DESARROLLO DE APLICACIONES MULTIPLATAFORMA

PROGRAMACIÓN

TEMA 3

ESTRUCTURAS DE CONTROL Y FUNCIONES

(Lenguaje de Programación Java)

CONTENIDOS

3.1. ESTRUCTURAS DE SELECCIÓN.

Una expresión es una serie de variables, constantes y/o datos unidos por operadores:

Ejemplo: *2 * PI * radio*

Una sentencia es una expresión que acaba en punto y coma (;):

Ejemplo: *area = 2 * PI * radio ;*

Un bloque de código es un conjunto de sentencias e instrucciones encerradas entre llaves { … }.

3.1. ESTRUCTURAS DE SELECCIÓN.

3.1.1. ESTRUCTURAS IF.

En Java hay tres tipos de estructuras de selección o condicionales IF:

if

if - else

if - else if – else

El formato es el siguiente:

```
if (expresion_booleana) {
    sentencias;
}
```

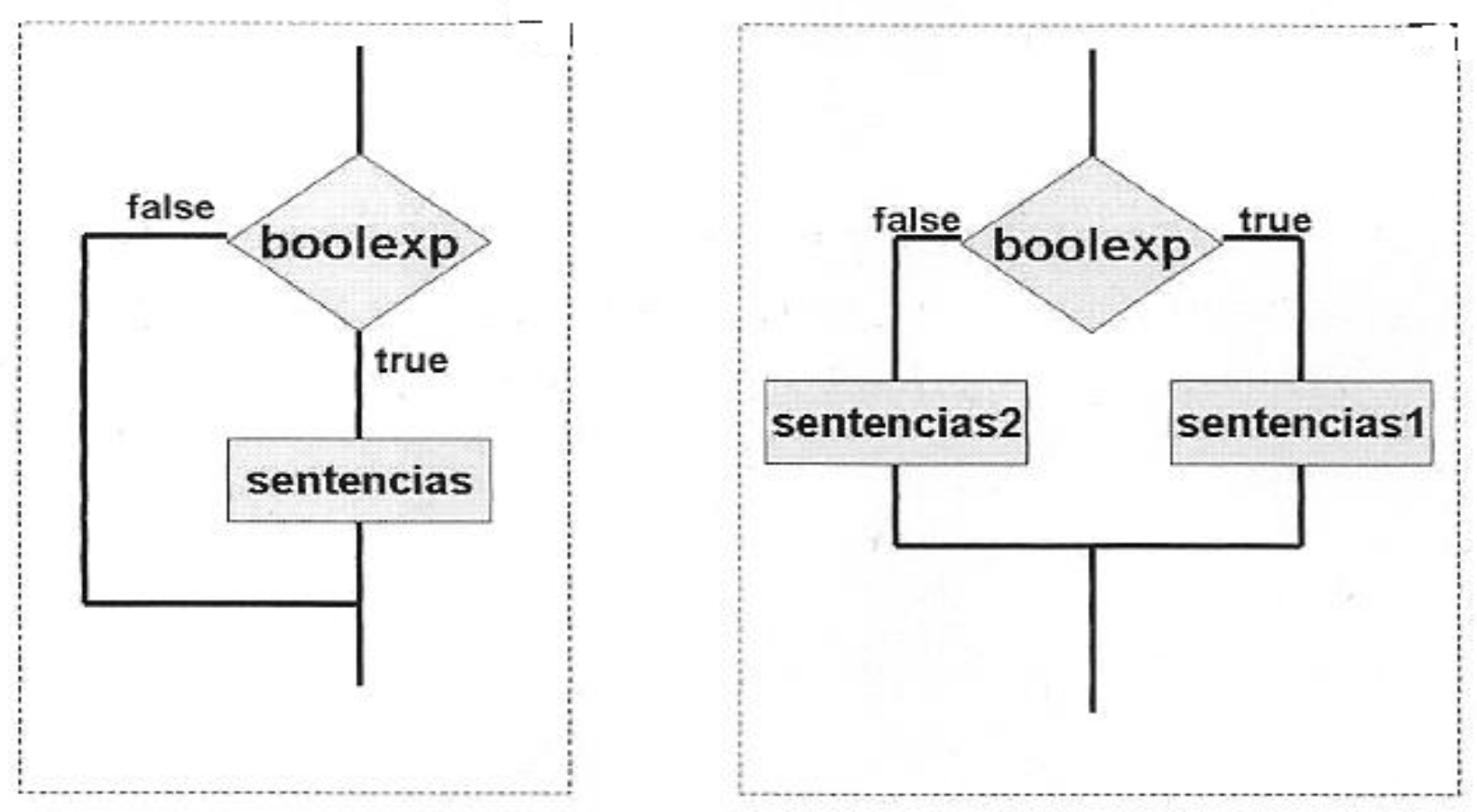
```
if (expresion_booleana) {
    sentencias1;
} else {
    sentencias2;
}
```

```
if (expresion_booleana1) {
    sentencias1;
} else if (expresion_booleana2) {
    sentencias2;
} else {
    sentencias3;
}
```

3.1. ESTRUCTURAS DE SELECCIÓN.

3.1.1. ESTRUCTURAS IF.

Una expresión booleana puede ser verdadera o falsa. Dependiendo de si es verdadera o falsa se ejecutarán o no unas sentencias. Además hay que tener en cuenta que las estructuras IF se pueden combinar o anidar unas dentro de otras.



3.1. ESTRUCTURAS DE SELECCIÓN.

3.1.1. ESTRUCTURAS IF.

Ejemplo I:

```
int x=0;
if (x==1) {
    System.out.println("La variable x vale 1");
}
if (x==1) {
    System.out.println("La variable x vale 1");
} else if (x>1) {
    System.out.println("La variable x es mayor que 1");
}
if (x==1) {
    System.out.println("La variable x vale 1");
} else if (x>1) {
    System.out.println("La variable x es mayor que 1");
} else {
    System.out.println("La variable x es menor que 1");
}
```

3.1. ESTRUCTURAS DE SELECCIÓN.

3.1.1. ESTRUCTURAS IF.

Ejemplo II:

```
if (x==1) {
    System.out.println("La variable x vale 1");
} else {
    if (x>1) {
        System.out.println("La variable x es mayor que 1");
    } else {
        System.out.println("La variable x es menor que 1");
    }
}
```

3.1. ESTRUCTURAS DE SELECCIÓN.

3.1.2. ESTRUCTURAS SWITCH.

Cuando una expresión puede tener varios valores y dependiendo del valor que tome hay que ejecutar una serie de sentencias u otras, tenemos dos opciones a la hora de programar.

La primera es utilizar la estructura SWITCH la cual deja el código limpio y fácil de interpretar, y la segunda, es utilizar estructuras IF concatenadas.

```
switch (expresion) {
    case valor1: sentencias1; break;
    case valor2: sentencias2; break;
    case valor3: sentencias3; break;
    ...
    case valorN: sentenciasN; break;
    [default: sentencias;]
}
```

Si no se escribe la sentencia break, el programa seguirá ejecutando las siguientes sentencias hasta encontrarse con un break o el fin del switch.

3.1. ESTRUCTURAS DE SELECCIÓN.

3.1.2. ESTRUCTURAS SWITCH.

Ejemplo:

```
int nota=10;

switch (nota) {
     case 5:
     case 6: System.out.println("APROBADO"); break;
     case 7:
     case 8: System.out.println("NOTABLE"); break;
     case 9:
     case 10: System.out.println("SOBRESALIENTE"); break;

     default:  System.out.println("SUSPENSO");
}
```

3.2. ESTRUCTURAS DE REPETICIÓN.

Las estructuras de repetición o bucles son utilizadas cuando una o varias sentencias han de ser ejecutadas cero, una o más veces.

Especial atención con los bucles infinitos. Los bucles infinitos son aquellos que no terminan nunca (aquellos cuya expresión *booleana* es cierta siempre). En el caso de producirse un bucle infinito, el programa se seguirá ejecutando y se quedará "colgado" hasta que el usuario "mate" el proceso.

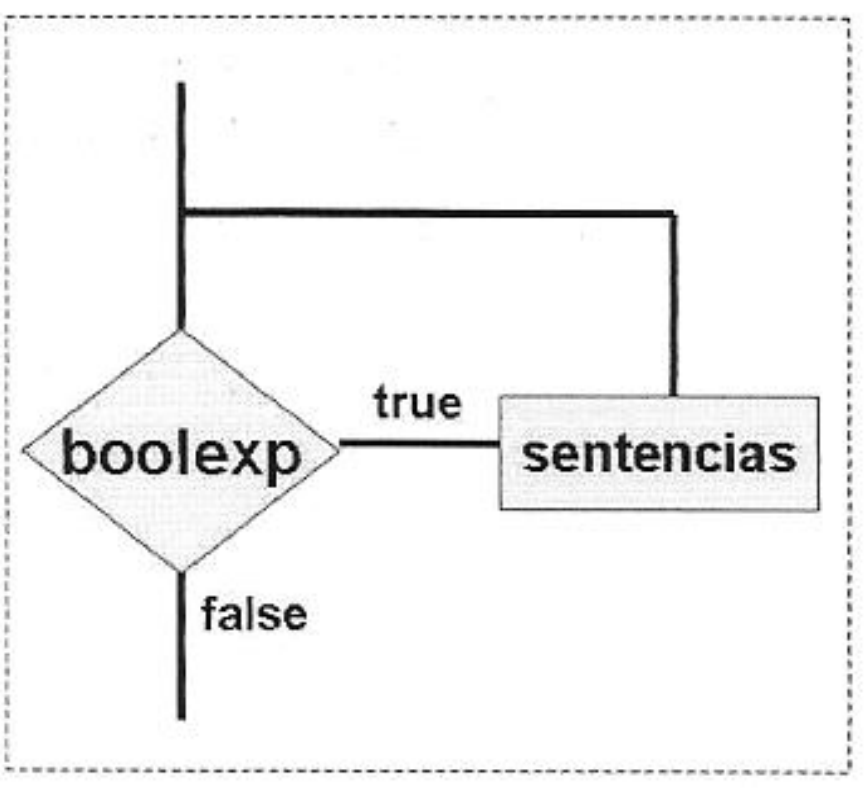
3.2. ESTRUCTURAS DE REPETICIÓN.

3.2.1. BUCLE WHILE.

El bucle WHILE se utiliza cuando se tiene que ejecutar un grupo de sentencias un número determinado de veces (0 o más veces).

El formato es el siguiente:

```
while (expresion_booleana) {

    sentencias;

}
```



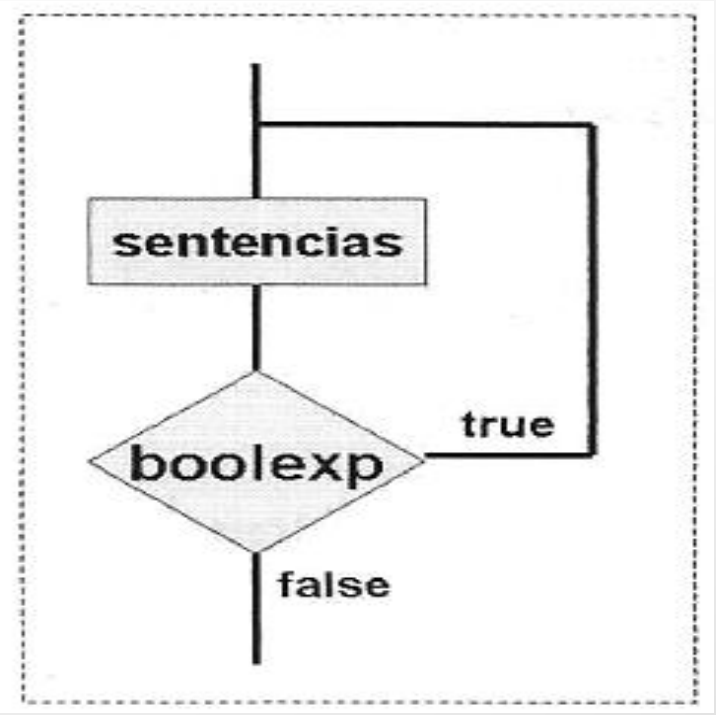
3.2. ESTRUCTURAS DE REPETICIÓN.

3.2.2. BUCLE DO WHILE.

El bucle DO WHILE es similar a la anterior, lo único que ocurre es que la comprobación se hace al final del bucle, con lo cual siempre se ejecutarán las sentencias al menos una vez.

El formato es el siguiente:

```
do {
    sentencias;
} while (expresion_booleana);
```



3.2. ESTRUCTURAS DE REPETICIÓN.

3.2.2. BUCLE DO WHILE.

Ejemplos:

```
int contador=1;

while (contador<=10) {

    System.out.println(contador);
    contador++;

}
```

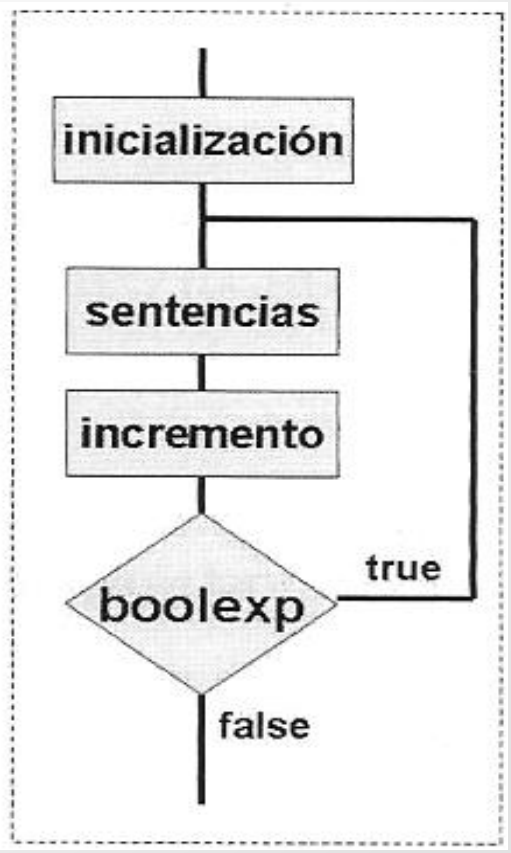
```
int contador=1;

do {

    System.out.println(contador);
    contador++;

} while (contador<=10);
```

3.2. ESTRUCTURAS DE REPETICIÓN.
3.2.3. BUCLE FOR.

El bucle FOR se utiliza cuando se necesita ejecutar una serie de sentencias un número fijo y conocido de veces.

El formato es el siguiente:

```
for ( inicializacion; expresion_booleana; incremento ) {
    sentencias;
}
```



Todo bucle FOR se puede conseguir mediante un bucle WHILE.

3.2. ESTRUCTURAS DE REPETICIÓN.

3.2.3. BUCLE FOR.

Ejemplos:

```
for (int contador=1; contador<=10; contador++) {

    System.out.println(contador);

}

for (int i=1; i<=10; i++)
    for (int j=1; j<=10; j++)
        System.out.println("("+i+","+j+")");

for (int i=10; i>=1; i--)
    for (int j=1; j<=10; j+=2)
        System.out.println("("+i+","+j+")");
```

3.3. ESTRUCTURAS DE SALTO.

Se desaconseja el uso de las sentencias "break" y "continue" salvo la sentencia "break" para la estructura switch. Las sentencias de salto dificultan la legibilidad de los programas y evitan que la programación sea estructurada.

La sentencia "break" sirve tanto para las estructuras de selección como para las estructuras de repetición. El programa al encontrar dicha sentencia se saldrá del bloque que está ejecutando.

La sentencia "continue", por el contrario, solo se utiliza en las estructuras de repetición y lo que hace es terminar la iteración i y continuar con al iteración i+1.

Las sentencias "break" y "continue" con etiquetas mantienen el mismo funcionamiento salvo que el programador puede controlar qué bucle es el que deja de ejecutar o en qué bucle continua.

La sentencia "return" es otra forma de salir de una estructura de control. La diferencia con las anteriores es que sale de la función que está ejecutando y permite devolver un valor.

3.3. ESTRUCTURAS DE SALTO.

Ejemplos:

```
int i=0;
while (i<10) {
     i++;
     if (i==5) break;
     System.out.println(i);
}
```

```
int j=0;
while (j<10) {
     j++;
     if (j==5) continue;
     System.out.println(j);
}
```

```
int i=0;
bucleext:
while (i<10) {
     i++;
     for (int j=0;j<i;j++) {
          System.out.print("*");
          if (i==5) break bucleext;
     }
     System.out.println("");
}
```

3.4. FUNCIONES.

Un método (o función o procedimiento) es un conjunto de instrucciones reutilizables del cual se puede hacer un uso continuo que se encuentra disponible en cualquier momento dentro de la aplicación.

Sirven para la optimización y reutilización del código de un programa.

El formato es el siguiente:

```
tipo_resultado nombre_funcion ( tipo_parametro nombre_parametro, ...) {

    sentencias;

}
```

La palabra "método" se utiliza en terminología de orientación a objetos y puede ser una función o un procedimiento. La diferencia entre una función y un procedimiento es que los procedimientos no devuelven nada y las funciones sí. En Java se especifica con la palabra clave "void".

3.4. FUNCIONES.

Una función consta de:

tipo_resultado: es el tipo de dato del valor devuelto por la función. Para devolver un dato se utiliza la palabra clave “return”.

nombre_funcion: es el nombre de la función y se utilizará para hacer referencia a ella y así, poder ser llamada desde otra zona de nuestro programa.

tipo_parametro: es el tipo de dato del parámetro que se le pasa a la función.

nombre_parametro: es el nombre del parámetro que se le pasa a la función. Una función puede tener uno, varios o ningún parámetro.

```
tipo_resultado nombre_funcion ( tipo_parametro nombre_parametro, ...) {

    sentencias;

}
```

3.4. FUNCIONES.

Ejemplos:

```
void metodo1(int i, int j) {

    System.out.println("Valor parametro i: "+i);
    System.out.println("Valor parametro j: "+j);
}

int funcion1(int i, int j) {

    return i+j;
}

int funcion2() {

    metodo1(1,2);
    return funcion1(1,2);
}
```

3.4. FUNCIONES.

Ejemplo:

```
import java.util.*;

public class MenuconFunciones {

    private static void opcion(int op) {
        System.out.println("Has seleccionado la opcion "+op);
        return;
    }

    private static int mostrarMenu() {
        int op;
        do {
            System.out.println("Seleccionar una opcion:");
            System.out.println("----------------------");
            System.out.println();
            System.out.println("1.- Opcion primera.");
            System.out.println("2.- Opcion segunda.");
            System.out.println("3.- Opcion tercera.");
            System.out.println();
            System.out.print("Opcion? ");
            Scanner sc=new Scanner(System.in);
            op=sc.nextInt();
            System.out.println();
        } while (op<0 || op>3);
        return op;
    }

    public static void main(String[] args) {
        int opt=mostrarMenu();
        switch (opt) {
            case 1: /*Aquí puede haber más código*/ break;
            case 2: /*Aquí puede haber más código*/ break;
            case 3: /*Aquí puede haber más código*/ break;
        }
        opcion(opt);
        System.out.println("");
    }

}
```

3.5. CLASE MATH.

La clase Math, perteneciente al paquete *java.lang*, dispone de funciones para realizar operaciones numéricas básicas como:

- Funciones trigonométricas.
- Funciones exponenciales y logarítmicas.
- Funciones de potencia y raíz cuadrada.
- Funciones de aproximación a un número decimal.
- Funciones de mayor y menor.
- Funciones aleatorias.
- …

Esta clase contiene constantes (E, PI) y funciones estáticas por lo que para invocarlas se realiza de una de las siguientes formas (método o función):

Math.nombre_función (parámetros)
tipo_variable nombre_variable = Math.nombre_función (parámetros)

3.5. CLASE MATH.

Ejemplos:

int abs (int x)
double ceil (double x)
double cos (double x)
double exp (double x)
double floor (double x)
double log (double x)
int max (int x, int y)
int min (int x, int y)
double pow (double x, double y)
double random()
long round (double x)
double sin (double x)
double sqrt (double x)

FIN