    if (num == CLAVE) break;  
    ...  
}
```



## 2.7.3. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. BUCLE FOR. EXPRESIONES NULAS EN BUCLES FOR

- Cualquiera de las tres o todas las expresiones que controlan un bucle **for** pueden ser nulas o vacías; el punto y coma (;) es el que marca una expresión vacía. Si la intención es crear un bucle *for* que actúe exactamente como *while*, se deben dejar vacías la primera y tercera sentencias; por ejemplo, el siguiente *for* funciona como *while*, y tiene vacías las expresiones de inicialización y de incremento. Ejemplo:

```
int contador = 0;
for (;contador < 5;)
{
    contador++;
    System.out.print(";Bucle! ");
}
System.out.println("\n Contador: " + contador);
```



## 2.8. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. SENTENCIA CONTINUE

- Esta sentencia se utiliza en el contexto de un bucle y hace que la ejecución prosiga con la siguiente iteración saltando las sentencias que están a continuación; en bucles **while/do while**, la ejecución prosigue con la condición de control del bucle, mientras que en bucles **for**, la ejecución prosigue con la expresión de incremento.
- Sintaxis:
  - `continue;`
  - `continue etiqueta;`



## 2.8. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. SENTENCIA CONTINUE

- Ejemplo: Se generan 5 números aleatorios de forma que se escriban todos excepto los múltiplos de 3.

```
public static void main(String[] a) {  
    final int CLAVE = 3;  
    final int RANGO = 10;  
    int n = 5;  
    for (int i = 0; i < n; i++) {  
        int numero;  
        numero = (int) (Math.random()*RANGO+1);  
        if (numero % CLAVE == 0) {  
            System.out.println("\nHa salido el 0");  
            continue;  
        }  
        System.out.print(" " + numero);  
    }  
}
```



## 2.8. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. SENTENCIA CONTINUE

- Cuando la sentencia **continue** se ejecuta con etiqueta en una estructura repetitiva, salta las sentencias que quedan hasta el final del cuerpo del bucle y la ejecución prosigue con la siguiente iteración que está enseguida de la etiqueta especificada; en bucles *while/do-while*, la ejecución prosigue con la evaluación de la condición de control que se encuentra después de la etiqueta; por último, en bucles *for* la ejecución prosigue con la expresión de incremento.





## 2.8. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. SENTENCIA CONTINUE

- Ejemplo: El programa Asteriscos escribe líneas con asteriscos en una cantidad igual al número de línea correspondiente del código fuente.

```
public static void main(String[] a) {  
    final int COLUMNA = 17;  
    final int FILA = 17;  
    siguiente: for (int f = 1; f <= FILA; f++){  
        System.out.println();  
        for (int c = 1; c <= COLUMNA; c++){  
            if (c > f)  
                continue siguiente;  
            System.out.print("*");  
        }  
    }  
}
```

Si se cumple la condición  $c > f$ , se ejecuta la sentencia continue siguiente, de forma que el flujo continúa en la expresión  $f++$  del bucle externo.



## 2.9. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. BUCLE DO...WHILE

- La sentencia **do-while** se utiliza para especificar un bucle condicional que **se ejecuta al menos una vez**; cuando se desea realizar una acción determinada al menos una o varias veces, se recomienda este bucle.

Acción (sentencia) a  
ejecutar al menos una vez

Expresión lógica que  
determina si la acción se repite

1. `do` *sentencia* `while` (*expresión*)

2. `do`  
    *sentencia*  
    `while` (*expresión*)

La construcción *do* comienza ejecutando *sentencia*; *enseguida* se evalúa *expresión*; si ésta es verdadera, entonces se repite la ejecución de *sentencia*; continuando hasta que *expresión* sea falsa.

## 2.9. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLE DO...WHILE

- Ejemplo: bucle para introducir un dígito del 0 al 9 (ambos inclusivos).


```
public static void main(String[] a) throws IOException {  
    Scanner entrada=new Scanner (System.in);  
    int digito;  
    do  
    {  
        System.out.println("Introduzca un dígito (0-9): ");  
        digito =  entrada.nextInt(); // lee número  
    } while ((digito>=0) && (digito<=9));  
}
```



## 2.9. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLE DO...WHILE

- Ejemplo: una opción de saludo al usuario dentro de un programa.

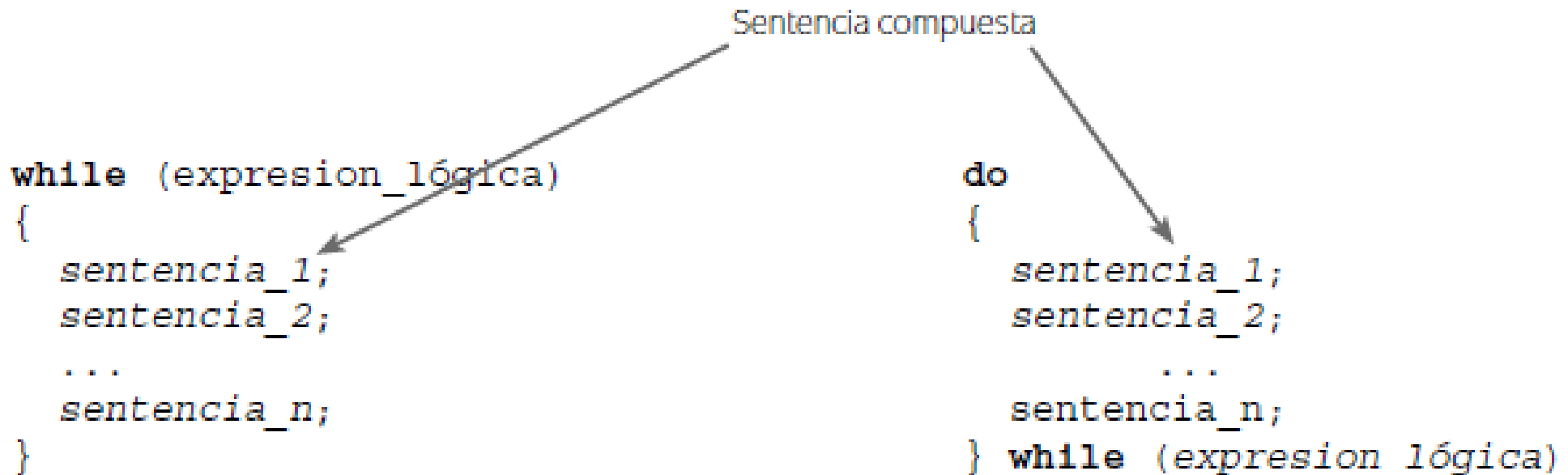
```
public static void main(String[] a) throws IOException {  
    Scanner entrada= new Scanner (System.in);  
    char opcion;  
    do {  
        System.out.println("Hola");  
        System.out.println("¿Desea otro tipo de saludo?");  
        System.out.println("Pulse s para si y n para no,");  
        System.out.print("y a continuación pulse intro: ");  
        opcion = (char) System.in.read();  
        System.in.skip(System.in.available()); //Salto los bytes que  
        pueden ser leídos  
    } while (opcion == 's' || opcion == 'S');  
    System.out.println("Adiós");  
}
```



## 2.10. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. DIFERENCIAS ENTRE WHILE Y DO-WHILE

- Una sentencia **do-while** es similar a **while** excepto que el cuerpo del bucle siempre se ejecuta al menos una vez.



## 2.10. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. DIFERENCIAS ENTRE WHILE Y DO-WHILE

- Ejemplo: visualizar las letras minúsculas con los bucles while y do-while.

```
// bucle do-while
char car = 'a';
do
{
    System.out.print(car + " ");
    car++;
} while (car <= 'z');
```

```
bucle while
char car = 'a';
while (car <= 'z')
{
    System.out.print(car + " ");
    car++;
}
```

- Ejemplo: Visualizar las potencias de dos (número entero) cuyos valores estén en el rango 1 a 1 000 con los bucles while y do-while.

```
// Realizado con while
pot = 1;
while (pot < 1000)
{
    System.out.println(pot);
    pot * = 2;
} // fin de while
```

```
// Realizado con do-while
pot = 1;
do
{
    System.out.println(pot);
    pot * = 2;
} while (pot < 1000);
```



## 2.11. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN.

### REPETICIÓN. COMPARACIÓN DE BUCLES WHILE, FOR Y DO-WHILE

- El bucle *for* se utiliza normalmente cuando el conteo está implicado o el número de iteraciones requeridas se pueda determinar al principio de la ejecución del bucle o, simplemente, cuando es necesario seguir el número de veces que un suceso particular ocurre.
- El bucle *do-while* se ejecuta de un modo similar a *while*, excepto que las sentencias del cuerpo siempre se ejecutan al menos una vez.

while	Su uso más frecuente es cuando la repetición no está controlada por contador; el test de condición precede a cada repetición del bucle; el cuerpo puede no ser ejecutado. Se debe utilizar cuando se desea saltar el bucle si la condición es falsa.
for	Bucle de conteo que se emplea cuando el número de repeticiones se conoce por anticipado y puede controlarlo un contador; también es adecuado para bucles que implican control no contable del bucle con simples etapas de inicialización y actualización; el test de la condición precede a la ejecución del cuerpo.
do-while	Es apropiado cuando se necesita que el bucle se ejecute al menos una vez.

## 2.11. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. COMPARACIÓN DE BUCLES WHILE, FOR Y DO-WHILE

- Ejemplo de la construcción de los tres bucles:

```
cuenta = valor_inicial;  
while (cuenta < valor_parada)  
{  
    ...  
    cuenta++;  
} // fin de while
```

```
cuenta = valor_inicial;  
if (valor_inicial < valor_parada)  
do  
{  
    ...  
    cuenta++;  
} while (cuenta < valor_parada);
```

```
for (cuenta = valor_inicial; cuenta < valor_parada; cuenta++)  
{  
    ...  
} // fin de for
```





## 2.12. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLES ANIDADOS

- Es posible **anidar bucles**, los cuales tienen un bucle externo y uno o más internos; cada vez que se repite el externo, los internos también lo hacen; los componentes de control se vuelven a evaluar y se ejecutan todas las iteraciones requeridas.
- Ejemplo: escribir las variables de control de dos bucles anidados.

```
public static void main(String[] a) {  
    System.out.println("\n\t\t i \t j"); // cabecera de salida  
    for (int i = 0; i < 4; i++) {  
        System.out.println("Externo\t\t " + i);  
        for (int j = 0; j < i; j++)  
            System.out.println("Interno\t\t\t " + j);  
    } // fin del bucle externo  
}
```



## 2.12. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. BUCLES ANIDADOS

- Ejemplo: escribir una aplicación que visualice el siguiente triángulo isósceles:

```
  *
 ***
*****
*****
*****
```

```
public static void main(String[] a) {
    final int NUMLINEAS = 5;
    final char BLANCO = 32; // ASCII 32 es espacio en blanco
    final char ASTERISCO = '*';
    // bucle externo: dibuja cada línea
    for (int fila = 1; fila <= NUMLINEAS; fila++) {
        System.out.print("\t");
        // primer bucle interno: escribe espacios
        for (int blancos = (NUMLINEAS - fila); blancos > 0;
            blancos--)
            System.out.print(BLANCO);
        for (int cuenta_as = 1; cuenta_as < 2 * fila; cuenta_as++)
            System.out.print(ASTERISCO);
        System.out.println(" ");
    } // fin del bucle externo
}
```

El bucle externo se repite 5 veces, uno por línea o fila; el número de repeticiones ejecutadas por los internos se basa en el valor de fila.

La primera fila consta de un asterisco y 4 blancos, la segunda está conformada por 3 blancos y 3 asteriscos, y así sucesivamente; la quinta tendrá 9 asteriscos ( $2 * 5 - 1$ ).

## 2. ESTRUCTURAS DE CONTROL. SELECCIÓN. REPETICIÓN. REPETICIÓN. EJERCICIOS

1. Realiza los ejercicios 2,3,5,9 de la hoja de *Ejercicios con String*.
2. Realiza los ejercicios de la hoja *Ejercicios con bucles*

