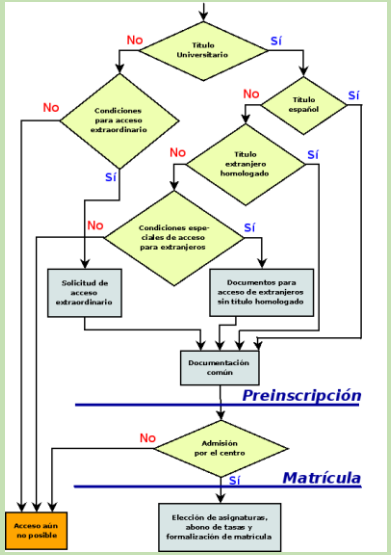
Capítulo 4
Control de flujo

Continuar

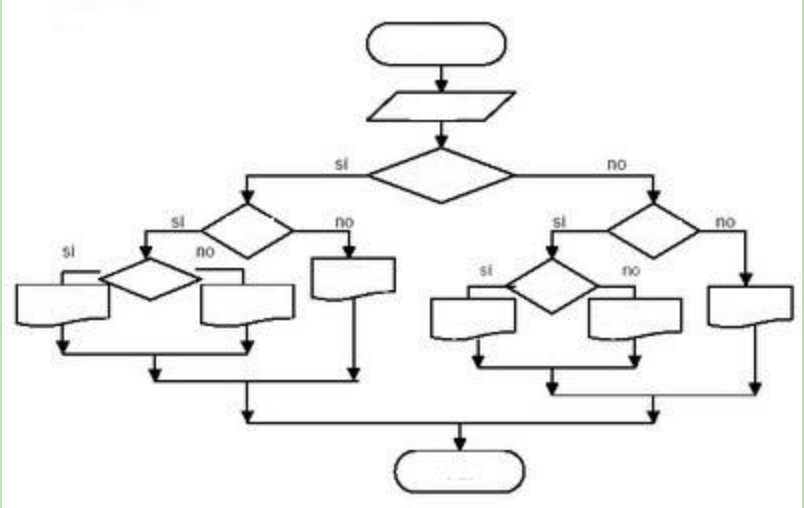
Introducción

El control de flujo permite encausar a la computadora sobre la ruta que debe seguir al momento de la ejecución de un programa, para ello se apoya en las estructuras de control que le permitirán tomar decisiones, repetir la ejecución de un grupo de instrucciones, subdividir los programas grandes en pequeños programas que realizan una actividad determinada. Se verá la segmentación y control de programas en subprogramas más pequeños llamados métodos y/o funciones, como una manera de preparar el terreno para atacar sistemas de programación más complejos en el momento en que se requiera.



Estructuras selectivas: simple, doble y múltiple

La complejidad de los problemas hacen que difícilmente los programas sean solamente una estructura secuencial, muchas veces es necesario ejecutar una instrucción o bloque de instrucciones dependiendo de si se cumple o no una condición. Las preguntas se plantean por medio de condiciones estructuradas en forma lógica y el resultado será verdadero o falso pero no ambos a la vez, en función del resultado será el bloque de acciones a ejecutar. Las instrucciones selectivas se clasifican en: simples, dobles y múltiples.



Instrucción selectiva simple

Se dice que la instrucción selectiva es simple si al cumplirse la condición se ejecuta una instrucción o bloque de instrucciones y en caso de ser falsa no se ejecuta acción alguna. El formato general de una instrucción selectiva simple es:

if (condición)

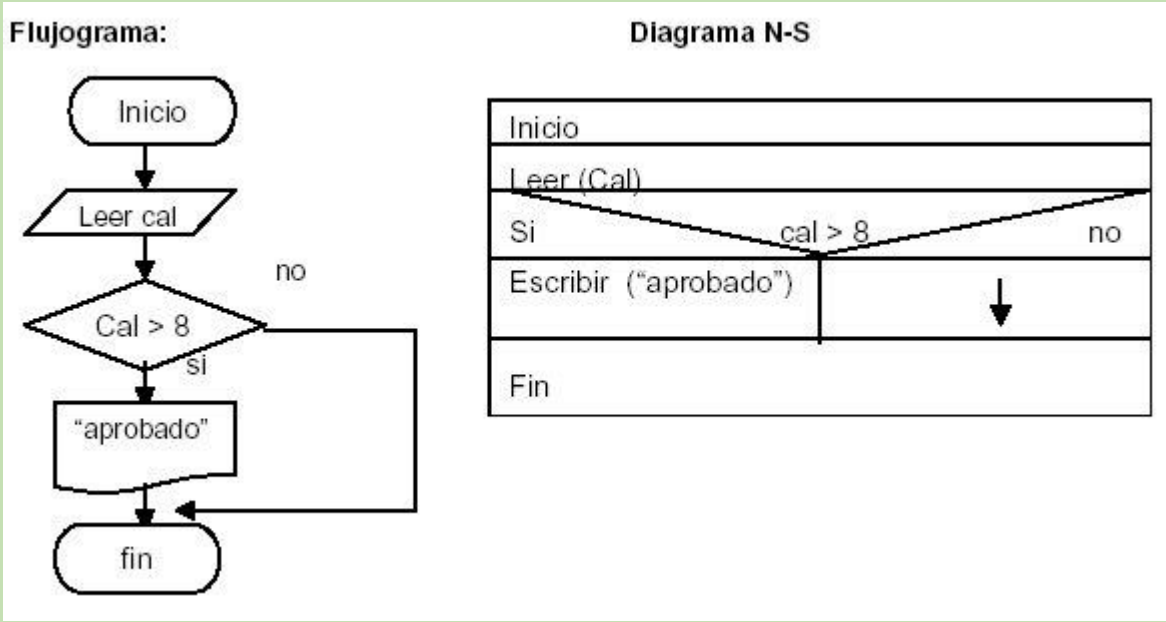
{

……

instrucciones.

……

}



Instrucción selectiva doble (if-else)

La estructura selectiva doble se utiliza cuando se tienen dos opciones de acción. Con la bifurcación doble se ejecuta un bloque de instrucciones A si se cumple la condición o bien se ejecuta el bloque de instrucciones B en caso de que no se cumpla.

if (condición) {

……

Instrucciones A

……

}

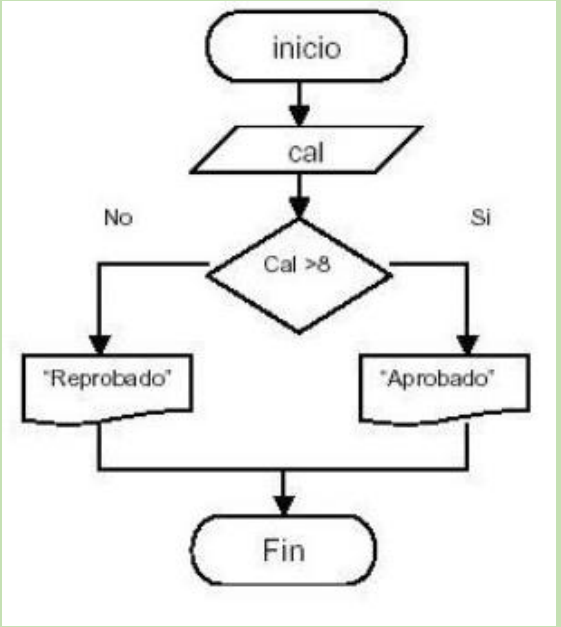
else {

……

Instrucciones B

……

}



Instrucción condicional múltiple (switch)

Instrucciones condicionales anidadas donde se tiene que escribir mucha información en cada línea, porque las instrucciones selectivas anidadas ocupan mucho espacio. Java tiene la instrucción de selección múltiple “switch” que permite sustituir estos ifs anidados por una estructura fácil de entender y usar.

Donde:

variable: Es una variable o expresión cuyo valor es un dato: entero, byte, short o char.

valor_1, valor_2,..,valor_n: Son valores de la variable para cada uno de los casos posibles (case).

break: Instrucción que tiene la finalidad de romper la ejecución del flujo.

default: En caso de que no se ejecuten ninguno de los casos previstos se ejecutarán las instrucciones por default.

Estructuras iterativas: mientras, repetir, desde

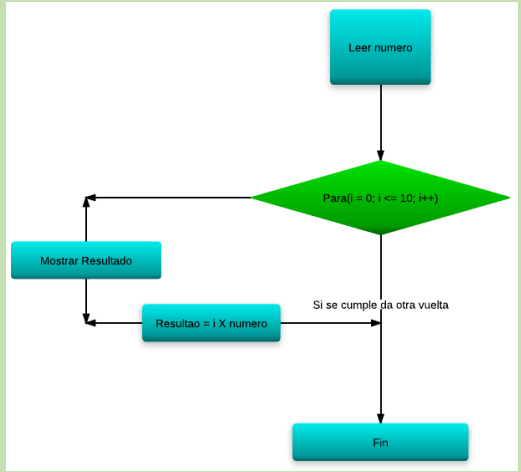
Una estructura iterativa permite ejecutar una instrucción o un bloque de instrucciones varias veces. En el lenguaje Java se tienen fundamentalmente las siguientes tres estructuras iterativas:

a) Mientras (**while**).

b) Repetir (**do-while**).

c) Desde (**for**).

Las tres permiten ejecutar cierto número de veces un bloque de instrucciones, pero tienen pequeñas diferencias que es conveniente analizar.



Mientras (while)

Con **while** se ejecuta el bloque de instrucciones encerradas entre llaves, mientras se cumpla la condición establecida. Su formato general es el siguiente:

while (condición) {

…….

Instrucciones;

……

}

Una característica de **while** es que si en la primera vez no se cumple la condición no se ejecutará ninguna vez el bloque de instrucciones encerradas entre llaves, debido a que la condición está al principio del bloque.

Repetir (do-while)

Con el ciclo do-while se ejecuta el bloque de instrucciones que se encuentran encerradas entre las llaves de la palabra *do* y la palabra while con la condición correspondiente. El formato general es el siguiente:

do {

……

Instrucciones;

……

}**while** (condición);

Desde (for)

La instrucción for requiere de una variable que tiene la finalidad de controlar el número de veces que se ejecuta el bloque de instrucciones. En “**inicio**” se coloca el valor con el cual inicia la variable. En “**condición**” se indica la proposición que deberá ser verdadera para que el ciclo continúe, en caso de ser falsa saldrá de dicho ciclo y finalmente en “**actualización**” se coloca una instrucción que tendrá la finalidad de modificar el valor de la variable con la finalidad de que después de ejecutar varias veces el bloque de instrucciones que están encerradas entre llaves, la condición sea falsa.

for (inicio; condición; actualización) {

……

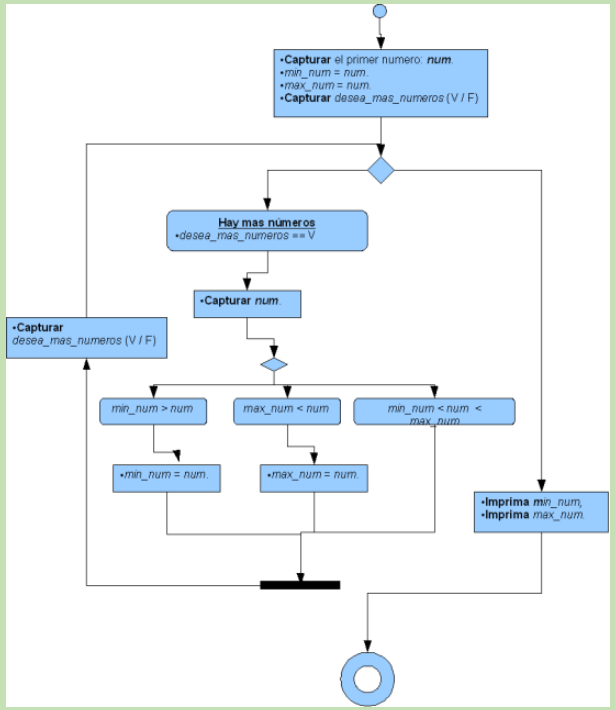
Instrucciones;

……

}

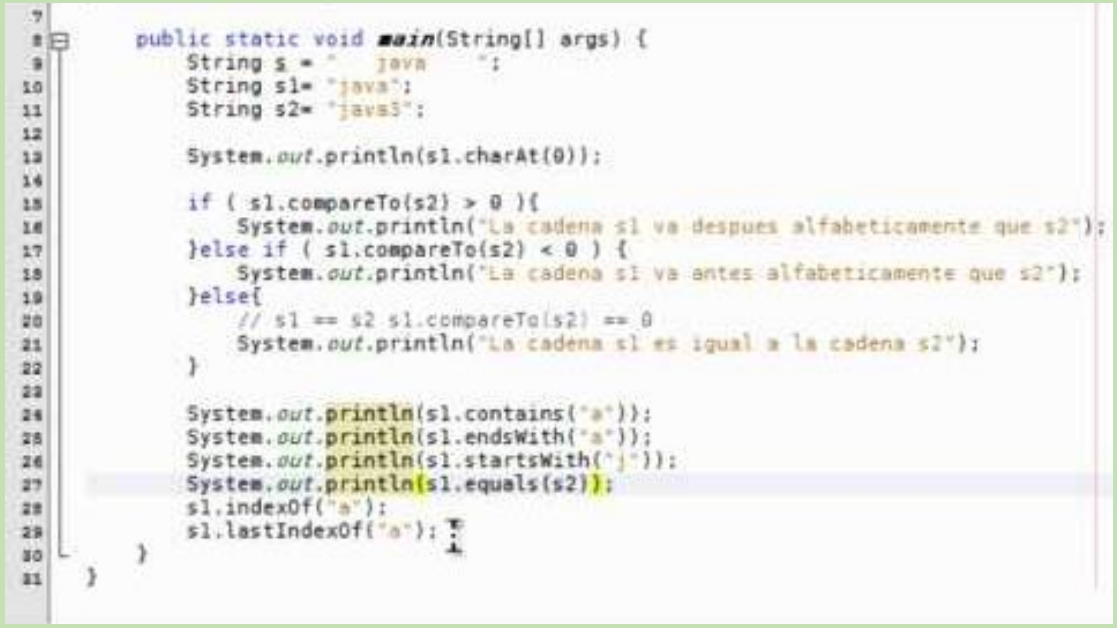
Ciclos anidados

En un programa es posible tener una instrucción dentro de otra y por lo tanto puede haber también instrucciones iterativas dentro de otra instrucción iterativa, esto permite crear programas con mayor potencia.



Manipulación de cadenas

Es posible llevar a cabo varias operaciones con los métodos de la clase String (cadena). El lenguaje Java tiene muchos más métodos que permiten la manipulación de cadenas. Considerar que cad1, cad2 y cad3 son datos tipo String.



```
7
8       public static void main(String[] args) {
9           String s = "   java    ";
10          String s1= "java";
11          String s2= "java3";
12
13          System.out.println(s1.charAt(0));
14
15          if ( s1.compareTo(s2) > 0 ){
16              System.out.println("La cadena s1 va despues alfabeticamente que s2");
17          }else if ( s1.compareTo(s2) < 0 ) {
18              System.out.println("La cadena s1 va antes alfabeticamente que s2");
19          }else{
20              // s1 == s2 s1.compareTo(s2) == 0
21              System.out.println("La cadena s1 es igual a la cadena s2");
22          }
23
24          System.out.println(s1.contains("a"));
25          System.out.println(s1.endsWith("a"));
26          System.out.println(s1.startsWith("j"));
27          System.out.println(s1.equals(s2));
28          s1.indexOf("a");
29          s1.lastIndexOf("a");
30      }
31  }
```

Diseño e implementación de funciones y métodos

Un método es un bloque de instrucciones que tiene un nombre, que opcionalmente puede recibir parámetros o argumentos, que se utiliza para realizar algo y que opcionalmente puede devolver un valor de algún tipo de dato conocido. La sintaxis de los métodos es:

modificador tipo nombre(*tipo* par1, *tipo* par2,…, *tipo* parn) {

.....

Instrucciones

.....

}

Funciones

Cuando un método recibe uno o varios argumentos y devuelve un solo valor recibe el nombre de función. Las funciones tienen el mismo formato que los métodos, porque son métodos con ciertas características, pero adicionalmente deben tener la línea return para regresar el valor. La función tiene el modificador static lo cual permite que la función sea reconocida en todos los métodos y clases de programa sin necesitar de un objeto.

```
24
25 import java.util.*;
26 import java.lang.*;
27
28 class Rextester {
29
30     public static void main(String args[]) {
31         int n = 6;
32         System.out.println("Cálculo del factorial de " + n);
33         System.out.println(n + "! = " + factorial(n));
34     }
35
36     // función recursiva para el cálculo del factorial de n
37     public static int factorial(int n) {
38       if(n==0) return 1; // caso base
39       else return n*factorial(n-1); // fórmula recursiva
40    }
41 }
```

Métodos

Los métodos se usan en la programación para dividir un programa grande en pequeños subprogramas con la finalidad de hacer más claros y sencillos los sistemas, aprovechando lo que realiza un método en el momento y lugar del programa cuando sea requerido.

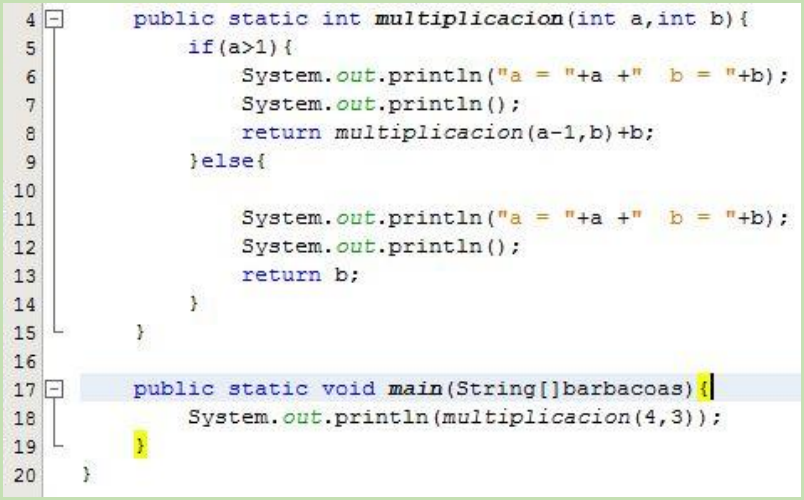
```
1  public class Calculos{
2
3      public int Suma(int a, int b){
4          return a + b;
5      }
6
7      public double Suma(double a, double b){
8          return a + b;
9      }
10
11     public long Suma(long a, long b){
12         return a + b;
13     }
14
15 }
```

Recursividad

La recursividad es una técnica de programación en donde un método se llama a sí mismo desde su propio código. La recursividad y la iteración están muy relacionadas, ya que un programa recursivo se puede escribir en forma iterativa. Todos los métodos recursivos se pueden programar de forma iterativa, aunque algunas veces la solución es más complicada, sin embargo no todos los métodos iterativos se pueden resolver en forma recursiva ya que deben reunir las siguientes características:

a) Deben tener uno o más puntos base o retorno.

b) Tienen un caso recursivo que permite llamarse a si mismo.



```
4      public static int multiplicacion(int a,int b){
5          if(a>1){
6              System.out.println("a = "+a +"  b = "+b);
7              System.out.println();
8              return multiplicacion(a-1,b)+b;
9          }else{
10
11             System.out.println("a = "+a +"  b = "+b);
12             System.out.println();
13             return b;
14         }
15     }
16
17     public static void main(String[]barbacoas){
18         System.out.println(multiplicacion(4,3));
19     }
20 }
```