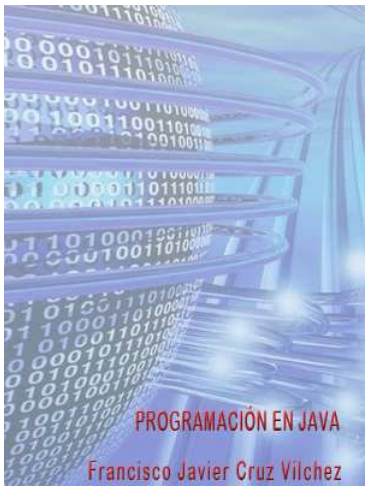


Programación en Java
Francisco Javier Cruz Vílchez
ISBN-13: Pendiente

PROGRAMACIÓN EN JAVA

Autor
Francisco Javier Cruz Vílchez

Indice

Capitulo I
Estructuras Secuenciales ……. …….01

Capitulo II
Control de ejecución ……... ……22

Capitulo III
Arreglos y Cadenas …….. 49

Capitulo IV
Clases y Métodos. ……. 70

Bibliografía ……. 94

CAPITULO 1
ESTRUCTURAS SECUENCIALES

OBJETIVOS :

Al finalizar el capitulo el alumno deberá aprender :

- **Que es java y definir claramente el concepto de maquina virtual.**
- **Aprenderá a definir un programa en java y sus componentes.**
- **Manejara los operadores aritméticos como booleanos.**
- **Planteara soluciones para programas de estructura secuencial**

¿QUÉ ES JAVA?

El significado de java, tal y como se le conoce en la actualidad, es el lenguaje de programación y un entorno de ejecución de programas escritos en java. Al contrario de los compiladores tradicionales , que convierten el código fuente en instrucciones a nivel de máquina, el compilador java traduce el código fuente java en instrucciones que son interpretadas por la maquina virtual de java (JVM, Java Virtual Machine). A diferencia de C y C++ en los que está inspirado. Java es un lenguaje interpretado.

Aunque hoy en día java es por excelencia el lenguaje de programación para Internet y la World Wide Web en particular, java no comenzó como proyecto Internet y por las circunstancias es idóneo para las tareas de programación de propósito general y, de hecho muchas de las herramientas java están escritas en java.

Características de Java

Java es un lenguaje interpretado. Cuando se escriben programas en java, bien en un entorno de desarrollo o un editor de texto necesita ser compilado en un conjunto de instrucciones optimizadas denominadas programas "*bytecode*". Este programa es independiente de la plataforma y no se puede ejecutar directamente por procesador. En su lugar, una máquina virtual java ejecuta (interpreta) los bytecode. Existen numerosas JVM disponibles para una gran variedad de plataformas que permiten a los programas Java ser independientes de la plataforma.

Por ejemplo, un programa java compilado en una estación de trabajo UNIX puede ejecutarse en un Macintosh o un Terminal de Windows xp.

Otra fortaleza del java proviene de sus bibliotecas incorporadas. Los paquetes que vienen con el entorno de desarrollo java contienen muchos centenares de clases integradas, con muchos millares de métodos.

Especificaciones del lenguaje Java

Los lenguajes de computadoras tienen reglas estrictas de uso que deben seguirse cuando se escriben programas con el objeto de ser comprendidos por la computadora.

La especificación es una definición técnica del lenguaje que incluye sintaxis, estructura y la interfaz de programación de aplicaciones que contienen clases predefinidas. El lenguaje evoluciona rápidamente y el mejor lugar para consultar las últimas versiones y actualizaciones del mismo se encuentra en el sitio Web de Internet de Sun.

- www.sun.com
- www.javasoft.com

Cómo crear un programa

Todos los programas en java minimamente deben tener una clase y un método de entrada y salida de la aplicación.

Por ejemplo en el siguiente caso que deseamos imprimir un mensaje en pantalla vamos a definir una clase que le denominamos saludo.

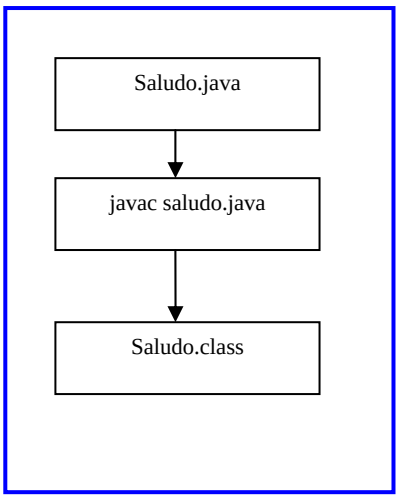
Nuestro fichero deberá tener el mismo nombre de la clase para que este pueda ser interpretado por la máquina virtual y con la extensión ***java***

```
class  saludo{
  public static void main (String [] args){
      System.out.println(“.....Saludos.....”)
    }
}
```

En java toda aplicación debe tener un punto de inicio de la aplicación esto se da con el método ***main()*** pero podemos tener clases en java que si no son invocadas desde el método ***main ()*** simplemente no funcionaran pero si pueden ser interpretadas por la máquina virtual de java.

Todas las clases después de ser interpretadas crean un fichero con la extensión ***class*** que es el puede ser interpretado por otra plataforma diferente a la que estamos trabajando.

Para el caso expuesto hemos creado una clase denominada ***saludo.java*** lo compilamos el fichero y se crea un fichero ***saludo.class*** la secuencia la mostramos en la figura siguiente.



Java es sensible a las letras mayúsculas y minúsculas como se muestra en el siguiente caso.

```
public class  Cerror
public  static void Main (String [] args){

   System.out.println("Hola Francisco …");

}
```

Como se muestra en el ejemplo cuando pretendemos interpretar con java las líneas de código lanzara mensajes de error dado que el ***main*** se encuentra mal escrito en dado que Main y main son dos cosas distintas en java .

Los nombres de las clases comienzan normalmente con una letra mayúscula y los nombres de métodos y variables con una letra minúscula.

Componentes de una Aplicación

En un programa se consideran los siguientes elementos:

- Comentarios
- Palabras Reservadas
- Modificadores
- Sentencias
- Bloques
- Clases
- Métodos
- Método *main*

- Comentarios: los comentarios se pueden dar en una sola linea se da con // donde se ignora todo el texto que se indica en ellas. Y cuando se quiere poner un comentario a un conjunto de lineas se hase indicando /* */

```
//Autor: Pio Nervo

/* Programas escrito
  *Por Pio nervo
*/
```

Usando comentarios :

```
//autor : anónimo
/* clase que calcula el área de un rombo
* dadas sus dos diagonales
*/
class CRombo {
 public static void main(String [] args){
 double d1,d2,Area;
            d1=13.56;
            d2=19;
            Area=(d1*d2)/2;
            System.out.println("El Area del Rombo para d1="+d1+" y d2 = "+d2);
            System.out.println("\tArea = "+Area);

            }
}
```

- **Palabras reservadas o palabras claves** : son palabras que tienen un determinado significado para el compilador y no se pueden ser utilizadas para otros fines. Por ejemplo la palabra while significa que se habrá de evaluar la expresión que viene a continuación hasta que se deje de cumplir una la condición.

- Modificadores : Existen en java palabras reservadas java denominadas modificadores que especifican las propiedades de los datos, métodos y clases, y cómo se pueden utilizar. Por ejemplo.

  public static, private final abstract protected

- Sentencias : Una sentencia representa una acción o una secuencia de acciones. Cada sentencia termina con un punto y coma por ejemplo;

```
double d1,d2,Area;//se definen tres variables
d1=13.56;          //se asigna el valor a la variable d1
d2=19;             // se asigna el valor a la variable d2
Area=(d1*d2)/2;  // se realiza un calculo que es asignado a la variable Area.

//se imprime el área en pantalla
System.out.println("El Area del Rombo para d1="+d1+" y d2 = "+d2);
```

Todas las sentencias terminan en un punto y coma.

- Bloque : Un bloque es una estructura de programa que agrupa sentencias. Los bloques comienzan con una llave de apertura ({) y terminan con una llave de cierre (}). Un bloque puede estar dentro de otro bloque.

Caso a

```
{  z=15;
   z=z+100;
   If(z>250){ z=z-5;}
}
```

Caso b

```
{int sum,x;
x=1;
suma=0;
   while (x<=10){
     sum +=x;
      x++;
   }
    System.out.println("La Suma es  : "+suma);
}
```

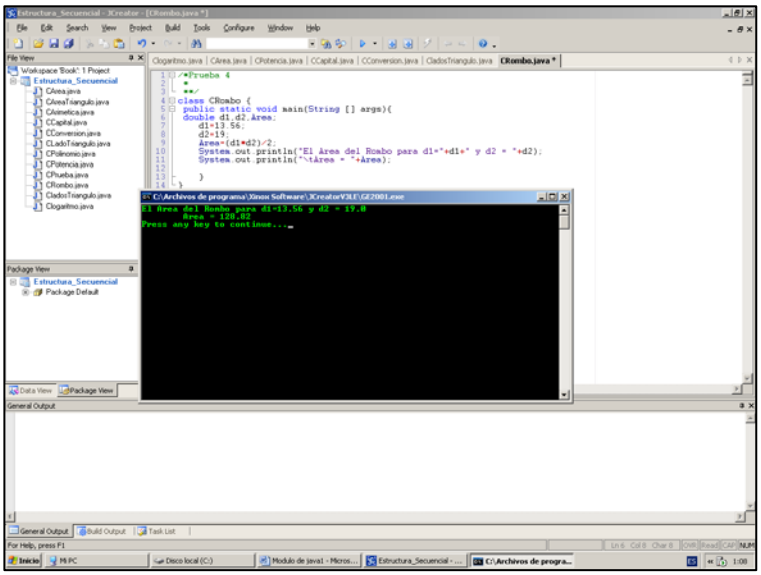
- Clases: La clase es la construcción fundamental de java y, como ya se ha comentado, constituye una plantilla o modelo para fabricar objetos. Un Programa consta de una o más clases y cada una de ellas puede contener declaraciones de datos y métodos. Cada clase java se compila en un archivo de bytecode con extensión class.

- Método : Un método es una colección de sentencias que realizan una serie de operaciones determinadas. Por ejemplo

```
System.out.println("El Area del Rombo para d1="+d1+" y d2 = "+d2);
```

Es un método que visualiza un mensaje en el monitor o consola. println está predefinida como parte del lenguaje estándar java el argumento se encuentra entre los paréntesis.

La aplicación anteriormente expuesta que calcula el área del rombo tendría la siguiente salida.



OPERADORES DE EXPRESIONES.

OPERADORES DE ASIGNACIÓN.

El operado = asigna el valor de la expresión derecha a la variable situada a su izquierda.

Codigo= 1000;

Nota=10.51;

Y =2*5*5+3*5+7;

Este operador es asociativo por la derecha, eso permite realizar asignaciones múltiples. Así:

A=B=C=45;

Además el operador de asignación =, java proporciona cinco operadores de asignación adicionales como se muestra en la tabla a continuación.

Estos operadores de asignación actúan como una notación abreviada para expresiones utilizadas con frecuencia. Así, por ejemplo, si se desea multiplicar 10 por i se puede escribir ;

i=i*10; equivale a escribir i *=10;

Símbolo	Uso	Descripción	Equivalencia

=	a = b	Asigna el valor de b a a	a=b
*=	a * =b	Multiplica a por b y asigna el resultado a la variable a.	a=a*b
/=	a /=b	Divide a entre b y asigna el resultado a la variable a.	a= a/b
%=	a %=b	Fija a el resto a/b	a= a%b
+=	a +=b	Suma b y a y lo asigna a la variable a.	a=a+b
-=	a - =b	Resta b de a y asigna el resultado a la variable a	a=a-b

OPERADORES DE ARIMÉTICOS.

Los operadores aritméticos sirven para realizar operaciones aritméticas básicas. Los operadores aritméticos java siguen las reglas algebraicas típicas de jerarquía o prioridad. Estas reglas especifican la precedencia de las operaciones aritméticas.

Por ejemplo si queremos multiplicar **a** por la cantidad **b+c** , escribimos:

a*(b+c)

Operador aritmético	**Expresión algebraica**	Expresión En Java
+	f + 7	f + 7
-	p - c	p - c
*	bm	b*m
/	x/y o x y	x / y
%	r mod s	r % s
Operadores Aritméticos		

Operador	Orden de Evaluación
( )	Se evalúan primero. Si los paréntesis están anidados, la expresión dentro del par más interno se evalúa primero. Si hay varios paréntesis *"en el mismo nivel"* (es decir, no anidados), se evalúan de izquierda a derecha
*, / ó %	Se evalúan en segundo lugar. Si hay varios, se evalúan de izquierda a derecha.
+ ó -	Se evalúan al último. Si hay varios, se evalúan de izquierda a derecha

A continuación presentamos el caso del cálculo de la media aritmética (promedio) de cinco términos:

$$\begin{array}{|l} Algebra: M = \dfrac{a+b+c+d}{5} \\ java: M = (a+b+c+d+e)/5; \end{array}$$

Los paréntesis son necesarios porque la división tiene mayor precedencia que la suma. Necesitamos dividir la cantidad completa (a+b+c+d+e) entre 5 . Si omitimos erróneamente los paréntesis, obtendremos a+b+c+d+e/5, que se evalúa así:

El siguiente es un ejemplo de la ecuación de una línea recta:

$$\boxed{\begin{array}{l} Álgebra: y = mx + b \\ java: y = m * x + b; \end{array}}$$

No necesitamos paréntesis. La multiplicación se aplica primero porque tiene mayor precedencia que la suma.

El siguiente ejemplo contiene operaciones de residuo, multiplicación, división, suma y resta:

$$\boxed{\begin{array}{l} Álgebra: z = pr\%q + w/x - y \\ java: z = p * r\%q + w/x - y; \end{array}}$$

Uso de paréntesis
(7 * (10 – 5) % 3) * 4 + 9

La subexpresión (10 – 5) se evalúa primero, produciendo (7 * 5 % 3)*4+9

A continuación se evalúa de izquierda a derecha la sub expresión (7*5 %3)

(35 % 3)*4+9
Seguida de
2 * 4 + 9
Se realiza a continuación la multiplicación, obteniendo

8+9
Y la suma produce el resultado final
17

OPERADORES DE INCREMENTACIÓN Y DECREMENTACIÓN

Incremento	Decremento
++n	- -n
n +=1	n - =1
n = n+1	n= n - 1
Operadores de incremento ++ y decremento - -	

Decir a++ es igual a =a+1. Las sentencias.
++n;
n++;
tiene el mismo efecto; así como

- -n ;
n- - ;

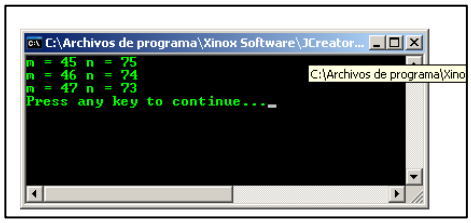
```
int a=1, b;
b=a++; //b vale 1 y a vale 2
```

```
int a=1,b;
b=++a; //b vale 2 y a vale 2
```

Demostración de incremento y decremento con el la siguiente clase de java.

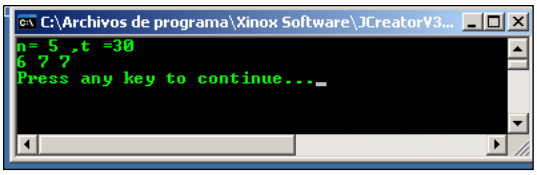
```
class Incremento {
        public static void main(String [] args){
        int m=45, n=75;
        System.out.println("m = "+m +" n = "+n);
        ++m;
        --n;
        System.out.println("m = "+m +" n = "+n);
        m++;
        n--;
        System.out.println("m = "+m +" n = "+n);
        }
}
```

Salida del programa anterior.




```
class OrdenOut {
        public static void main(String [] args){
        int n=5, t;
        t=++n*--n;
        System.out.println("n= "+n+" ,t ="+t) ;
        System.out.println(++n+ " "+ ++n +" "+ n++);

        }
}
```




```
class CPrueba {
    public static void main(String [] args){
        int a=10,b=3,c=1,d,e;
        float x,y;
        x=a/b;
        System.out.println("el valor de b Inicial : "+b);
        d=a+b++;
        System.out.println("el valor de b Final   : "+b);
        e=++a - b;
        y =(float)a/b;
        System.out.println("El valor de x ="+x);
        System.out.println("El Valor de d ="+d);
        System.out.println("El Valor de e ="+e);
        System.out.println("El Valor de y ="+y);
```

```
    }
}
```

OPERADORES LÓGICOS

Operador	Operación lógica	Ejemplo
&& o &	Operando1 && Operando2	m<n && i>j
|| o |	Operando1 || Operando2	m = 5 || n! = 10
!	no lógica	!(x > = y)
^	Operando1 ^| Operando2	x < n ^ n > 9

Los operadores lógicos de Java son : not (!) and (&&), or(||) y or exclusivo(^).

1) El operador lógico (not no) produce falso si su operando es verdadero y viceversa.
2) El operador lógico && (and) produce verdadero sólo si ambos operandos son verdaderos.
3) El operador lógico || (or o) produce verdadero si cualquiera de los operandos es verdadero y produce falso sólo si ambos operandos son falsos.
4) El operador lógico ^ (or exclusivo) produce verdadero si ambos operandos son distintos y produce falso sólo si ambos operandos son iguales .

Ejemplos

1. if ((a<b) && (c>d)){
   System.out.println("Los resultados no son válidos");
   }

2. if((ventas >50000) || (horas<100)){ prima=100000;}
   Si la variables ventas es mayor a 50000 o bien la variable horas es menor que 100, entonces asignar a la variable prima el valor 100000.

3. if ((x<5)^(b<19))
   {
   System.out.println("Par de valores no válidos");
   }

   Se visualiza el mensaje Par de valores no válidos para valores de x menores que 5 y de b mayores o igual que 19; o bien, para valores de x mayores o igual que 5 y de b menores que 19

EJERCICIOS RESUELTOS

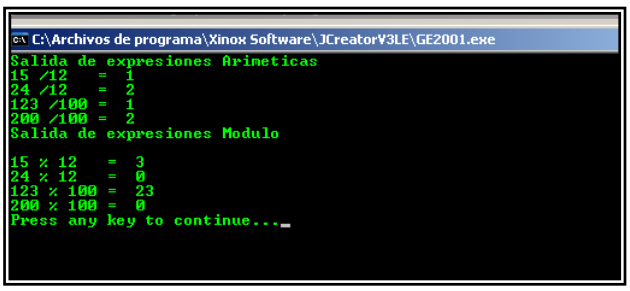
1.- Determinar el valor de las siguientes expresiones

15/12	15%12
24/12	24 % 12
123/100	123 % 100
200/100	200 % 100

Para ver la solución de los casos anteriormente planteados definiremos una clase denominada COperacion001.java.

```
class COperacion001 {
    public static void main(String [] args ){
            System.out.println("Salida de expresiones Arimeticas");
            System.out.println("15 /12   =  "+15/12);
            System.out.println("24 /12   =  "+24/12);
            System.out.println("123 /100 =  "+123/100);
            System.out.println("200 /100 =  "+200/100);

            System.out.println("Salida de expresiones Modulo\n");
            System.out.println("15 % 12   =  "+15%12);
            System.out.println("24 % 12   =  "+24%12);
            System.out.println("123 % 100 =  "+123%100);
            System.out.println("200 % 100 =  "+200%100);
    }
}
```



2.- Escribir un programa que determine la suma de la suma de las cifras de un entero positivo de 4 cifras.

```
import java.io.*;
class CSumaCifras {
        public static void main(String[] args) throws IOException
        {int num,suma, millares, centenas,unidades, decenas, cociente;

BufferedReader entrada =new BufferedReader(
                                new InputStreamReader(System.in));
System.out.print("Ingrese un Numero entero de Cuatro Cifras :\t");
System.out.flush();
num=Integer.parseInt(entrada.readLine());
System.out.println();
```

```
unidades=num % 10;
cociente=num/10;
decenas=cociente % 10;
cociente = cociente/10;
centenas= cociente %10;
millares=cociente/10;
suma=unidades+decenas+centenas+millares;
System.out.println("La suma de los digitos  es "+suma);
}         }
```

3.- Una temperatura Celsius (Centígrados) puede ser convertida a una temperatura equivalente F de acuerdo a la siguiente fórmula.

$$f = \frac{9}{5} * c + 32$$

```
import java.io.*;
class CGrados {
     public static void main(String[] args ) throws IOException{
      double c,f,aux;
      c=0;f=0;aux=0;
      BufferedReader entrada =new BufferedReader(
                                                    new InputStreamReader(System.in));
      Double G;
      System.out.print("Ingrese Los Grados Centigrados :\t");
      System.out.flush();
      G=Double.valueOf(entrada.readLine());
      c=G.doubleValue();
      aux=(double)9/5;
      f=(aux*c)+32;
      System.out.println("Imprimiendo la Variable Auxiliar "+aux);
      System.out.println("El Equivalente de los : "+c+" Grados Centigrados a Farenthei : "+f);

  }
}
```

4.- Un sistema de ecuaciones lineales
ax+by=c
dx+ey=f

se puede resolver con las siguientes fórmulas:

$$x = \frac{ce - bf}{ae - bd} \qquad y = \frac{af - cd}{ae - bd}$$

```
import java.io.*;
class CEcuacionesLineales {
public static void main (String[] args) throws IOException{
 double a,b,c,d,e,f,x,y;
 BufferedReader entrada =new BufferedReader(
                                  new InputStreamReader(System.in));
        System.out.flush();
        Double A,B,C,D,E,F;

        System.out.print("Ingrese el Valor de A :\t");
        A=Double.valueOf(entrada.readLine());
        a=A.doubleValue();
        System.out.println();

        System.out.print("Ingrese el Valor de B :\t");
        B=Double.valueOf(entrada.readLine());
        b=B.doubleValue();
        System.out.println();

        System.out.print("Ingrese el Valor de C :\t");
        C=Double.valueOf(entrada.readLine());
        c=C.doubleValue();
        System.out.println();

        System.out.print("Ingrese el Valor de D :\t");
        D=Double.valueOf(entrada.readLine());
        d=D.doubleValue();
        System.out.println();

        System.out.print("Ingrese el Valor de E :\t");
        E=Double.valueOf(entrada.readLine());
        e=E.doubleValue();
        System.out.println();

        System.out.print("Ingrese el Valor de F :\t");
        F=Double.valueOf(entrada.readLine());
        f=F.doubleValue();
        System.out.println();
        x=(c*e-b*f)/(a*e-b*d);
```

```
        y=(a*f-c*d)/(a*e-b*d);
        System.out.println("IMPRIMIENDO LOS VALORES DE X y Y ");
        System.out.println("Valor de X es : "+x);
        System.out.println("Valor de Y es : "+y);
      }
    }
```

5.- Solución que determina el valor de un polinomio en función de x ,a,b,c

```
        class CPolinomio {
         public static void main(String [] args){
         double a,b,c,x;
         double total;
                a=5;
                b=-1.7;
                c=2;
                x=10.5;
                total= a*x*x*x+b*x*x-c*x+3;

                System.out.println("El Polinomio a*x*x*x+b*x*x-c*x+3");
                System.out.println("El Total del Polinomio es = "+total);    }
}
```

6.- Solución que calcula el área de un triangulo en función de sus lados

```
        class CAreaTriangulo {
        public static void main(String[] args){
         double a,b,c,p;
         double AreaTriangulo;
         a=4;
         b=3;
         c=5;
         p=(a+b+c)/2;
         System.out.println("El Area del Triangulo es ");
         System.out.println("\ta= "+a);
         System.out.println("\tb= "+b);
         System.out.println("\tc= "+c);
         AreaTriangulo=Math.sqrt(p);
         System.out.println("\n\n\tEl Area del Triangulo es :"+AreaTriangulo);}
         }
```

7.- Solución dados dos lados de un triángulo y el ángulo entre ellos se calcula el tercer lado.

```
        class CLadoTriangulo {
                public static void main(String[] args){
                double b,c, angulo;
                double a;
```

```
            b=18;
            c=15;
            angulo=34;
            a=Math.sqrt(b*b+c*c -2*b*c*Math.cos(angulo));
            System.out.println("La longitud del lado del Triangulo : "+a);
            }
    }
```

8.-Calcula el Monto a devolver si nos prestan un capital c, a una tasa de interés t% durante n periodos

$$m=c(1+i)^2$$

```
import java.io.*;
class CCapital {
public static void main(String [] args) throws IOException
    {
            double c,i,n,m;
            BufferedReader entrada =new BufferedReader(
                                                new InputStreamReader(System.in));
            System.out.flush();
            Double C,I,N;

    System.out.print("\n\nIngrese el Capital c :\t");
    C=Double.valueOf(entrada.readLine());
    c=C.doubleValue();
    System.out.println();

    System.out.print("Ingrese el Interes i:\t ");
    I=Double.valueOf(entrada.readLine());
    i=I.doubleValue();
    System.out.println();

    System.out.print("Ingrese el periono n:\t ");
    N=Double.valueOf(entrada.readLine());
    n=N.doubleValue();
    System.out.println();

    m=c*(Math.pow((1+i),n));
    System.out.print("El Monto a devolver es :\t"+m);
    System.out.println("\n\n\n");
}}
```

9.- Escribir un programa para la conversión de grados sexagesimales a radianes y Centecimales.

```
import java.io.*;
class CConversion {
public static void main(String [] args) throws IOException
{
 double s=0,c=0;
 double r=0;

 BufferedReader entrada =new BufferedReader(
                          new InputStreamReader(System.in));

System.out.println("Ingrese los ángulos en grados Sexagesimales");
Double S;
System.out.flush();
  S=Double.valueOf(entrada.readLine());
  s=S.doubleValue();
  c=(10/9)*s;

  r=s*Math.PI;
  System.out.println("El valor en Grados Centecimaes = "+c);
       System.out.println("El valor en Radianes           = "+r); }
     }
```

10.- Escribir un programa que calcule el área del rombo.

```
class CRombo {
 public static void main(String [] args){
 double d1,d2,Area;
d1=13.56;
d2=19;
Area=(d1*d2)/2;
System.out.println("El Area del Rombo para d1="+d1+" y d2 = "+d2);
System.out.println("\tArea = "+Area);}
}
```

11.- En todo triángulo se cumple que cada lado es proporcional al seno del ángulo opuesto. Esta ley se llama "**ley de senos**".

```
class CladosTriangulo {
     public static void main(String [] args ){
      double c=7 ,alfa=80, beta=50, gama=50;
      double a,b;
       a=c*Math.sin(alfa)/Math.sin(gama);
       b=c*Math.sin(beta)/Math.sin(gama);

       System.out.println("El lado a ="+a);
       System.out.println("el lado b ="+b);
      }
         }
```

12.- Solución que eleva un número al cuadrado y al cubo y lo presenta en tres columnas.

```
import java.io.*;
class CPotencia {
public static void main(String[] args ) throws IOException{
      double a,cuadrado,cubo;
      BufferedReader entrada =new BufferedReader(
                        new InputStreamReader(System.in));
 Double A;
 System.out.print("Ingrese un Numero :\t");
 System.out.flush();
 A=Double.valueOf(entrada.readLine());
 a=A.doubleValue();
 cuadrado=Math.pow(a,2);
 cubo=Math.pow(a,3)       ;
 System.out.println("El valor de "+a+ "  Elevado al Cuadrado y al Cubo es :");
 System.out.print("\n\n\n");
 System.out.println("Base\t"+"" +"Cuadrado\t"+"" +"Cubo\t");
 System.out.println(""+a+"\t"+cuadrado+"\t\t"+cubo);
     }

}
```

13.- Escribir una solución que permita realiza la conversión de grados sexagesimales a Radianes y Centecimales.

```
import java.io.*;
class CConversion {
public static void main(String [] args) throws IOException
{
 double s,c;
 double r;

 BufferedReader entrada =new BufferedReader(
                                          new InputStreamReader(System.in));

  System.out.println("Ingrese los ángulos en grados Sexagesimales");
  Double S;
  System.out.flush();
  S=Double.valueOf(entrada.readLine());
  s=S.doubleValue();
  c=(10/9)*s;
  r=s*Math.PI;
  System.out.println("El valor en Grados Centecimaes = "+c);
  System.out.println("El valor en Radianes           = "+r); }
}
```

14.- Se tiene una circunferencia de radio r, inscrita en un triángulo de lados a,b,c. Encuentre el área de este triángulo en función de *a,b,c y r*.

```
import java.io.*;
class CArea {
public static void main (String [] args)throws IOException
 {

 float a,b,c,r;
 float area=(float)0.0;

 BufferedReader entrada = new BufferedReader(
                                          new InputStreamReader(System.in));

 System.out.flush();
 System.out.println("\t\t\nIngrese los lados del Triangulo");
 System.out.println("\t\t\nLado A= ");
 a=(Float.valueOf(entrada.readLine())).floatValue();
 System.out.println("\t\t\nLado =B ");
 b=(Float.valueOf(entrada.readLine())).floatValue();
 System.out.println("\t\t\nLado =C ");
 c=(Float.valueOf(entrada.readLine())).floatValue();
 System.out.println("\t\t\n Ingrese el radio del Circulo   ");
 r=(Float.valueOf(entrada.readLine())).floatValue();
  area=((a+b+c)/2)*r;

System.out.println("El area del Triandulo es :"+area);  }
}
```

15. determinar el valor de x*log(x)

```
import java.io.*;

class Clogaritmo {
public static void main(String [] args) throws IOException
        {
    double f,x;
    BufferedReader entrada =new BufferedReader(
                        new InputStreamReader(System.in));
        Double d;
        System.out.println("\n Valor de x: ");
        System.out.flush();
        d=Double.valueOf(entrada.readLine());
        x=d.doubleValue();
        /*log(double a) logaritmo neperiano (natural) de a*/
        f=x*Math.log(x);
        System.out.println("f("+x+")="+f);

        }
    }
```

EJERCICIOS PROPUESTOS

1.- Escribir un programa que lea el radio de un circulo y a continuación visualice: área del círculo y a continuación visualice: área del círculo y la longitud de la circunferencia del círculo

2.- Escribir un programa de desglose cierta cantidad de segundos introducida por teclado en su equivalente en semanas, días, horas, minutos y segundos.

3.- Escribir un programa que exprese cierta cantidad de soles en billetes y monedas de curso legal.

4.- Escribir un programa para convertir una medida dada en pies a sus equivalentes en:

a) Yardas
b) Pulgadas
c) Centímetros
d) Metros

(1 pie =12 pulgadas, 1 yarda = 3 pies, 1 pulgada = 2.54cm, 1m= 100cm). Leer el número de pies e imprimir el número de yardas, pies, pulgadas, centímetros y metros.

CAPITULO 2

CONTROL DE EJECUCIÓN

OBJETIVOS:
Al finalizar este capítulo, el alumno aprenderá a:

- **Manejara las condiciones lógicas en el planteo de soluciones en un lenguaje de programación.**
- **Plantear soluciones manejando la estructura condicional if-else**
- **Usar la sentencia de selección múltiple switch**
- **Manejara los conceptos de estructuras repetitivas en un lenguaje de programación while, for, do while.**
- **Utilizará estructuras repetitivas para la solución problemas que se le planteen.**

Java utiliza todas las instrucciones de control de ejecución de C, así es que se verá familiar si se ha programado con C o C++. Muchos lenguajes de programación procesales tienen algún tipo de instrucciones de control, y a menudo de pasan entre lenguajes. En Java, las palabras clave incluyen **ifelse**, **while**, **do-while**, **for**, y una instrucción de control llamada **switch**. Java no soporta **goto** (que todavía puede ser la forma mas conveniente forma de solucionar cierto tipo de problemas).

true y false

todas las instrucciones condicionales usan la *verdad* o *falsedad* de una expresión condicional para determinar el camino de ejecución. Un ejemplo de expresión condicional es **A == B**. Estas utilizan el operador condicional == para ver si el valor de **A** es equivalente a el valor de **B**. La expresión retorna **true** o **false**, Cualquiera de los operadores relacionales que se han visto antes en este capítulo pueden ser utilizados para producir expresiones condicionales. Debe notarse que Java no permite el uso de un número como tipo **boolean**, aun cuando esto esta permitido en C y en C++ (donde verdadero en un valor distinto de cero y falso es cero). Si se quiere utilizar un tipo que no sea **boolean** en una prueba del tipo **boolean**, como **if(a)**, debemos convertirlo a el valor **boolean** utilizando una expresión condicional, como es **if(a ¡= 0)** .

if-else

La instrucción **if-else** es probablemente la forma mas básica de controlar el flujo de un programa. El **else** es opcional, así es que se puede utilizar **if** de dos formas:

Sintaxis :

- **Primera Forma**

```
if(exprecondición)
   instrucciones1
```

- **Segunda Forma**

```
if(condición)
   instrucciones1
else
   instrucciones2
```

La sentencia **if** permite a un programa tomar una decisión para ejecutar una acción u otra, basándose en el resultado verdadero o falso de una expresión. La sintaxis para utilizar esta sentencia es la siguiente:

Una sentencia **if** se ejecuta de la forma siguiente:

1. Se evalúa la condición.
2. Si el resultado de la evaluación de la condición es verdadero (**true**) se ejecutara lo indicado por las *instrucciones1*.
3. Si el resultado de la evaluación de la condición es falso (**fase**), se ejecutara lo indicado por la *instrucciones2*, si la cláusula **else** se ha especificado.
4. Si el resultado de la evaluación de la condición es falso, y la cláusula **else** se ha omitido, la *instrucciones1* se ignora.
5. En cualquier caso, la ejecución continua en la siguiente sentencia ejecutable que haya continuación de la sentencia **if..**

La condición debe producir un resultado del tipo **boolean**. Las *instrucciones* terminan con un punto y coma o una instrucción compuesta, que es un grupo de instrucciones encerrado entre llaves. En cualquier momento que la palabra "*instrucción*" sea utilizada, siempre implica que la instrucción pueda ser simple o compuesta.

A continuación se exponen algunos ejemplos para que vea de una forma sencilla como se utiliza la sentencia **if**

```
if(x!= 0)
  b= a/x;
  b= b+1;
```

- En este ejemplo, la condición viene impuesta por la expresión *x!=0.* Entonces *b= a/x*, que sustituye a la sentencia 1 del formato general, se ejecuta si la expresión es verdadera (x distinta de 0) y no se ejecutara si la expresión es falsa (x igual a 0). En cualquier caso, se continua la ejecución en la línea siguiente, *b=b+1*. Veamos otro ejemplo:

- En este otro ejemplo, la condición viene impuesta por una expresión *a<b.* Si al evaluar la condición se cumple que *a* es menor que *b*, entonces se ejecuta la sentencia *c=c+1.* En otro caso, esto es, si *a* es mayor o igual que *b*, se continua en la línea siguiente, ignorándose la sentencia *c=c+1.*

```
if( a < b )c = c+1;
// Siguiente línea del programa
```

- En el ejemplo siguiente, la condición viene impuesta por la expresión *a! = 0 && b!= 0.*

  Si al evaluar la condición se cumple que a y b son distintas de cero, entonces se ejecuta la sentencia *x=i.* En otro caso, la sentencia *x=i* se ignora, continuando la ejecución en la línea siguiente.

```
if( a != 0 && b !=0)
  x = i;
//siguiente línea del programa
```

- En el ejemplo siguiente, si se cumple que *a* es igual a *b*5* , se ejecutan las sentencias *x=4 y a=a+x.* en otro caso, se ejecuta la sentencia *b= 0* . en ambos casos, la ejecución continúa en la siguiente línea de programa.

```
if(a== b*5)
{
   x=4;
  a= a+x;
```

```
}
else
  b=0;
// Siguiente línea del programa
```

- Un error típico es escribir. En lugar de la condición del ejemplo anterior, la siguiente:

```
if(a = b*5)
//…
```

- En este caso, suponiendo por ejemplo que *a* es de tipo **int**, el compilador mostrara un mensaje de error indicado que no puede convertir un **int** a **boolean**, porque la sentencia anterior es equivalente a escribir:

```
a = b*5;
if(a)
//…
```

Donde se observa que *a* no puede dar un resultado **boolean**. Si seria correcto lo siguiente:

```
a = b*5;
if (a != 0 )
//…
```

Que equivale a:

```
if (( a = b*5) != 0 )
//…
```

- En este otro ejemplo que se muestra a continuación, la sentencia **return** se ejecutara solamente cuando *car* sea igual al carácter *'s'*.

```
If( car = = 's')
   return;
```

ANIDAMIENTO DE SENTENCIAS if

Como ejemplo se puede observar el siguiente segmento de programa que escribe un mensaje indicando como es un número *a* con respecto a otro *b* (mayor, menor o igual):

```
if(a > b)
  flujoS.println(a+ "ves mayor que"+ b);
else if (a < b)
```

```
  flujoS.println(a+ “es menor que”+b);
else
  flujoS.println(a+ “es igualа”+b);
// siguiente línea del programa
```

Es importante observar que una vez que se ejecuta una acción como resultado de haber evaluado las condiciones impuestas, la ejecución del programa continua en la siguiente linea a la estructura a que dan lugar las sentencias **if…else** anidadas. En el ejemplo anterior si se cumple que *a* es mayor que *b*, se escribe el mensaje correspondiente y se continua en la siguiente línea de programa.

Así mismo, si en el ejemplo siguiente ocurre que *a* no es igual a *0*, la ejecución continua en la siguiente línea del programa.

```
  if( a = = 0)
   if (b ¡= 0)
    s = s + b;
else
    s = s + a;
// siguiente línea del programa
```

Si en lugar de la solución anterior, lo que deseamos es que se ejecute s = s + a cuando a no es igual a 0, entonces tendremos que incluir entre llaves el segundo if sin la cláusula else; esto es:

```
if(a = = 0)
{
If(b ¡= 0)
   s = s + b;
}
else
  s = s + a;
// siguiente línea del programa
```

Como ejercicio sobre la teoría expuesta, vamos a realizar una aplicación que de como resultado el menor de tres números *a, b* y *c*. La forma de proceder es comparar cada número con los otros dos una sola vez. La simple lectura del código que se muestra a continuación es suficiente para entender el proceso seguido.

```
// La clase Leer debe estar en alguna carpeta de las especificadas
// Por la variable de entorno CLASSPATH.
//
public class CMENOR
{
   // Menor de tres números a, b y c
```

```
   public static void main(String[] args)
      {
         float a, b, c, menor;

//Leer los valoresde a, b y c
System.out.print ("a : "); a = Leer.datofloat( );
System.out.print ("b : "); b = Leer.datofloat( );
System.out.print ("c : "); c = Leer.datofloat( );
// Obtener el menor
if ( a < b)
   if(a < c)
     menor = a ;
  else
    menor = c ;
else
    if(b < c)
       menor = b ;
else
          menor = c ;
          System.out.println("menor = "+ menor) ;
    }
}
```

ESTRUCTURA else if

La estructura presentada a continuación, aparece con bastante frecuencia y es por lo que se le da un tratamiento por separado. Esta estructura es consecuencia de las sentencias **if** anidadas. Su formato general es:

```
if (condición 1)
    Sentencia 1;
else if (condición 2)
    Sentencia 2;
else if (condición 3)
    Sentencia 3;
.
.
.
else
    Sentencia n;
```

La evaluación de esta estructura sucede así: si se cumple la *condición 1*, se ejecuta la *sentencia 1* y si no se cumple se examinan secuencialmente las condiciones siguientes hasta el último **else**, ejecutándose la sentencia correspondiente al primer **else if**, cuya *condición* sea cierta. Si todas las condiciones son falsas, se ejecutan la sentencia *n* correspondiente al último **else**. En cualquier caso, se continúa en la primera sentencia ejecutable que haya a continuación de la estructura.

Las *sentencias 1, 2, …, n* pueden ser sentencias simples o compuestas.

Por ejemplo al efectuar una compra en un cierto almacén, si adquirimos mas de 100 unidades de un mismo articulo, nos hacen un descuento de un 40 %; entre 25 y 100 un 20%; entre 10 y 24 un 10%; y no hay descuento para una adquisición de menos de 10 unidades. Se pide calcular el importe a pagar. La solución se presentara de la siguiente forma:

```
Código articulo..... 111
Cantidad comprada... 100
Precio unitario..... 100

Descuento..... 20.0%
Total..... 8000.0
```

En la solucion presentada como ejemplo, se puede observar que como la cantidad comprada esta entre 25 y 100, el descuento aplicado es de un 20%.

La solución de este problema puede ser de la forma siguiente:

- Primero definimos las variables que vamos a utilizar en los cálculos.

```
int ar, cc;
float pu, Desc;
```

- A continuación leemos los datos *ar, cc* y *pu*.

```
System.out.print("código articulo……") ;
ar = Leer.datoint() ;
System.out.print("cantidad comprada...") ;
cc = Leer.datoint() ;
System.out.print("precio unitaria...") ;
pu = Leer.datofloat() ;
```

- Como los datos, realizados los calculos y escribimos el resultado.

```
if ( cc > 100)
    Desc = 40F ;        // descuento 40%
else if ( cc >= 25)
    Desc  = 20F ;       // descuento 20%
else if (cc >= 10)
  Desc = 10F ;   // descuento 10%
else
  desc = 0.0F;    // descuento 0%
system.out.println("descuento....."+Desc+"%") ;
system.out.println("total....."+cc * pu * (1 – desc / 100));
```

se puede observar que las condiciones se han establecido según los descuentos de mayor a menor. *Como ejercicio, piense p pruebe que ocurriría si establece las condiciones según los descuentos de menor a mayor. La aplicación completa se muestra a continuación.*

```
// la clase leer debe estar en alguna carpeta de las especificadas
```

```
// por la variable de entorno CLASSPATH.
//
public class CDescuento
{
    public static void main(String[] args)
  {
     int ar, cc;
     float pu, desc;

     System.out.print("codigo articulo..... ") ;
     ar = Leer.datoInt( ) ;
     System.out.print("cantidad comprada.....") ;
     cc = Leer.datoInt( ) ;
     System.out.print("precio unitario.....") ;
     pu = Leer.datoFloat( ) ;
     System.out.println( ) ;

     if( cc > 100)
       Desc = 40F;          // descuento 40%
     else if (cc >= 25)
       Desc = 20F;          // descuento 20%
     else if (cc >= 10)
       Desc = 10F;          // descuento 10%
     Else
            Desc = 0.0F;              // descuento 0%
        System.out.println("descuento...."+ desc + "%");
        System.out.println("total........"+Cc * pu *(1- desc/ 100));

    }
}
```

SENTENCIA SWITCH

La sentencia switch permite ejecutar una de varias acciones, en función del valor de una expresión. Es una sentencia especial para decisiones múltiples.

La sintaxis a utilizar esta dada en la siguiente expresión sentencia es:

```
switch (expresión)
{
     Case expresión – constante 1:
           [sentencia 1;]
   [case expresión-constante 2:]
           [sentencia 2;]
   [case expresión- constante 3:]
           [sentencia 3;]
.
.
.
[default:]
           [sentencia n;]
}
```

donde ***expresión*** es una entera de tipo ***char, byte, short o int*** y ***expresión-constante*** es una constante también entera y de los mismos tipos. Tanto la expresión como las expresiones constantes son convertidas implícitamente a int. Por ultimo, sentencia es una sentencia simple o compuesta. En el caso de tratarse de una sentencia compuesta, no hace falta incluir las sentencias simples entre {}.

La sentencia **switch** evalúa la expresión entre paréntesis y compara su valor con las constantes de cada **case**. La ejecución de las sentencias del bloque de la sentencia **switch**, comienza en el **case** cuya constante coincida con el valor de la expresión y continua hasta el final del bloque o hasta una sentencia que transfiera el control fuera del bloque de **switch**; por ejemplo, **break**. La sentencia **switch** puede incluir cualquier numero de cláusulas **case**.

Si no existe una constante igual al valor de la expresión, entonces se ejecutan las sentencias que están a continuación de **default**, si esta cláusula ha sido especificada. La cláusula **default** puede colocarse en cualquier parte del bloque y no necesariamente al final.

En una sentencia **switch** es posible hacer declaraciones en el bloque de cada **case**, igual que en cualquier otro bloque, pero no al principio del bloque **switch**, antes del primer **case**. Por ejemplo:

```
switch (m)
 {
      int n = 0, k = 2 // declaración no permitida
      case 7 :
           Int i = 0 ;    // declaración permitida
           while (i < m)
            {
                n += (k + i)*3;
                i++;
            }
            break;
         case 13:
             //…
             break;
             //…
}
```

El error se ha presentado en el ejemplo anterior puede solucionarse así:

```
int n= 0, k=2;
switch (m)
{
     //…
}
```

Para ilustrar la sentencia **switch**, vamos a realizar un programa que lea una fecha representada por dos enteros, *mes* y *año*. Y de cómo resultado los días correspondientes al *mes*. Esto es:

Introducir mes (##) y año (####): 5 2002
El mes 5 del año 2002 tiene 31 días

Hay que tener en cuenta que febrero puede tener 28 días, o bien 29 si el año es bisiesto. Un año es bisiesto cuando es múltiplo de 4 y no de 100 o cuando es múltiplo de 400. por ejemplo, el año 2000 por las dos primeras condiciones no seria bisiesto, pero si lo es porque es múltiplo de 400; el año 2100 no es bisiesto porque aunque sea múltiplo de 4, también lo es de 100 y no es múltiplo de 400.

La solución de este problema puede ser de la siguiente forma:

- Primero definimos las variables que vamos a utilizar en los cálculos.

  int dias = 0, mes = 0, año = 0;

- A continuación leemos los datos *mes* y *año*.

  System.out.print("mes (##) ; ") ; mes = Leer .datoInt( ) ;
  System.out.print("año   (####) : ") ; año = Leer .datoInt( );

- Después comparamos al mes con las constantes 1, 2, ….., 12. Si *mes* es 1, 3, 5, 7, 8, 10 o 12 asignamos a *días* el valor 31. Si *mes* es 4, 6, 9, u 11 asignamos q *días* el valor 30. Si *mes* es 2, verificamos si el *año* es bisiesto, en cuyo caso asignamos a *días* el valor 29 y si no es bisiesto, asignamos a *días* el valor 28.
  Si *mes* no es ningún valor de los anteriores enviaremos un mensaje al usuario indicándole que el mes no es valido. Todo este proceso lo realizaremos con una sentencia **switch.**

```
switch  (mes)
{
     Case 1: case 3: case 5: case 7: case 8: case 10: case12:
        Dias = 31;
        Break;
     case 4: case6: case 9: case 11:
        Días = 30;
        break;
      case 2
        // ¿ es el año bisiesto?
      if((año %4==0)&&(año%100¡= o)||(año%400== 0))
           Dias = 29;
        Else
           Dias = 28;
           break;
      default:
        System.out.println("\nElmes no es valido");
        break;
}
```

Cuando una constante coincidan con el valor de *mes*, se ejecutan las sentencias especificadas a continuación de la misma, siguiendo la ejecución del programa por los bloques de las siguientes cláusulas **case**, a no ser que se tome una acción explicita para abandonar el bloque de la sentencia **switch**. Esta es precisamente la función de la sentencia **break** al final de cada bloque **case**.

- Por ultimo si el mes es valido, escribimos el resultado solicitado.

```
If  (mes >= 1 && mes <= 12)
 System.out.printhn("\ nEl mes "+ mes + " del año " + año +"tiene"+ dias + "dias");
```

El programa completo se muestra a continuación:

```
// la clase leer debe estar en alguna carpeta de las especificadas
// por la variable de entorno CLASSPATH.
//
public class CDiasMes
{
    // Dias correspondientes a un mes de un ano dado
    public static void main(String [] args)
    {
       int dias = 0, mes = 0, año = 0;

                System.out.print("mes (##); mes = leer.datoint( );
                System.out.print("año (####): "); año = leer.datoint( );

      switch (mes)
      {
         case 1:          // enero
         case 3:          //marzo
         case 5:           //mayo
         case 7:          //Julio
         case 8:          //agosto
         case 10:         //octubre
         case 12:         //diciembre
             días = 31
             break;
         case 4:          //abril
         case 6:          // junio
         case 9:          //septiembre
         case11:          //noviembre
            días = 30;
            break;
        case 2:           //febrero
            // ¿ es el año bisiesto?
          if((año%4==0)&&(año%100¡=0)||(año%400== 0))
              dias = 29;
            Else
              dias = 28;
              break;
         default:
           System.out.println("\nElmesnoes valido");
           break;
}
        If (mes >= 1 && mes <=12)
        System.out.println("\nEl mes +"+ mes "del año" + año + "tiene "+ días + "días");
  }
}
```

El que las cláusulas **case** estén una a continuación de otra o una debajo de otra no es mas que una cuestión de estilo, ya que java interpreta cada carácter nueva línea como un espacio en blanco; esto es, el código al que llega el compilador es el mismo en cualquier caso.

La sentencia **break** que se ha puesto a continuación de la cláusula **default** no es necesaria; simplemente obedece a un buen estilo de programación. Así, cuando tengamos que añadir otro caso ya tenemos puesto **break**, con lo que hemos eliminado una posible fuente de errores.

return

La palabra clave return tiene dos propósitos: especifica que valor de retorno tendrá un método (si no tiene un tipo **void** como retorno) y produce un retorno inmediato de ese valor. El método **test()** anterior puede ser reescrito para tomar ventaja de esto.

```
public class IfElse {
static int test(int testval, int t) {
  int result = 0;
  if(testval > t)
  return +1;
  else if(testval < t)
  return -1;
  else
  return 0; // Match
}
public static void main(String[] args) {
System.out.println(test(10, 5));
System.out.println(test(5, 10));
System.out.println(test(5, 5));
}
} ///:~
```

Aquí no hay necesidad de la palabra **else** ya que el método no continúa luego de ejecutar un **return.**

Estructuras de Iteración.

Los bucles de control **while**, **do-while** y **for** son clasificados como *instrucciones de interacción*. Una *instrucción* se repite hasta que la expresión de control booleana evaluada es *falsa*. La forma del bucle **while** es

```
while(expresión booleana){
        instrucción1.
        Instrucción2.
        Instrucción3.
.
.
.
        Instrucción n


}
```

La *expresión booleana* es evaluada una vez al inicio del bucle y nuevamente cada nueva iteración de la *instrucción*. He aquí un simple programa ejemplo que genera números al azar hasta que una condición particular es encontrada:

```
public class WhileTest {
   public static void main(String[] args) {
       double r = 0;
         while(r < 0.99d) {
                 r = (Math.random());
                 System.out.println(r);
         }
   }
} ///:~
```

Utiliza un método estático llamado **random()** en la librería **Math**, que genera un valor **double** entre 0 y 1 (Esto incluye 0 pero no 1). La expresión condicional para el **while** dice “manténgase haciendo este bucle hasta que el numero sea mayor a 0.99 o mayor”. Cada vez que se ejecute este programa obtendremos una lista de números de diferente tamaño dado que se generan números de manera aleatoria.

El siguiente ejemplo, que visualiza el codigo ASCII de cada uno de los caracteres de una cadena de texto introducida por el teclado, da ligar a un bucle infinito, porque la condicion es siempre cierta(valor trae). Para salir del bucle infinito tiene que pulsar las teclas *ctrl. + C.*

```
import java.io.*;

public class CAscii
{
    // codigo ASII de cada uno de los caracteres de un texto
public static void main(String[] args )
    {
        char car = 0;  //car = character nulo (\0)

       try
       {
            System.out.print("Ingrese una cadena de texto: ") ;
            while (true)// condicion siempre cierta
            {
                car = (char)System.in.read( ); // leer el siguiente carácter
                if ( car ¡= ' \r' && car ¡= '\n')
                 System.out.println("el codigo ASCII de "+ car + " es " + (int)car);
                // si no hay datos disponibles, solicitarlos
                if (System.in.available( ) = = 0 )
                        System.out.print("Introduzca una cadena de texto: ");
        }
     }
     catch(I0Exception ignorada) {}
     }
}
```

A continuación ejecutamos la aplicación. Introducimos, por ejemplo, el carácter 'a' y observamos los siguientes resultados:

```
Introduzca una cadena de texto: a [entrar]
El código ASCII de a es 97
Introduzca una cadena de texto:
```

Este resultado demuestra que cuando escribimos 'a' y pulsamos la tecla entrar para validar la entrada, solo se visualiza el código ASCII de ese carácter; los caracteres \r y \n introducidos al pulsar *Entrar* son ignorados porque así se ha programado. Cuando se han leído todos los caracteres del flujo de entrada, se solicitan nuevos datos. Lógicamente, habrá comprendido que aunque se lea carácter a carácter se pueda escribir, hasta pulsar *Entrar*, un texto cualquiera. Por ejemplo:

```
Introduzca una cadena de texto: hola[entrar]
El código ASCII de h es 104
El código ASCII de o es 111
El código ASCII de l es 108
El código ASCII de a es 97
Introduzca una cadena de texto:
```

El resultado obtenido permite observar que el bucle **while** se esta ejecutando sin pausa mientras hay caracteres en el flujo de entrada. Cuando dicho flujo queda vacío y se ejecuta el método **read** de nuevo, la ejecución se detiene a la espera de nuevos datos.

Modifiquemos ahora el ejemplo anterior con el objetivo de eliminar el bucle infinito. Esto se puede hacer incluyendo en el **while** una condición de terminación; por ejemplo, leer datos hasta alcanzar la marca de fin de fichero. Recuerde que para el flujo estándar de entrada, esta marca se produce cuando se pulsan las teclas *ctrl.+D* en UNIX, o bien *ctrl.+Z* en aplicaciones Windows de consola, y que cuando **read** lee una marca de fin de fichero, devuelve el valor -1.

```
import java.io.*
public class CAscii
{
  // Codigo ASCII de cada uno de los caracteres de un texto
    public static void main (String[] args)
     {
        final char eof = (char) -1;
        char car = 0;  // car character nulo (\0)
         try
         {
            System.out.println("introduzca una cadena de texto.");
            System.out.println("para terminar pulse ctrl.+z\n");
            while (( car = (char)System.in.read()) != eof)
             {
                If (car != '\r' && car != '\n')
                    System.out.println("el código ASCII de "+ car +  "es" + (int) car);}
          }
        }
    }
```

Una solución posible de esta aplicación es la siguiente:

```
Introduzca una cadena de texto.
Para terminar pulse ctrl. + z
Hola [entrar]
El código ASCII de h es 104
El código ASCII de o es 111
El código ASCII de l es 108
El código ASCII de a es 97
Adiós [entrar]
El codigo ASCII de a es 97
El codigo ASCII de d es 100
El codigo ASCII de i es 105
El codigo ASCII de o es 162
El codigo ASCII de  s es 115
[Ctrl.] [z]
```

do-while

la forma del **do-while** es:

```
do {
        instrucción1.
        Instrucción2.
        Instrucción3.
.
.
.
        Instrucción n


}
while(expresion booleana)
```

la única diferencia entre **while** y **do-while** es que la instrucción en **do-while** se ejecuta siempre por lo menos una vez, aún si la expresión se evalúa como falsa la primera vez. En un **while**, si la condición es falsa la primera vez la instrucción nunca se ejecuta. En la práctica, **do-while** es menos común que while.

for

Un bucle **for** realiza un inicialización antes de la primera iteración. Luego realiza pruebas condicionales y, al final de cada iteración, alguna forma de "adelantamiento de a pasos". La forma del bucle **for** es:

```
for (inicialización; expresión booleana; paso)
instrucción
```

Cualquiera de las expresiones *inicialización*, *expresión booleana* o n pueden ser vacías. La expresión es probada luego de cara iteración, y tan pronto como se evalúa en **false** la ejecución continúa en la línea seguida por la instrucción **for**. En el final de cada bucle, paso se ejecuta. Los bucles **for** usualmente son utilizados para tareas de "cuenta".

```
// Demuestra los bucles "for" listando
// todos los caracteres ASCII.
public class ListCharacters {
public static void main(String[] args) {
for( char c = 0; c < 128; c++)
        if (c != 26 ) // Borrado de pantalla ANSI
           System.out.println("value: " + (int)c +" character: " + c);
  }
}
```

Se puede ver que la variable **c** esta definida en el punto donde va a ser utilizada, dentro de la expresión de control del bucle **for**, antes que al comienzo de el bloque indicado por la llave de apertura. El alcance de c es la expresión controlada por el **for**.

Los lenguajes tradicionales como C requiere que todas las variables sean definidas en el comienzo del bloque así es que cuando el compilador crea un bloque puede asignar el espacio para esas variables. En Java y en C++ se pueden declarar las variables por todo el bloque, de código definiéndolas en el punto en que se necesiten. Esto permite un estilo de código mas natural y hace el código mas fácil de entender.
Se pueden definir múltiples variables con la instrucción **for**, pero estas deben ser del mismo tipo:

```
for(int i = 0, j = 1;i < 10 && j != 11;i++, j++)
/* cuerpo del bucle */;
```

la definición del tipo **int** en la instrucción **for** cubre **i** y **j**. La habilidad para definir variables en la expresión de control está limitada por el bucle **for**. No se puede utilizar este método con ninguna otra instrucción de selección o iteración.

Otros casos del operador for:

En el siguiente ejemplo se puede observar la utilización de la coma como separador de las variables de control y de las expresiones que hacen evolucionan los valores que intervienen en la condición de finalización.

```
int f, c;
for (f=3,c=6;f+c<40; f++, c += 2)
    System.out.println("f="+f+"\tc = " + c);
```

Este otro ejemplo que va continuación, imprime los valores desde 1 hasta 10 con incrementos de 0.5.

```
for (float i = 1; i <= 10; i += 0.5)
     System.out.print(i + "");
```

El siguiente ejemplo imprime las letras del abecedario en orden inverso.

```
char car;
for ( car = 'z'; car >= 'a'; car - - )
     System.out.print(car + "");
```

El ejemplo siguiente indica como realizar un bucle infinito. Para salir de un bucle infinito tiene que pulsar las teclas *ctrl.+ c*.

El operador coma

Hasta hora el *operador* coma (no el *separador* coma, que es utilizado para separar definiciones y argumentos de funciones) le hemos dado un solo un uso en Java:

En la expresión de control de un bucle **for**. En la inicialización y en la parte de pasos de la expresión de control se pueden tener varias instrucciones separadas por comas, y aquellas instrucciones serán evaluadas de forma secuencial. El trozo de código anterior utiliza esta definición.

```
public class CommaOperator {
public static void main(String[] args) {
   for(int i = 1, j = i + 10; i < 5;i++, j = i * 2) {
         System.out.println("i= " + i + " j= " + j);
   }
 }
}
```

Y su salida:

```
i= 1 j= 11
i= 2 j= 4
i= 3 j= 6
i= 4 j= 8
```

Como se puede ver en ambos en la inicialización y en las parte de pasos las instrucciones son evaluadas en orden secuencial. Además, la parte de la inicialización puede tener cualquier número de definiciones *de un tipo*.

break y continue

Dentro del cuerpo de cualquiera de las instrucciones de la iteración se puede tener control del flujo del bucle utilizando **break** y **continue**. **break** se sale del bucle sin ejecutar el resto de las instrucciones en el bucle. **continue** detiene la ejecución de la actual iteración y regresa al comienzo del bucle para comenzar en la siguiente iteración. Este programa muestra ejemplos de **break** y **continue** con bucles **for** y **while**:

```
// Demonstrates break and continue keywords.
public class BreakAndContinue {
        public static void main(String[] args) {
                for(int i = 0; i < 100; i++) {
                if(i == 74) break; // Fuera del bucle
                if(i % 9 != 0) continue; // Siguiente iteración
                System.out.println(i);
                }
                int i = 0;
                // Un “bucle infinito":
                while(true) {
                i++;
                int j = i * 27;
                if(j == 1269) break; // fuera del bucle
                if(i % 10 != 0) continue; // Arriba en el bucle
```

```
                System.out.println(i);
                }
        }
} ///:~
```

Explicación .-

En el bucle **for** el valor de **i** nunca llega a 100 porque la instrucción **break** sale del bucle cuando **i** es **74**. Normalmente, se usará una línea **break** como esta solo si no se conoces cuando va a suceder la condición de terminación. La instrucción ***continue*** produce que la ejecución regrese a el comienzo de la iteración del bucle (de esta manera se incrementa **i**) cuando quiera que **i** no sea divisible entre 9. Cuando esto se sucede el valor es impreso.

La segunda parte muestra un “bucle infinito” que puede, en teoría, continuar para siempre. Sin embargo, dentro del bucle hay una instrucción **break** que saldrá del bucle. Además, se puede ver que **continue** regresa a el inicio del bucle sin completar el resto (Estas impresión se sucede en el segundo bucle solo cuando el valor de i es divisible entre 10).

La salida es:

```
0
9
18
27
36
45
54
63
72
10
20
30
40
```

El valor de 0 es impreso porque 0 % 9 produce 0.
Una segunda forma de bucle infinito es **for(;;)**. El compilador trata **while(true)** y **for(;;)** de la misma forma así es que cualquiera que se use es por preferencias de programación.

EJERCICO RESUELTOS

1. Escribir una solución en java que permita leer un número entero n mayor de cero y que imprima los n términos de la serie **10,15,23,35,53,80,. . . …. ..** Además debe imprimir la suma de los n términos.

```
public class CSerie {
    public static void main(String args []){
            int n;
            int te=10;
            int k1 = 5, k2=3;
            int con = 0;
            double sum=0;
            System.out.println("Programa que calcula los terminos de una serie");
            do{
                    System.out.print("Ingrese un Nº Positivo Mayor de Cero");
                    n=Leer.datoInt();
                    System.out.println();
            }
            while(n<=0);
            do{
                    System.out.print("  ,"+te);
                    sum +=te;
                    con++;
                    if(con<n){
                            te +=k1;
                            k1 +=k2;
                            k2 +=con;
                    }
            }
            while(con<n);
            System.out.println("\n\n La suma de los Términos es "+sum);
    }
}
```

2. Escribir un programa que permita leer un número entero n mayor de cero y que imprima los n términos de la serie: **2, 1, 1,2, 8, 64, … …** Además, debe imprimir la suma de n términos.

```
public class CTestSerie {
    public static void main (String args[]){
            int n;
            int ter = 2;
            double  raz = 0.5;
            int con = 0;
            double  sum= 0;
            System.out.println("programa que calcula los terminos de la serie");
            do{
                    n= Leer.datoInt();
            }
```

```
                while(n<0);
                do{
                        System.out.print(" ,"+ter);
                        sum +=ter;
                        con++;
                        if(con<n){
                                ter *=raz;
                                raz *=2;
                        }
                }
                while(con<n);
                System.out.println();
                System.out.println("La suma de los Términos es : "+sum);
        }
    }
```

3. Escribir un programa que calcule el factorial de un número entero mayor de cero. El programa debe repetirse mientras se desee.

```
import java.io.*;
class CFactorial {
        public static void main (String[] args)throws IOException {
        long fact=1;
        long n=0;
        char resp;
        do{

                        do{
                         System.out.print("Ingresar un Nº mayor de Cero = ");
                         n=Leer.datoLong();
                         System.out.println();
                        }
                        while(n<0);
                        if (n==0||n==1){
                                fact=1;
                        }
                        else{
                                for(long ter=1;ter<=n;ter++){
                                        fact=fact*ter;
                                }
                        }
                        System.out.println("El Factorial de N ="+n+" Es igual a : "+fact);
                        System.out.print("Desea Continuar [S|N] =");
                        resp=(char)System.in.read();
                        n=0;
                        fact=1;
  }
  while(resp=='S');
}
```

4. Escribir un programa que lea 3 números enteros positivos A,B,C y calcule la suma de los n términos de la serie :

$$\frac{1}{A}-\frac{2}{1+B}+\frac{4}{1+3B}-\frac{7}{1+5B}+\frac{11}{1+7B}.....$$

```
class CTestSerie04 {
	public static void main(String args[]){
		int a,b,n,cont=0, i=-1, num=1, den,k=1;
		double suma=0;
		//Ingresamos a,b,cy n
		do{
			System.out.println("Ingrese el Valor de A : =  ");
			a=Leer.datoInt();
			System.out.println("Ingrese el Valor de B : =  ");
			b=Leer.datoInt();
			System.out.println("Ingrese el Valor de N : =  ");
			n=Leer.datoInt();

		}
		while (a<=0||b<=0||n<=0);

		suma=1.0/a; cont=1;

		System.out.print(" "+1+"/"+a);
		while(cont<n){
			num=num+cont;
			den=1+k*b;
			k=k+2;
			System.out.print(" "+num+"/"+den);
			suma=suma+(num/den)*i;
			i=-i;
			cont=cont+1;
		}
		System.out.println();
		System.out.print("La Suma es "+suma+" Total de Terminos "+cont);
	}
}
```

5. Se desea ingresar los siguientes datos de n alumnos: nota(0-20), sexo (M,F) y estado civil (S,C,D).

   Se quiere determinar:

   a. Número de hombres y número de mujeres (h,m)
   b. No de hombres aprobados y No de mujeres aprobados (hap, map)
   c. No de hombres casados aprobados y desaprobados (hcap,hcdes)
   d. No de hombres casados (hcas)
   e. No de mujeres solteras aprobadas (msap)
   f. No de mujeres divorciadas (mdiv)
   g. No total de desaprobados

```
import java.io.*;
public class CTestAlumnos {
	public static void main(String args[])throws IOException{
```

```
int hap=0,map=0;        /*Hombres y Mujeres aprobadas.*/
int hcap=0,msap=0;      /* Hombres casados y mujeres Solteras*/
int hcdes=0;            /* Hombres Casados Desaprobados*/
int hcas=0, hs=0;       /* Hombres casados y Hombres solteros*/
int h=0, m=0, n=0;      /* Nº de Hombres y mujeres y Nº Total de datos*/
int des=0, mdiv=0;      /* Total de Desaprobados y mujeres divorciadas*/
int nota, cont=0;
char sexo, est;         /* Estado Civil : Soltero, Casado o Divorciado*/
do{
        System.out.print("Ingrese el Nº de Datos :");
        n=Leer.datoInt();
        System.out.println();
}while(n<=0);

while(cont<n){
        do{
                System.out.print("Ingrese Nota : ");
                nota=Leer.datoInt();
                System.out.println();
        }
        while(nota<0 ||nota>20);

        do{
                System.out.print("Ingrese Sexo [M|F] ");
                sexo=(char)System.in.read();
                System.out.print(" "+sexo);
        }
        while(sexo!='F' && sexo!='M'&&sexo!='f'&&sexo!='m');

        do{
                System.out.print("Ingrese Estado Civil [S|C|D] ");
                est=(char)System.in.read();
                System.out.println();
        }
        while(est!='S' && est!='C'&& est!='D');
        cont++;

        if(sexo=='M'){
                h++;
                if(nota>=11){
                   hap++;
                   if(est=='C'){
                          hcap++;
                          hcas++;
                     }
                 }
                 else{ des++;
                     if(est=='C')
                     {hcdes++;
                       hcas++;
                     }

                   }
              }
             else{m++;
                if(nota>=11)
                { map++;
                  if(est=='S') msap++;
                }
                else des++;
                if(est=='D') mdiv++;
             }
}
System.out.println("REPORTE DE RESULTADOS ");
```

```
                System.out.println("Nº DE HOMBRES                                    "+h);
                System.out.println("Nº DE HOMBRES APROBADOS                          "+hap);
                System.out.println("Nº DE HOMBRES CASADOS                           "+hcas);
                System.out.println("Nº DE HOMBRES CASADOS APROBADOS      "+hcap);
                System.out.println("Nº DE HOMBRES CASADOS DESAPROBADOS"+hcap);
                System.out.println("Nº DE MUJERES                                     "+m);
                System.out.println("Nº DE MUJERES APROBADAS                          "+map);
                System.out.println("Nº DE MUJERES DESAPROBADAS                      "+mdiv);
                System.out.println("Nº DE MUJERES SOLTERAS APROBADAS      "+msap);
                System.out.println("TOTAL DE DESAPROBADOS                            "+des);
        }

}
```

6. Escribir un programa que permita clacular el MCM y el MCD de 2 *Numeros enteros positivos

```
class CMcmMcd {
public static void main(String args[]){
  int n1,n2;
  int mcd =1, mcm, div =2;

   do{
            System.out.println("Ingrese dos Numeros Enteros");
            System.out.print("Ingrese dos N1 : = ");
            n1=Leer.datoInt();
            System.out.println();
            System.out.print("Ingrese dos N2 : = ");
            n2=Leer.datoInt();
   }
   while(n1<=0||n2<=0);

   while(div<=n1 && div<n2){
    if (n1%div ==0 && n2%div==0)
    { n1=n1/div;
      n2=n2/div;
      mcd=mcd*div;
    }
    else
            div =div+1;
   }
       mcm=mcd*n1*n2;
   System.out.println("El Máximo Común Divisor "+mcd);
   System.out.println("Minimo Común Múltiplo   "+mcm);
  }
}
```

7. Escribir un programa que diga si un número es primo.

```
public class CNumPrimo {
    public static void main(String args[]){
    int n, i, band;
    System.out.println("Ingrese un Numero");
    n=Leer.datoInt();
    i=n/2;
    band=1;
    while(i>1){
            if(n%i==0) band=0;
            i=i-1;
    }
     if(band==1) System.out.println("Es Primo");
```

```
        else System.out.print("No es Primo");

    }
}
```

8. Encuentre el promedio ponderado de n números.

```
public class Cpromedio {
    public static void main(String args[]){
            int i,n,p;
            float nro, sn=0,sp=0;
            System.out.print("Cantidad de Números");
            n=Leer.datoInt();
            for(i=1;i<=n;i++){
                    System.out.print("Numero : = ");
                    nro=Leer.datoFloat();
                    System.out.println();
                    System.out.print("Peso    : =");
                    p=Leer.datoInt();
                    System.out.println();
                    sn=sn+nro*p;
                    sp=sp+p;
            }
            System.out.print("El promedio Ponderado es : "+sn/sp);
    }
}
```

EJERCICIOS PROPUESTOS

1. Escribir un programa que permita leer números enteros diferentes de cero y que al finalizar imprima.

El máximo y mínimo No positivo	La cantidad de pares e impares positivos.
El máximo y el mínimo No negativo	La cantidad de pares e impares negativos
La cantidad de datos leídos	La cantidad de No, positivos y negativos.

El programa termina cuando se ingresa No cero.

a. Si no se ingreso ningún No. Negativo debe imprimirse un mensaje adecuado.

b. Si no se ingresó ningún No positivo debe imprimir un mensaje adecuado.

2. Escribir un programa que calcule un No y calcule la suma de los n términos de la serie :

X es un No real mayor igual 0.1. La sumatoria termina cuando se tiene un término $x^n/n!$ <0.0001 también se debe indicar cuántos términos se sumaron.

3. Plantear una solución en java para imprimir una tabla de valores de la función.

$$S(x)=\sum_{n=1}^{21}\frac{(-1)^{(n-2)/2}x^n}{n!}$$

Para valores de x de 0 a 20 en incrementos de 0.5 y alineados lado a lado con valores de sen(x) de la siguiente manera:

X S(X) SEN(X).

4.- Encontrar por computadora, el límite de **[1-cos(x)]/sqrt(x)**, cuando x tiende a cero.

5.- Utilizar el producto para obtener el valor de **pi**.

$$\Pi=2\frac{2*2*4*4*6*6}{1*3*3*5*5*7}\cdots\cdots$$

CAPITULO 3

ARREGLOS Y CADENAS

OBJETIVOS:
Al finalizar este capítulo, el alumno aprenderá a:

- Diferenciar entre un tipo simple y un tipo estructurado.
- Aprenderá a diferenciar entre un dato primitivo y dato de tipo estructurado.
- Acceder a los elementos de un vector y de una matriz en java.
- Inicializar elementos de un arrays
- Pasar arrays a un método y distinguir paso por valor y paso por referencia.
- Conocer algoritmos sencillos de ordenación de arrays.

3.1 Definiendo Arreglos

Los arreglos son estructuras de datos de tipo estáticas dado que antes de su utilización necesariamente tenemos que definir el tamaño de estas y no podemos manejar este tipo de asignación en tiempo de ejecución dado que por la misma naturaleza de la estructura no lo permite.

Los arreglos se caracterizan por que guardan datos primitivos que nos permitirán formas estructuras de datos más complejas. En Java los arreglos se definen de la siguiente manera.

```
class test
public static void main(String args[]){
    int n = new int[5]; //Arreglo de enteros
     flota notas;
     notas = new flota[26]; //Definimos un arreglo tipo float
}
```

En el ejemplo anterior hemos definido un el arreglo n donde guardamos datos primitivos de tipo entero, cuando definimos un arreglo todas las posiciones del arreglo tienen como dato guardado ***null*** a menos que se diga lo contrario.

También tenemos diferentes tipos de declaraciones de un array :

1. char cad[], p;

cad es un array tipo char y p es una variable tipo char

2. int []v, w;
   Tanto v como w son arreglos tipo int.

El tamaño es una constante representada por una variable protegida ***final.***

final int N=20;

float vector[];

vector =new float[N];

3.2 Subíndices de una array.

Ejemplos :

int edad = new int[5];

Este array contiene cinco elementos donde el primer elemento es : edad[0]…edad[4].

int [] pesos, longitudes

Declaramos dos array de enteros (no asignamos tamaño).

Racional [] ra = Racional[5]

Definimos un arreglo de cinco objetos racional

El almacenamiento de los array en memoria se almacena en bloques contiguos. Sí por ejemplo los array

```
int edades[];
char codigo[];
edades = new int [5];
codigos = new char[5];
```

tengamos presente que todos los subíndices de los arreglos en java empiezan con cero.

3.3 El tamaño de los array . Atributo length

Java considera el array como un objeto, debido a ello se puede conocer el número de elementos de un array accediendo al campo ***length***. Este campo resulta muy útil cuando se pasa un array a un método.

```
double [] v= new double[5];
System.out.println(v.length);

double suma (double [] w) {
double suma =0.0;
        for (int i=0; i< w.length; i++)
                s+=w[i];
        return s;
}
```

También podemos inicializar un arreglo como en los siguientes casos:

int numeros []= {10,20,30,40,60}
//Definimos un array de 6 elementos y se inicializan.
int n[] ={3,4,5};
//Definimos un array de 3 elementos.

char c[] ={'L','U',`i`,`s`} ***//Definimos un array de 4 elementos***

presentar un ejemplo de un arreglo donde se ingresan un conjunto elementos de un arreglo .

Los array en java se inicializan en cero por defecto en java como se muestra en el siguiente ejemplo:

```
int lista[]=new int [10];
for(int j=0; j<=9;j++)
  System.out.println("\n Lista "+j+"="+lista[i]);
```

3.4 ARRAY DE CARACTERES Y CADENAS DE TEXTO.

Java soporta cadenas de texto utilizando la clase `String y StringBuffer` implantada en el paquete `java.lang.`

Es importante diferenciar entre un array de caracteres y una cadena de caracteres. Los String son objetos en los que se almacenan las cadenas, tienen diversos constructores y métodos. Los array de tipo char son una secuencia de caracteres, con las mismas caracteristicas que los array de otro tipo.

String mas ="Programador en Java"; *// Crea objetos cadena.*
Char datos[]={'F','i','c','h','e','r','o'}; *//Definimos un array de 7 elementos.*

Las cadenas definidas como objetos StringBuffer pueden cambiar la longitud de la cadena y el contenido.

```
StringBuffer cc = new StringBuffer("Cadena Variable");
cc.replace('v','V');
```

La clase String tiene definido un método legth() de la clase String. Una confusión habitual es obtener la longitud de un array con una llamada **length();** o a la inversa, obtener la longitud de una cadena con el campo **length.**

```
String buf ="Cadena de Oro";
char acad[] = new char(6);
for(int j=0; j<6; j++)
   acad[j]=(char)'a';
```

```
    acad[j]=(char)'a'+j;
System.out.println("La longitud de buf =  "+buf.length());
System.out.println("La longitud de acad =  "+buf.length);
```

3.5 COPIA DE ARRAY

Los elementos de un arrays se pueden asignar a otro arrays con bucle que recorra cada elemento, el arrays destino tiene que estar definido con al menos los mismos elementos Así como en el ejemplo.

```
final int N=12;
int v1[]= new int [N],v2[]=new int[N];
for(int i=0; i<N;i++)
  v1[i]=(int)(Math.random()*199+1);
//Los elementos son copiados de v1 a v2.
for(int i=0;i<N;i++)
  v2[i]=v1[i];
```

La forma sintáctica de llamadas al método ***arraycopy***:	
System.arraycopy(arrayOrigen,inicioOrigen,arrayDestino,inicioDestino,elementos)	
arrayOrigen:	Nombre del array desde el que se va copiar
inicioOrigen:	Posición del array origen desde la se inicia la copia.
arrayDestino:	Nombre del array en el que se hace la copia
inicioDestino:	Es la posición del array destino donde empieza la copia
Elementos:	Numero de elementos del array origen que se va a copiar.

3.6 ARRAY MULTIDIMENSIONALES

Los array multidimensionales son aquellos que tienen más de dimensión y, en consecuencia, más de un índice. Los array más usuales son los de dos dimensiones, conocidos también con el nombre de *tablas* o matrices. Sin embargo, es posible crear array de tantas dimensiones como requieran sus aplicaciones esto es, tres, cuatro o más dimensiones.

Ejemplos de declaración de tablas:

```
char pantalla[][]= new char[80][24];
int puestos[][] =new int[10][5];
final int N=4;
double [][]matriz = new double[N][N];
```

3.6.1 INICIALIZACION DE ARRAY MULTIDIMENSIONALES

- int tabla[][] ={{51,52,53},{54,55,56}};
  Se ha definido una matriz de dos filas tres columnas cada fila.
  O bien en estos casos otros formatos más amigables:

```
int tabla[][]={{51,52,53},
               {54,55,56}};
```

- int tabla2[][] ={
  {1,2,3,4},
  {5,6,7,8},
  {9,10,11,12}
  };

Java trata los arrays de dos o más dimensiones como array de arrays, por ello se pueden crear arrays de dos dimensiones no cuadradas. En los siguientes ejemplos se crean arrays de distintos elementos cada fila.

- double tb[][] ={{1.5,-2.5},{5.0,-0.0,1.5}};
  Se ha definido una matriz de dos filas, la primera columnas y las segunda con tres.
- int[] a={1,3,5}, b={2,4,6,8,10};
  int mtb[][]={a,b};

  Se ha definido el array a de tres elementos, el b de cuatro elementos y la matriz mtb de dos filas, la primera con tres elementos o columnas, y la segunda con cuatro.

3.6.2 ACCESO A LOS ELEMENTOS DE LOS ARRAY BIDIMENSIONALES.

Ejemplos de inserciones:

```
Tabla[2][3]=4.5;
Resistencias[2][4]=50;

Ventas=Tabla[1][1];
Dia=semana[3][6];
```

Ejemplos de Lectura y Escritura de elementos de arrays bidimensionales

```
int tabla[][] = new int[3][4];
tabla[1][1]=Integer.parseInt(entrada.readLine());
System.out.println(tabla[1][1]+" =");
```

Ejemplo de Multiplicar dos Matrices de dos dimensiones.

```
public class Test1{
  static void MultiplicarPorDosMatriz2D(double[][] x) {
    for (int f = 0; f < x.length; f++) {
      for (int c = 0; c < x[0].length; c++)
```

```
        x[f][c] *= 2; }
  }

  public static void main(String[] args)  {
    double[][] m = {{10, 20, 30}, {40, 50, 60}};
    MultiplicarPorDosMatriz2D(m);
    // Visualizar la matriz por filas
    for (int f = 0; f < m.length; f++) {
      for (int c = 0; c < m[0].length; c++)
          System.out.printf("%10.2f", m[f][c]);
          System.out.println();
    }
  }
}
```

Salida por Pantalla

20,00 40,00 60,00
80,00 100,00 120,00

3.6.2. UTILIZACIÓN DE ARRAY COMO PARÁMETROS.

En java, todas las variables de tipo primitivos, *double, flota, char, int, bolean* se pasan por **valor**. ***Los objetos siempre se pasan por referencia***. Los array en java se consideran que son objetos y por tanto se pasan por referencia (*dirección*). Esto significa que cuando se llama a un método o función y se utiliza un array como parámetro, se debe tener cuidado de no modificar el array en una función llamada.

Dadas las declaraciones:

```
final int Max =100;
double datos[] = new double [Max];
```

se puede declarar un método que acepte un array de valores double como parámetro. El método SumaDeDatos() tiene la cabecera:

```
double SumaDeDatos(double w[]);
```

En la llamada al método SumaDeDatos() el argumento array se pone escribiendo el identificador del array:

```
SumaDeDatos(datos);
```

El método SumaDeDatos() no es difícil de escribir. Un simple bucle for suma los elementos del array y una sentencia return devuelve el resultado de nuevo al llamador:

```
double SumaDeDatos(double w[]){
        double suma=0.0;
        int n=w.length;
        while(n>0)
           suma +=W[--n];
        return suma; }
```

3.7 STRINGS.

Los *strings* (o cadenas de caracteres) en Java son objetos y no vectores de caracteres como ocurre en C.

Existen dos clases para manipular strings: `String` y `StringBuffer`. `String` se utiliza cuando las cadenas de caracteres no cambian (son constantes) y `StringBuffer` cuando se quiere utilizar cadenas de caracteres dinámicas, que puedan variar en contenido o longitud.

3.7.1 La clase String.

Como ya se ha mencionado, se utilizarán objetos de la clase `String` para almacenar cadenas de caracteres constantes, que no varían en contenido o longitud.

Cuando el compilador encuentra una cadena de caracteres encerrada entre comillas dobles, crea automáticamente un objeto del tipo `String`.

Así la instrucción:

```
System.out.println( "Hola mundo");
```

Provoca que el compilador, al encontrar "Hola mundo", cree un objeto de tipo `String` que inicializa con el *string* "Hola mundo" y este objeto es pasado como argumento al método `println()`.

Por la misma razón, también puede inicializarse una variable de tipo `String` de la siguiente forma:

```
String s;
s = "Hola mundo";
```

o

```
String s = "Hola mundo";
```

Además, por ser `String` una clase, también puede inicializarse una variable de dicha clase mediante su constructor:

```
String s;
```

o

```
String s = new String("Hola mundo");
```

Se ha mencionado que la clase `String` sirve para almacenar cadenas de caracteres que no varían, sin embargo es posible hacer lo siguiente:

```
String s = "Hola mundo";
s = "El mensaje es: " + s;
```

Esto no significa que la longitud de "`s`" pueda ser variada. Lo que ha ocurrido en realidad es que el compilador ha creado un objeto de la clase `String` al encontrar "El mensaje es: ", después ha realizado la operación de concatenación de strings entre ese objeto creado y s; y el resultado a sido asignado a un nuevo objeto creado por el compilador; este objeto es asignado a la variable s (el puntero de s apunta al nuevo objeto) y el objeto al que apuntaba anteriormente s ya no tiene ninguna referencia, por lo que el "recolector de basura" liberará la memoria utilizada por el mismo al detectar su inaccesibilidad.

El operador + está sobrecargado y puede concatenar `String`s con enteros, `StringBuffer`s, etc.

La siguiente expresión es correcta:
String s = "Quiero" + 2 + "cafés";

Puede concatenarse un `String` con cualquier objeto. El resultado es el `String` original concatenado al `String` que devuelve el método `toString()` que toda clase hereda de la clase raíz `Object` y que puede estar, o no, redefinido.

Pruébese a pasar como parámetro un objeto de cualquier clase al método `System.out.println()`.
Por ejemplo:

```
class Clase {
}
class MuySimple {
        public static void main(String arg[]) {
                Clase c = new Clase();
                System.out.println(c);
        }
}
```

Produce la salida:
`Clase@1cc731`
(Resultado de llamar al método `toString()` de la clase **Clase**.)

3.7.1.1 Constructores de la clase String.

- **public String();**
  Construye un *string* vacío.
- **public String(byte bytes[]);**
  Construye un *string* a partir de un vector de bytes codificados dependiendo de la plataforma (por ejemplo, ASCII), por lo tanto, la longitud del vector no tiene porqué coincidir siempre con la del *string*.

  Ejemplo: byte bytes[]={65,66};
  String s = new String(bytes); → "AB"

- **public String(byte bytes[], int posición, int cantidad);**
  Construye un *string* a partir de un vector de bytes codificados dependiendo de la plataforma. Para ello toma el primer byte que se encuentra en la posición indicada como parámetro y tantos bytes del vector como se indica en el parámetro "cantidad".

  Ejemplo: byte bytes[]={65,66,67,68,69};
  String s = new String(bytes, 1 , 3); → BCD
- **public String(char valor[])**
  Construye un *string* inicializado por un vector de caracteres (valor).

Ejemplo: char c[] = {'H','o','l','a'};
String s = new String(c);

- **public String(char valor[], int posición, int cantidad);**
  Construye un *string* inicializado por un subvector de caracteres. El primer carácter del *string* será el indicado por el primer entero (posición), teniendo en cuenta que los vectores comienzan por el índice cero. La longitud del *string* será especificado por el segundo entero (cantidad).

  Ejemplo: char c[] = {'H','o','l','a'};
  String s = new String(c , 1 , 2); □□"ol"
- **public String(String valor);**
  Construye un `String` que es copia del `String` especificado como parámetro.
  Ejemplo: String s = new String("Hola mundo");
- **public String(StringBuffer buffer);**
  Construye un nuevo String a partir de un StringBuffer.

Lectura de Cadenas.

La clase *BufferedReader* tiene diversos métodos para la lectura de datos. Al ejecutarse un programa, Java incorpora el objeto System.in asociado con el flujo (stream) de entrada byte a byte. Además la clase *InputStreamReader* convierte el flujo de byte en cadenas de caracteres. Esa es la razón de que se cree el objeto entrada(El nombre puede ser cualquiera) para lectura desde el teclado:

```
BufferedReader entrada= new BufferedReader(
                        new InputStreamReader(System.in));
```

El método readLine() es el más utilizado para entrada de datos desde el teclado, pertenece a la clase BufferedReader, devuelve un objeto cadena con los caracteres leídos hasta encontrar la marca de fin de línea. El método devuelve null si encuentra la marca de fin de fichero.

Ejemplo:
El programa que escribe y lee líneas de caracteres y se escriben en pantalla.

```
import java.io.*;
public class EntradaCadenas {
        public static void main(String args[])throws IOException{
                int ncd =0;
                String micd;
                BufferedReader entrada = new BufferedReader(
                        new InputStreamReader(System.in));
                System.out.println("Ingrese una cadena de caracteres ;"+
                        "Termina Crtl+Z");
                System.out.flush();
                do{
                        micd=null;
                        micd= entrada.readLine();
                        if(micd !=null){
                                ncd++;
```

```
                        System.out.println(micd);
                    }
                }
                while (micd !=null);
                System.out.println("\nNumero de Lineas :"+ncd);
        }
}
```

Método read()

Java considera la entrada o la salida de datos como un flujo de caracteres. La clase **System** incorpora los objetos **out, in y err** que son creados automáticamente al ejecutar el programa. El objeto out está asociado a una salida estándar, el método print() y println() se utiliza con mucha frecuencia para la salida de cadenas de caracteres. El objeto in está asociado con la entrada estándar, tiene métodos para la lectura de caracteres. El método read() lee un carácter del dispositivo estándar de entrada, normalmente el teclado, y devuelve el carácter leído.

```
int read();
```

El método está sobrecargado para poder leer de la entrada un array de caracteres o un rango de caracteres :

```
int read(byte [] dat);
int read(byte [] dat, int posición, int numCaracteres);
```

Ejemplo :
El siguiente programa lee un texto desde el teclado, cuenta el número de letras i,j,k,l,m,n (*tanto en mayúsculas como minúsculas*).

```
import java.io.*;
public class CuentaCar {
                public static  void main(String args[])throws IOException {
                int  car;
                int cuenta=0;
                System.out.println("\nIngrese el texto , termina con #");
                while((car=System.in.read())!='#')
                        if((car>='i'&&car>='n' )||(car>='I'&&car>='N' )){
                                ++cuenta;
                        }

                System.out.println("Numero de Ocurrencias : "+cuenta);
        }
}
```

LONGITUD Y CONCATENACIÓN DE CADENAS

Longitud de una cadena : Método length()

La clase String tiene definido el método length(), que calcula el número de caracteres del objeto cadena :

```
String cad ='1234567890'
int i = cad.length();
```

En estas sentencias asigna 10 a la variable i

Concatenación de cadenas:

Concatenación de cadmeas con el operador +

```
String c1 ='Angela';
String c2 ='Paloma';
String c3= c1+c2;
String cd;
cd= 'Musica'+'Clasica';
```

Concatenación de cadeneas : ***Método concat()***

La clase String tiene definido el método concat() para unir cadenas y devolver un nuevo objeto cadena.

```
String concat(String);

String dst ='Desconocido';
String org='Rutina';
System.out.println('Concatena = '+dst.concat(org));
System.out.println("Cadena dst = +dst);// dst no ha cambiado
System.out.println("Cadena dst = +org);// org no ha cambiado.
```

Obtención de caracteres de una cadena

Obtención de un carácter : Método charAt()

El método charAt() permite obtener un carácter de una cadena. Tiene como parámetro el índice del carácter a obtener. El índice inferior es el cero, como ocurre en los arrays, y el índice superior length() -1. Si el indice esta fuera de rango el programa envía una excepción un error en tiempo de ejecución.

```
"ABCDEF".charAt(0) // Devuelve el carácter 'A'
String v = new String("Nuestro Honor");
System.out.print(v.charAt[0]+ v.charAt[2]+ v.charAt[4]);
```

En el ejemplo siguiente se determina el número de vocales que tiene una cadena.

```
public class CnumVocales {

   public static int numVocales(String cadena) {
        int k=0;
        for(int i=0;i<cadena.length();i++)
                        switch(cadena.charAt(i))
                {
                        case 'a': case'e': case 'i':case'o': case'u':
                        case 'A': case'E': case 'I': case 'O': case 'U':
                        k++;
                }
                return k;
        }


public static void main(String [] args){
String mensaje = new String("Universidad Nacional de Piura");

System.out.println("El número de Vocales de la Frase es :"+numVocales(mensaje));
```

```
    }
}
```

Dejando propuesto para el alumno que determine el número de vocales de cada tipo tiene una frase ingresada por teclado.

Obtención de un array de caracteres : Métodos getChars()

Con el método getChars(), un rango de caracteres de una cadena se copia en un array de tipo char aunque realmente prefiero implementar mis propia rutina para que realice esta funcionalidad.

En las siguientes sentencias se extraen caracteres de una cadena y se almacenan en un array.

```
final int M=10;
String bs ='Hoy es un buen día';
char ds[] = new char[M];
bs.getChars(0,2,ds,0); //ds contiene los caracteres  "Ho" en la posición 0 y 1
bs.getChars(4,5,ds,0); // ds contiene los caracteres "puede"en la posición 0…4//
bs.getChars(10,3,ds,5);// los caracteres "ser se copian en las posiciones 5…7" de ds.
bs.getChars(0,bs,length()/2,ds,0); // la mitad de los caracteres de las cadena se copian
                                   // en ds a partir de la posición 0.
```

Obtención de una subcadena : Método substring()

Una cadena se puede descomponer en subcadenas, el método substring() obtiene una subcadena de la cadena original que se devuelve como otro objeto cadena. El método tiene como argumentos la posición inicial y final; el rango de caracteres obtenido va desde inicial hasta final-1.

```
String substring(int inicial, int final);
```

Ejemplo :

```
String dc ='Terminal Inteligente'
System.out.println(dc.substring(3,8));//minal, caracteres 3…7
System.out.println(dc.substring(9,dc.length()) //"inteligente"
```

Comparación de cadenas

Dado que las cadenas son objetos que contienen una secuencia de caracteres, la Clase String proporciona un conjunto de métodos que comparan cadenas alfabéticamente. Estos métodos comparan los caracteres de dos cadenas utilizando el código numérico de su representación. Los métodos son:

- equal
- equalIgnoreCase()
- regionMatches()
- compareTo()
- startsWith()
- endsWith()

Método compareTo()

Cuando se desea determinar si una cadena es igual a otra, o mayor o menor que otra, debe utilizar el método **compareTo()** . La comparación siempre es alfabética.

compareTo() compara la cadena que llama al método con la cadena que se pasa como argumento, y devuelve 0 si las dos cadenas son idénticas, un valor menor que cero si la cadena 1 es mayor que la cadena 2, o un valor mayor que cero si la cadena 1 es mayor que la cadena 2 (*los términos mayor que y menor que* se refieren a la ordenación alfabética de las cadenas).

int compareTo(cadena 2)

< 0	si	cad1	es menor que	cad2
=0	si	cad1	es igual a	cad2
>0	si	cad1	es mayor que	cad2

```
public class CCompateTo {
        public static void main(String [] args){
        String c1="Universo Java";
        String c2="Universo Visual J";
        int i;
        i=      c1.compareTo(c2);
        if (c1.compareTo(c2)<0){
                System.out.println(c2);
        }
        else
        {
                System.out.println(c1);
        }
        System.out.println("El valor de i = "+i);
        //Otros Casos
        System.out. println("Compara Windows con Waterloo
:"+"Windows".compareTo("Waterloo"));
        System.out.println("Windows es mayor que Waterloo");
          }
        }
```

Salida en Pantalla es :

```
Universo Visual J
El valor de i = -12
Compara Windows con Waterloo  :8
Windows es mayor que Waterloo
```

CONVERSIÓN DE CADENAS

Método toUpperCase( );

Este método devuelve una cadena con los mismos caracters que la cadena que llama al método, excepto las letras minúsculas, se convierten en mayúsculas

```
String org= "la ducle vida";
        System.out.println(org.toUpperCase());
```

Salida:

LA DUCLE VIDA

Método toLowerCase( );

Este método devuelve una cadena con los mismos caracteres que la cadena que llama al método, excepto las letras mayúsculas, que se convierten en minúsculas

```
String org= "la Casa Vieja";
System.out.println(org.toLowerCase()); //escribe LA DULCE VIDA
```

Método replace( );

Este método crea una nueva cadena en la que se ha sustituido todas las ocurrencias de un carácter por otro . El método tiene dos argumentos , el primero representa el carácter de la cadena origen que va a ser cambiado y el segundo argumento el carácter que le sustituye .

```
String org= "la Casa Vieja";
System.out.println(org.replace(' ','#')); //salida : la#Casa#Vieja
```

Método toCharArray( );

Este método devuelve un array con los caracteres de la cadena que llama al método. Para ello crea un array de tantos elementos como la longitud de la cadena y copia los caracteres :

```
String cad ="Ventana";
char[] ac =cad.toCharArray();
```

3.7.2 La clase StringBuffer.

La clase String tiene una característica que puede causar problemas, y es que los objetos String se crean cada vez que se les asigna o amplia el texto. Esto hace que la ejecución sea más lenta. Este código:

```
String frase=“Esta “;

frase += “es ”;
frase += “la ”;
frase += “frase”;
```

En este código se crean cuatro objetos String y los valores de cada uno son copiados al siguiente. Por ello se ha añadido la clase **StringBuffer** que mejora el rendimiento. La concatenación de textos se hace con el método **append:**

```
StringBuffer frase = new StringBuffer(“Esta ”);
frase.append(“es “);
frase.append(“la “);
```

```
frase.append("frase.");
```

Por otro lado el método **toString** permite pasar un **StringBuffer** a forma de cadena String.

```
StringBuffer frase1 = new StringBuffer("Valor inicial");
...
String frase2 = frase1.toString();
```

Se recomienda usar StringBuffer cuando se requieren cadenas a las que se las cambia el texto a menudo. Posee métodos propios que son muy interesantes para realizar estas modificaciones (**insert, delete, replace,…**).

3.7.2.1 Constructores de la clase StringBuffer.

- **public StringBuffer();**
  Crea un **StringBuffer** vacío.
- **public StringBuffer(int longitud);**
  Crea un **StringBuffer** vacío de la longitud especificada por el parámetro.
- **public StringBuffer(String str);**
  Crea un **StringBuffer** con el valor inicial especificado por el **String**.

3.7.2.2 Métodos de la clase StringBuffer.

De forma análoga a lo que ocurre con la clase **String**, la clase **StringBuffer** proporciona una gran cantidad de métodos que se encuentran enumerados en

StringBuffer **append(***tipo* variable**)** Añade al *StringBuffer* el valor en forma de cadena de la variable.

char **charAt(**int pos**)** Devuelve el carácter que se encuentra en la posición *pos.*

int **capacity()** Da como resultado la capacidad actual del *StringBuffer*

StringBuffer **delete(**int inicio, int fin) Borra del *StringBuffer* los caracteres que van desde la posición *inicio* a la posición *fin*

StringBuffer **deleteCharAt(**int pos**)** Borra del *StringBuffer* el carácter situado en la posición *pos* .

void **ensureCapacity(**int capadMinima**)** Asegura que la capacidad del *StringBuffer* sea al menos la dada en la función.

void **getChars(**int srcInicio, int srcFin,char[] dst, int dstInicio**) :**Copia a un array de caracteres cuyo nombre es dado por el tercer parámetro, los caracteres del *StringBuffer* que van desde *srcInicio* a *srcFin.* Dichos caracteres se copiarán en el array desde la posición *dstInicio*

StringBuffer insert(int pos, *tipo* valor**)** Inserta el valor en forma de cadena a partir de la posición *pos* del *StringBuffer*.

int length() Devuelve el tamaño del *StringBuffer* .

StringBuffer replace(int inicio**, int** fin**, String** texto**)** Reemplaza la subcadena del *StringBuffer* que va desde *inicio* a *fin* por el *texto* indicado.

StringBuffer reverse() Se cambia el *StringBuffer* por su inverso

void setLength(int tamaño**)** Cambia el tamaño del *StringBuffer* al tamaño indicado.

String substring(int inicio**)** Devuelve una cadena desde la posición inicio.

String substring(int inicio, **int** fin**)** Devuelve una cadena desde la posición *inicio* hasta la posición *fin.*

String toString() Devuelve el *StringBuffer* en forma de cadena

Ejemplo de cadena Inversa hacemos uso de un método que le da vuelta a una cadena

```
class CadInversa {
   public static String cadenaInversa( String fuente ) {
      // Se obtiene la longitud de la cadena que se pasa
      int longitud = fuente.length();

      // Se crea un stringbuffer de la longitud de la cadena
      StringBuffer destino = new StringBuffer( longitud );

      // Se recorre la cadena de final a principio, añadiendo
      // cada uno de los caracteres leidos al stringbuffer
      for( int i=(longitud-1); i >= 0; i-- )
         destino.append( fuente.charAt( i ) );

      // Devolvemos el contenido de la cadena invertida
      return( destino.toString() );
      }

   public static void main( String args[] ) {
      // Imprime el resultado inverit la cadena que se toma por
      // defecto
      System.out.println( cadenaInversa( "Hola Mundo" ) );
      }   }
```

3.7.3 Búsqueda de caracteres y cadenas.

La clase String contiene dos métodos que permiten localizar carcatres en cadenas y subcadenas en cadenas. Estos métodos son **indexOf()** y **lastIndexOf();** pueden llamarse para buscar un carácter o una cadena, por tanto están sobrecargados con diversas versiones.

3.7.3.1 Método *indexOf()*

Permite buscar caracteres y patrones de caracteres (*SubCadenas*) en cadenas. En general localiza la primera ocurrencia del argumento en la cadena que llama al método. Devuelve la posición de la primera ocurrencia del carácter en la cadena; si no encuentra el carácter devuelve -1. El argumento del método es la representación entera del caracter. Normalmente se bucan caracteres, por lo que hay que hacer un cast a int en la llamada. Por ejemplo:

int indexOf(int c)

Así, por ejemplo, se quiere buscar 'v' en una cadena pat

```
String pat= Java, lenguaje de alto nivel;
int k;
k=pat.indexOf((int)'v');
```

en este caso en cuentra 'v' en la posición 2.

La búsqueda se inicia a partir de la posición 0; para iniciar la búsqueda a partir de otra posición hay una versión de indexOf() con un segundo argumento de tipo entero que representa dicha posición:

int indexOf(in c, int p);

Con el método indexOf( ) también se puede bucar un patrón de caracteres o subcadenas; en este caso el argumento es un objeto cadena con los caracteres a buscar. El método devuelve la posición de inicio de la primera ocurrencia, o bien -1 si no es localizada.

int indexOf(String patron)

por ejemplo:

```
String pat="La familia de programadores de alto nivel";
int k;
k=pat.indexOf("ama");
```

en esta búsqueda encuentra la cadena en la posición 19, que se asigna a la variable k.

Para que la búsqueda de la subcadena se inicie en una posición distinta de la cero se pasa un segundo argumento entero con la posición en la comienza la búsqueda.

int indexOf(string patron , int p)

por ejemplo :

```
String pat=("La familia de programadores de alto nivel");
int k;
k=pat.indexOf("de", 17)
```

Esta búsqueda se inicia a partir de la posición 17, encuentra la cadena en la posición 27

EJERCICOS RESUELTOS

1. Escribir un programa, que lea una lista de números enteros positivos y los muestre en pantalla. Luego, si hay números repetidos, deben eliminarse de la lista y dejar sólo uno de cada número e imprimir la nueva lista.

```
class CMatrizTest01 {
    public static void main(String args[]){
      final int MAX=10;
      int num[]= new int[MAX];

      leenum(num);
      presentar(num);
      eliminar(num);
      System.out.println("==============Lista Depurada============");
      presentar(num);
    }
  public static  void leenum(int a[]){
    int ter=0;
    System.out.println("Generamos los Términos del Arreglo de Manera Aleatoria");
            for(int i=0;i<a.length;i++){
                    ter=(int)(Math.random()*10);
                    a[i]=ter;
            }
  }
  public static void presentar(int a[]){
     for(int i=0; i<a.length;i++){
            System.out.println("A["+i+"]= "+a[i]);
     }
  }

  public static void eliminar(int a[]){
      int val=-1;
      int aux[]=new int[a.length];
      for(int i=0; i<a.length;i++) { //copiamos los elementos de a aux.
            aux[i]=a[i];
      }
      /*Buscamos Datos Repetidos y si lo hay*/
      /*se remplaza su posición por val     */
      for(int i=0;i<(a.length-1);i++)
            for(int j=i+1;j<a.length;j++){
                    if(aux[i]==aux[j])
                            aux[i]=val;
            }

       int j=0;
       for(int i=0;i<aux.length;i++){
             if(aux[i]!=val){
                    a[j]=aux[i];
                     j++;
            }
       }
```

```
        for(int K=j;K<a.length;K++){
                a[K]=0;
        }

    }
}
```

2. Escribir un programa que dados dos arreglos numéricos a y b, de n1 y n2 elementos respectivamente guarden en un tercer arreglo c, todos los números que están en a pero no están en b.

```
class CArregloTest02 {
    public static void main(String args[]){
      final int MAX=10;
      int a[]= new int[MAX];
      int b[]= new int[MAX];
      int c[]= new int[MAX];
      leenum(a);
      leenum(b);
      System.out.println("ARREGLO A :        ");
      presentar(a);
      System.out.println("ARREGLO B :        ");
      presentar(b);
      separar(a,b,c);
      System.out.println("ARREGLO C :        ");
      presentar(c);
    }
  public static  void leenum(int A[]){
    int ter=0;
    System.out.println("Generamos los Términos del Arreglo");
            for(int i=0;i<A.length;i++){
                    ter=(int)(Math.random()*10);
                    A[i]=ter;
            }
  }
  public static void presentar(int A[]){
     for(int i=0; i<A.length;i++){
     /*      System.out.println("A["+i+"]= "+A[i]);*/
     System.out.println(" "+A[i]);
     }
  }
  public static void separar(int a1[],int b1[],int c1[]){
    int x,y;
    int j,i,band=0,k=0;
    for(i=0 ; i<a1.length;i++){
            for(j=0;j<b1.length;j++)
                    if(a1[i]==b1[j]){
                            band=1;
                            break;
                    }
                    if(band==0){
                            c1[k]=a1[i];
                            k++;
                    }
                    else band=0;
     }
  }
}
```

EJERCICIOS PROPUESTOS

1. Escribir un programa que reciba como parámetros 2 cadenas de caracteres y determine si son iguales o si son diferentes y cual es la mayor.

2.- Si x representa la media de los números $x_1, x_2, \ldots x_n$ entonces la varianza es la media de los cuadrados de las desviaciones de los números de la media.

$$\text{var}\,ianza = \frac{1}{n}\sum_{i=1}^{n}(x_i - x)^2$$

Y la desviación estándar es la raíz cuadrada de la varianza. Escribir un programa que lea una lista de números reales, los cuente y a continuación calcule e imprima su media, varianza y desviación estándar. Utilizar un método para calcular la media, otro para la varianza y otro para la varianza.

3.-Escriba un programa que determine si una palabra es palíndromo. Un palíndromo es un array de caracteres que se lee de igual forma en ambos sentidos; por ejemplo ana.

4.- Escribir un programa en el que se genere aleatoriamente un vector de 20 números enteros. El vector ha de quedar de tal forma que la suma de los 10 primeros elementos sea mayor que la suma de los 10 últimos elementos. Mostrar el vector original.

5.-Dada una cadena fuente y una secuencia de caracteres guardados en un array, escribir un método que devuelva la posición de la primera ocurrencia de cualquiera de los caracteres del array cadena.

CAPITULO 4

CLASES Y METODOS

OBJETIVOS:
Al finalizar este capítulo, el alumno aprenderá a:

- **Definir una clase y crear una instancia de la clase**
- **Manejara los modificadores de acceso de las clases y de sus métodos.**
- **Aprender a definir un constructor de clase y sobre carga de estos**
- **Conocer sobrecarga de métodos.**

4.1 Declaración de clase.

La declaración mínima para una clase es la siguiente:

class NombreClase

Una declaración de este tipo indica que la clase no desciende de ninguna otra, aunque en realidad, todas las clases declaradas en un programa escrito en Java son descendientes, directa o indirectamente, de la clase `Object` que es la raíz de toda la jerarquía de clases en Java.

```
class ObjetoSimpleCreado {
        String variable = "Una variable";
        int entero = 14;
        public String obtnerString() {
        return variable;
   }
}
class ObjetoSimple {
                public static void main(String arumentos[]) {
                ObjetoSimpleCreado varObj = new ObjetoSimpleCreado();
                System.out.println(varObj.toString());
        }
}
```

Muestra en pantalla la siguiente línea de texto:
`ObjetoSimpleCreado@13937d8`

En este caso, la clase `ObjetoSimpleCreado` ha sido declarada como no descendiente de ninguna otra clase, pero a pesar de ello, hereda de la superclase `Object` (`java.lang.Object`) todos sus métodos, entre los que se encuentran el método `toString()` que, en este caso, devuelve el siguiente valor: “ObjetoSimpleCreado@13937d8” (el nombre de la clase junto con el puntero al objeto). Este método, que heredan todas las clases que puedan declararse, debería ser redefinido por el programador para mostrar un valor más

significativo. Si en lugar de la instrucción `System.out.println(varObj.toString());` se hubiera utilizado la siguiente: `System.out.println(varObj.obtenerString())` la salida por pantalla habría sido:

Una variable

4.2 Modificadores de clase.

Los modificadores de clase son palabras reservadas que se anteponen a la declaración de clase.

Los modificadores posibles son los siguientes:

- `public`.
- `abstract`.
- `final`.

La sintaxis general es la siguiente:

modificador class NombreClase [extends NombreSuperclase] [implements listaDeInterfaces]

Si no se especifica ningún modificador de clase, la clase será visible en todas las declaradas en el mismo paquete[16]. Si no se especifica ningún paquete, se considera que la clase pertenece a un paquete por defecto al cual pertenecen todas las clases que no declaran explícitamente el paquete al que pertenecen.

Si no se especifica ningún modificador de clase, la clase será visible en todas las declaradas en el mismo paquete Si no se especifica ningún paquete, se considera que la clase pertenece a un paquete por defecto al cual pertenecen todas las clases que no declaran explícitamente el paquete al que pertenecen.

4.2.1 public.

Cuando se crean varias clases que se agrupan formando un paquete (*package*), únicamente las clases declaradas `public` pueden ser accedidas desde otro paquete.

Toda clase public debe ser declarada en un fichero fuente con el nombre de esa clase pública: NombreClase.java. De esta afirmación se deduce que en un fichero fuente puede haber más de una clase, pero sólo una con el modificador public.

4.2.2 abstract.

Las clases abstractas no pueden ser instanciadas sirven únicamente para declarar subclases que deben redefinir aquellos métodos que han sido declarados `abstract`**.** Esto no quiere decir que todos los métodos de una clase abstracta deban ser abstractos, incluso es posible que ninguno de ellos lo sea. Aún en este último caso, la clase será considerada como abstracta y no podrán declararse objetos de esta clase.

Cuando alguno de los métodos de una clase es declarado abstracto, la clase debe ser obligatoriamente abstracta, de lo contrario, el compilador genera un mensaje de error. Todas estas clases se crean en el paquete por defecto.

```
abstract class Animal {
        String nombre;
        int patas;
        public Animal(String n, int p) {
                nombre=n;
                patas=p;
        }
   abstract void habla(){};
          // método abstracto que debe ser redefinido por las subclases
}

class Perro extends Animal {
        // La clase perro es una subclase de la clase abstracta Animal
        String raza;
        public Perro(String n, int p, String r) {
                super(n,p);
        raza=r;
    }
public void habla() {
        // Este método es necesario redefinirlo para poder instanciar
        // objetos de la clase Perro
        System.out.println("Me llamo "+nombre+": GUAU, GUAU");
        System.out.println("mi raza es "+raza);
   }
}

class Gallo extends Animal {
// La clase Gallo es una subclase de la clase abstracta Animal
        public Gallo(String n, int p) {
             super(n,p);
        }
        public void habla() {
        // Este método es necesario redefinirlo para poder instanciar
        // objetos de la clase Gallo
              System.out.println("Soy un Gallo, Me llamo "+nombre);
              System.out.println("Kikirikiiii");
        }
}
class Abstracta {
        public static void main(String argumentos[]) {
                Perro toby = new Perro("Toby",4,"San Bernardo");
                Gallo kiko = new Gallo("Kiko",2);
                kiko.habla();
                System.out.println();
                toby.habla();
        }
}
```

Salida por pantalla del programa:

```
Soy un Gallo, Me llamo Kiko
Kikirikiiii
Me llamo Toby: GUAU, GUAU
mi raza es San Bernardo
```

El intento de declarar un objeto del tipo Animal, que es `abstract`, habría generado un mensaje de error por el compilador. Las clases abstractas se crean para ser superclases de otras clases. En este ejemplo, se ha declarado el método `habla()` como abstracto porque queremos que todos los animales puedan hablar, pero no sabemos qué es lo que van a decir

(qué acciones se van a realizar), por lo que es declarada de tipo `abstract`**.** Las clases que heredan de `Animal` deben implementar un método `habla()` para poder heredar las características de `Animal`.

4.2.3 `final`.

Una clase declarada `final` impide que pueda ser superclase de otras clases. Dicho de otra forma, ninguna clase puede heredar de una clase `final`. Esto es importante cuando se crean clases que acceden a recursos del sistema operativo o realizan operaciones de seguridad en el sistema. Si estas clases no se declaran como final, cualquiera podría redefinirlas y aprovecharse para realizar operaciones sólo permitidas a dichas clases pero con nuevas intenciones, posiblemente oscuras.

A diferencia del modificador `abstract`, pueden existir en la clase métodos `final` sin que la clase que los contiene sea `final` (sólo se protegen algunos métodos de la clase que no pueden ser redefinidos). Una clase no puede ser a la vez `abstract` y `final` ya que no tiene sentido, pero sí que puede ser `public abstract` o `public final.`

4.3 El cuerpo de la clase.

Una vez declarada la clase, se declaran los atributos y los métodos de la misma dentro del cuerpo.

```
Declaración de clase
{
        Declaración de atributos
        Declaración de clases interiores
        Declaración de Métodos
}
```

La declaración de clases interiores (también conocidas como clases anidadas) no es imprescindible para programar en Java..

4.3.1 Declaración de atributos.

Los atributos sirven, en principio, para almacenar valores de los objetos que se instancian a partir de una clase.

La sintaxis general es la siguiente:

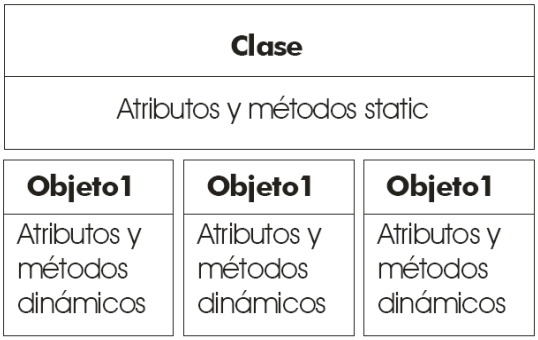
[modificadorDeÁmbito**] [static] [final] [transient] [volatile]** tipo nombreAtributo

Existen dos tipos generales de atributos:

- **Atributos de objeto.**
- **Atributos de clase.**

```
class Calculadora {
        static public int factorial(int n) {
        int fact=1;
                while (n>0) {
```

```
                        fact *=n--;
                }
        return fact;
        }
}
public class app {
    public static void main(String[] args) {
        System.out.println(Calculadora.factorial(5));
  }
}
```



En el ejemplo calculadora no ha hecho falta crear un objeto para calcular el factorial esto se puede realizar por que el método factorial de la clase calculadora se le ha indicado con que es stactic, caso contrario no hubiéramos podido utilizar la el método de la clase sin necesidad de de definir el un objeto previamente.

Los atributos de clase son variables u objetos que almacenan el mismo valor para todos los objetos instanciados a partir de esa clase, esto lo podríamos utilizar para el caso que necesitáramos contar cuantos objetos tenemos de una determinad clase.

Dicho de otra forma: mientras que a partir de un atributo de objeto se crean tantas copias de ese atributo como objetos se instancien, a partir de un atributo de clase sólo se crea una copia de ese atributo que será compartido por todos los objetos que se instancien. Si no se especifica lo contrario, los atributos son de objeto y no de clase. Para declarar un atributo de clase se utiliza la palabra reservada `static`.

La declaración mínima de los atributos es:

tipo nombreAtributo

Si existen varios atributos del mismo tipo (en la misma clase), se separan sus nombres mediante comas (**,**):

```
class Punto {
        int x, y;
        String nombre;
…
}
```

4.3.1.1 Atributos `static`.

Mediante la palabra reservada `static` se declaran atributos de clase.

```
class Persona {
        static int numPersonas=0; // atributo de clase
        String nombre; // atributo de objeto
        public Persona (String n) {
        nombre = n;
        numPersonas++;
}
        public void muestra() {
        System.out.print("Soy "+nombre);
```

```
        System.out.println(" pero hay "+ (numPersonas-1) +" personas más.");
    }
}

class Static {
public static void main(String argumentos[]) {
        Persona p1,p2,p3;
        // se crean tres instancias del atributo nombre
        // sólo se crea una instancia del atributo numPersonas
        p1 = new Persona("Pedro");
        p2 = new Persona("Juan");
        p3 = new Persona("Susana");
        p2.muestra();

        p1.muestra();
    }
}
```

Salida por pantalla:

```
Soy Juan pero hay 2 personas más.
Soy Pedro pero hay 2 personas más.
```

En este caso, `numPersonas` es un atributo de clase y por lo tanto es compartido por todos los objetos que se crean a partir de la clase `Persona`. Todos los objetos de esta clase pueden acceder al mismo atributo y manipularlo. El atributo nombre es un atributo de objeto y se crean tantas instancias como objetos se declaren del tipo `Persona`. Cada variable declarada de tipo `Persona` tiene un atributo nombre y cada objeto puede manipular su propio atributo de objeto. En el ejemplo, se crea un atributo `numPersonas` y tres atributos nombre (tantos como objetos de tipo `Persona`).

4.3.1.3 Atributos `final`.

La palabra reservada `final` calificando a un atributo o variable sirve para declarar constantes, no se permite la modificación de su valor. Si además es `static`, se puede acceder a dicha constante simplemente anteponiendo el nombre de la clase, sin necesidad de instanciarla creando un objeto de la misma. El valor de un atributo final debe ser asignado en la declaración del mismo. Cualquier intento de modificar su valor generará el consiguiente error por parte del compilador.

```
class Circulo {
        final double PI=3.14159265;
        int radio;
        Circulo(int r) {
                radio=r;
        }
        public double area() {
                return PI*radio*radio;
        }
}
class Final {
```

```
        public static void main(String argumentos[]) {
                Circulo c = new Circulo(15);
                System.out.println(c.area());
        }
}
```

Podría ser útil en algunos casos declarar una clase con constantes:

```
class Constantes {
        static final double PI=3.14159265;
        static final String NOMBREEMPRESA = "Ficticia S.A.";
        static final int MAXP = 3456;
        static final byte CODIGO = 1;
}
```

Para acceder a estas constantes, no es necesario instanciar la clase `Constantes`, ya que los atributos se han declarado `static`**.** Simplemente hay que anteponer el nombre de la clase: `Constantes.PI`, `Constantes.CODIGO`, etc. Para utilizarlas

4.3.2 Modificadores de ámbito de atributos.

Los modificadores de ámbito de atributo especifican la forma en que puede accederse a los mismos desde otras clases. Estos modificadores de ámbito son:

- `private`.
- `public`.
- `protected`.
- **El ámbito por defecto**.

4.3.2.1 Atributos `private`.

El modificador de ámbito `private` es el más restrictivo de todos. Todo atributo `private` es visible únicamente dentro de la clase en la que se declara. No existe ninguna forma de acceder al mismo si no es a través de algún método (no `private`) que devuelva o modifique su valor.

Una buena metodología de diseño de clases es declarar los atributos `private` siempre que sea posible, ya que esto evita que algún objeto pueda modificar su valor si no es a través de alguno de sus métodos diseñados para ello.

4.3.2.2 Atributos `public`.

El modificador de ámbito `public` es el menos restrictivo de todos. Un atributo `public` será visible en cualquier clase que desee acceder a él, simplemente anteponiendo el nombre de la clase. Las aplicaciones bien diseñadas minimizan el uso de los atributos `public` y maximizan el uso de atributos `private`. La forma apropiada de acceder y modificar atributos de objetos es a través de métodos que accedan a los mismos, aunque en ocasiones,

para acelerar el proceso de programación, se declaran de tipo `public` y se modifican sus valores desde otras clases.

```
final class Empleado {
        public String nombre;
        public String dirección;
        private int sueldo;
}
```

En este ejemplo existen dos atributos `public` (nombre y dirección) y uno `private` (sueldo). Los atributos nombre y dirección podrán ser modificados por cualquier clase, por ejemplo de la siguiente forma:

```
emple1.nombre="Pedro López";
```

Mientras que el atributo sueldo no puede ser modificado directamente por ninguna clase que no sea `Empleado`. En realidad, para que la clase estuviera bien diseñada, se deberían haber declarado `private` los tres atributos y declarar métodos para modificar los atributos. De estos métodos, el que modifica el atributo sueldo podría declararse de tipo `private` para que no pudiera ser utilizado por otra clase distinta de `Empleado`.

4.3.2.3 Atributos `protected`.

Los atributos `protected` pueden ser accedidos por las clases del mismo paquete (*package*) y por las subclases del mismo paquete, pero no pueden ser accedidas por subclases de otro paquete, aunque sí pueden ser accedidas las variables `protected` heredadas de la primera clase.

Esto parece un poco confuso, veámoslo con un ejemplo:

```
package PProtegido;
        public class Protegida {
                protected int valorProtegido;
                public Protegida(int v) {
                        valorProtegido=v;
                }
        }

package PProtegido;
        public class Protegida2 {
                public static void main(String argumentos[]) {
                        Protegida p1= new Protegida(0);
                        p1.valorProtegido = 4;
                        System.out.println(p1.valorProtegido);
            }
        }
```

En este caso, por pertenecer la clase `Protegida2` al mismo paquete que la clase `Protegida`, se puede acceder a sus atributos `protected`. En nuestro caso, `valorProtegido`.

```
package OtroPaquete;
```

```
import PProtegido.*;
        public class Protegida3 {
        public static void main(String argumentos[]) {
                Protegida p1= new Protegida(0);
                p1.valorProtegido = 4;
                System.out.println(p1.valorProtegido);
         }
}
```

En este caso, se importa el paquete `Pprotegido` para poder acceder a la clase `Protegida`, pero el paquete en el que se declara la clase `Protegida3` es `OtroPaquete` distinto al que contiene `Protegida`, por lo que no se puede acceder a sus atributos `protected`. El compilador Java mostraría un error al acceder a `valorProtegido` en la línea `p1.valorProtegido = 4;` *(puesto que intenta acceder, para modificar, un atributo protegido)* y en la línea `System.out.println(p1.valorProtegido);` *(por el mismo motivo, a pesar de que se trate sólo de leer el valor).*

Sin embargo:

```
package OtroPaquete;
import PProtegido.*;
        public class Protegida4 extends Protegida {
                public Protegida4(int v) {
                super(v);
        }
        public void modifica(int v) {
                valorProtegido=v;
        }
}
```

En este caso, se ha declarado una subclase de `Protegida`. Esta clase puede acceder a sus atributos (incluso a `valorProtegido`), por ejemplo, a través del método `modifica`, pero:

```
package OtroPaquete;
        public class EjecutaProtegida4 {
                public static void main(String argumentos[]) {
                        Protegida4 p1= new Protegida4(0);
                        p1.valorProtegido = 4;
                        System.out.println(p1.valorProtegido);
                }
        }
```

En este caso, a pesar de que *(el objeto)* la variable es del tipo `Protegida4` (subclase de Protegida), y la clase `EjecutaProtegida4` pertenece al mismo paquete que `Protegida4`, no se puede acceder a los atributos `private`. Sólo los métodos de la clase `Protegida4` pueden hacerlo. Así:

```
        package OtroPaquete;
                public class EjecutaProtegida4_2 {
                public static void main(String argumentos[]) {
```

```
                Protegida4 p1= new Protegida4(0);
                p1.modifica(4);
        }
}
```

Esta clase sí que puede modificar el atributo `protected` pero únicamente a través del método de la clase `Ejecuta4` denominado `modifica`.

En resumen: Un atributo protegido sólo puede ser modificado por clases del mismo paquete, ahora bien, si se declara una subclase entonces esa subclase es la encargada de proporcionar los medios para acceder al atributo protegido.

4.3.2.4 El ámbito por defecto de los atributos.

Los atributos que no llevan ningún modificador de ámbito pueden ser accedidos desde las clases del mismo paquete, pero no desde otros paquetes.

4.3.2.5 Resumen de ámbitos de atributos.

	Acceso desde			
Modificador	Misma Clase	SubClase	MismoPaquete	TodoelMundo
Private	Si	No	No	No
Public	Si	Si	Si	Si
Protected	Si	Según	Si	No
Por defecto	Si	No	Si	No

4.4 Métodos

Sintaxis general de los métodos:

```
Declaración de método {
        Cuerpo del método
}
```

4.4.1 Declaración de método.

La declaración mínima sin modificadores de un método es:

```
TipoDevuelto NombreMétodo (ListaParámetros)
```

Donde:

- *TipoDevuelto* es el tipo de dato devuelto por el método (función). Si el método no devuelve ningún valor, en su lugar se indica la palabra reservada `void`. Por ejemplo: `void noDevuelveNada.`
- *NombreMétodo* es un identificador válido en Java.

☞ • *ListaParámetros* si tiene parámetros, es una sucesión de pares **tipo - valor** separados por comas. *Ejemplo: int mayor(int x , int y)*. Los parámetros pueden ser también objetos. **Los tipos simples de datos se pasan siempre por valor y los objetos y vectores por referencia**.

Cuando se declara una subclase, esa subclase hereda, en principio, todos los atributos y métodos de la superclase (clase padre). Estos métodos pueden ser redefinidos en la clase hija simplemente declarando métodos con los mismos identificadores, parámetros y tipo devuelto que los de la superclase. Si desde uno de estos métodos redefinidos se desea realizar una llamada al método de la superclase, se utiliza el identificador de la superclase y se le pasan los parámetros.

Ejemplo: suponiendo que se ha declarado una clase como heredera de otra (SuperC) en la que existe el método int mayor(int x, int y), se puede redefinir este método simplemente declarando un método con el mismo identificativo y parámetros int mayor(int x , int y). Si desde dentro del método redefinido se desea hacer referencia al método original se podría utilizar: var = SuperC(x,y); También se pueden declarar métodos para una misma clase con los mismos identificadores pero con parámetros distintos.

```
class Mayor {
// Declara dos métodos con el mismo identificativo
// uno de ellos acepta dos enteros y el otro dos
// enteros largos.
// ambos métodos devuelven el valor mayor de los
// dos enteros que se pasan como parámetros.
        static int mayor(int x, int y) {
                if (x>y)
                        return x;
                else
                        return y;
        }
        static long mayor(long x, long y) {
                if (x>y)
                        return x;
                else
                        return y;
        }
}
class ConstantesMayor {
static final int INT = 15;
static final long LONG = 15;
}
class SubMayor extends Mayor {
// modifica la clase Mayor de la siguiente forma:
// los métodos devuelven el valor mayor de entre
// los dos parámetros que se le pasan, pero
// siempre, como mínimo devuelve el valor
// de las constantes INT y LONG

        static int mayor(int x, int y) {
```

```
// llama al método mayor de la superclase
        int m = Mayor.mayor(x,y);
                return Mayor.mayor(m,ConstantesMayor.INT);
        }

        static long mayor(long x, long y) {
        // llama al método mayor de la superclase
                long m = Mayor.mayor(x,y);
                return Mayor.mayor(m,ConstantesMayor.LONG);
        }
}
class EjecutaMayor {
public static void main(String argumentos[]) {
        int int1=12,int2=14;
        long long1=20, long2=10;
        System.out.println("ENTEROS:");
        System.out.println("mayor de 12 y 14 = " +Mayor.mayor(int1,int2));
        System.out.println("mayor de 12 y 14 y 15 ="+SubMayor.mayor(int1,int2));
        System.out.println("ENTEROS LARGOS:");
        System.out.println("mayor de 20 y 10 = " +Mayor.mayor(long1,long2));
        System.out.println("mayor de 20 y 10 y 15 ="+SubMayor.mayor(long1,long2));
        }
}
```

Declaración completa de métodos.

Sintaxis general:

> [**ModificadorDeÁmbito**] [**static**] [**abstract**] [**final**] [**native**] [**synchronized**]
> **TipoDevuelto NombreMétodo (** [**ListaParámetros**] **)** [**throws ListaExcepciones**]

Devolución De Valores

Los métodos pueden devolver valores básicos (int, short, double, etc.), Strings, arrays e incluso objetos.

En todos los casos es el comando **return** el que realiza esta labor. En el caso de arrays y objetos, devuelve una referencia a ese array u objeto. Ejemplo:

```
class FabricaArrays {
public int[] obtenArray(){
        int array[]= {1,2,3,4,5};
        return array;
   }
}
public class returnArray {
        public static void main(String[] args) {
                FabricaArrays fab=new FabricaArrays();
                int nuevoArray[]=fab.obtenArray();
        }
}
```

SobreCarga de Métodos

Una propiedad de la POO es el polimorfismo. Java posee esa propiedad ya que admite sobrecargar los métodos. Esto significa crear distintas variantes del mismo método.

Ejemplo:

```
class Matemáticas{
        public double suma(double x, double y) {
        return x+y;
}
public double suma(double x, double y, double z){
        return x+y+z;
}
public double suma(double[] array){
        double total =0;
        for(int i=0; i<array.length;i++){
                total+=array[i];
        }
        return total;
}
```

La clase matemáticas posee tres versiones del método suma. una versión que suma dos números double, otra que suma tres y la última que suma todos los miembros de un array de doubles. Desde el código se puede utilizar cualquiera de las tres versiones según convenga.

4.4.1.1 Métodos `static`.

Los métodos `static` son métodos de clase (no de objeto) y por tanto, no necesita instanciarse la clase (crear un objeto de esa clase) para poder llamar a ese método. Se ha estado utilizando hasta ahora siempre que se declaraba una clase ejecutable, ya que para poder ejecutar el método `main()` no se declara ningún objeto de esa clase.

Los métodos de clase (`static`) únicamente pueden acceder a sus atributos de clase (`static`) y nunca a los atributos de objeto (no `static`). Ejemplo:

```
class EnteroX {
        int x;
        static int x() {
                return x;
        }
        static void setX(int nuevaX) {
                x = nuevaX;
        }
}
```

Mostraría el siguiente mensaje de error por parte del compilador:

```
MetodoStatic1.java:4: Can't make a static reference to
nonstatic variable x in class EnteroX.
return x;
^
MetodoStatic1.java:7: Can't make a static reference to
nonstatic variable x in class EnteroX.
x = nuevaX;
^
2
```

Sí que sería correcto:

```
class EnteroX {
```

```
        static int x;
                static int x() {
                        return x;
                }
          static void setX(int nuevaX) {
                x = nuevaX;
          }
        }
```

Al ser los métodos `static`, puede accederse a ellos sin tener que crear un objeto `EnteroX`:

```
        class AccedeMetodoStatic {
        public static void main(String argumentos[]) {
                EnteroX.setX(4);
                System.out.println(EnteroX.x());
          }
        }
```

4.4.1.2 Métodos `abstract`.

Los métodos abstract se declaran en las clases abstract. Es decir, si se declara algún método de tipo abstract, entonces, la clase debe declararse obligatoriamente como abstract.

Cuando se declara un método abstract, no se implementa el cuerpo del método, sólo su signatura. Las clases que se declaran como subclases de una clase abstract deben implementar los métodos abstract. Una clase abstract no puede ser instanciada, únicamente sirve para ser utilizada como superclase de otras clases.

4.4.1.3 Métodos `final`.

Los métodos de una clase que se declaran de tipo final no pueden ser redefinidos por las subclases. Esta opción puede adoptarse por razones de seguridad, para que nuestras clases no puedan ser extendidas por otros.

```
abstract class Animal {
        String nombre;
        int patas;
        public Animal(String n, int p) {
                nombre=n;
                patas=p;
        }
        public final int numPatas(){
                return patas;
        }
        abstract void habla();
}


class Perro extends Animal {
        String raza;
        public Perro(String n, int p, String r) {
                        super(n,p);
                raza=r;
```

```
        }
        public void habla() {
                System.out.println("Me llamo "+nombre+": GUAU, GUAU");
                System.out.println("mi raza es "+raza);
                System.out.println("Tengo "+numPatas()+" patas.");
        }
}

class Gallo extends Animal {
        public Gallo(String n, int p) {
                super(n,p);
        }
        public void habla() {
                System.out.println("Soy un Gallo, Me llamo "+nombre);
                System.out.println("Kikirikiiii");
        }
}

class FinalAbstracta {
        public static void main(String argumentos[]) {
                Perro toby = new Perro("Toby",4,"San Bernardo");
                Gallo kiko = new Gallo("Kiko",2);
                kiko.habla();
                System.out.println();
                toby.habla();
        }
}
```

En este caso, la clase Animal es abstracta (no puede instanciarse), sólo puede utilizarse como superclase de otras (Perro y Gallo). Uno de los métodos de Animal es final y por lo tanto no puede redefinirse. Cualquier intento de declarar un método (numPatas) en cualquier subclase de Animal generaría un error del compilador.

En el siguiente caso tenemos otro caso donde hemos definido una clase abstracta.

```
abstract class vehiculo {
        public int velocidad=0;
        abstract public void acelera();
        public void para() {velocidad=0;}
}
class coche extends vehiculo {
        public void acelera() {
          velocidad+=5;
        }
}
public class prueba {
        public static void main(String[] args) {
                coche c1=new coche();
                c1.acelera();
                System.out.println(c1.velocidad);
                c1.para();
                System.out.println(c1.velocidad);
        }
}
```

4.4.1.4 Modificadores de ámbito de los métodos.

Los modificadores de ámbito de los métodos son exactamente iguales que los de los atributos, especifican la forma en que puede accederse a los mismos desde otras clases. Estos modificadores de ámbito son:

- ☞ • `private`.
- ☞ • `public`.
- ☞ • `protected`.
- ☞ • **El ámbito por defecto**.

	Acceso desde			
Modificador	Misma Clase	SubClase	MismoPaquete	TodoelMundo
Private	Si	No	No	No
Public	Si	Si	Si	Si
Protected	Si	Según	Si	No
Por defecto	Si	No	Si	No

4.5 Constructores.

Un constructor es un método especial de las clases que sirve para inicializar los objetos que se instancian como miembros de una clase.

Para declarar un constructor basta con declarar un método con el mismo nombre que la clase. **No se declara el tipo devuelto por el constructor** (ni siquiera `void`), aunque sí que se pueden utilizar los modificadores de ámbito de los métodos: `public`, `protected`, `private`.

Los constructores tienen el mismo nombre que la clase y todas las clases tienen uno por defecto (que no es necesario declarar), aunque es posible sobrescribirlo e incluso declarar distintos constructores (sobrecarga de métodos) al igual que los demás métodos de una clase.

```
class Nif {
int dni;
char letra;
static char tabla[]={'T','R','W','A','G','M','Y','F','P',
'D','X','B','N','J','Z','S','Q','V',
'H','L','C','K','E'};

public Nif(int ndni,char nletra) throws NifException{
        if (Character.toUpperCase(nletra)==tabla[ndni%23]) {
                dni=ndni;
                letra=Character.toUpperCase(nletra);
        }
```

```
            else
                throw new LetraNifException("Letra de NIF incorrecta");
}

public Nif(int ndni) {
            dni=ndni;
            letra=tabla[dni%23];
}

public Nif(String sNif) throws NifException, LetraNifException {
char letraAux;
StringBuffer sNumeros= new StringBuffer();
int i,ndni;
            for (i=0;i<sNif.length();i++) {
                        if ("1234567890".indexOf(sNif.charAt(i))!=-1) {
                                    sNumeros.append(sNif.charAt(i));
                        }
            }
try {
       dni=Integer.parseInt(sNumeros.toString());
       letraAux=Character.toUpperCase(sNif.charAt(
       sNif.length()-1));
 } catch (Exception ex) {
        throw new NifException("NIF incorrecto");
  }
letra=tabla[dni%23];
     if ("ABCDEFGHIJKLMNOPQRSTUVWXYZ".indexOf(letraAux)!=-1) {
        if (letraAux!=letra) {
         throw new LetraNifException("Letra de NIF incorrecta");
        }
     } else letra=tabla[dni%23];
}

public char obtenerLetra() {
    return letra;
}
public int obtenerDni() {
    return dni;
}
public String toString() {
   return (String.valueOf(dni)+String.valueOf(letra));
}

public String toStringConFormato() {
  String sAux= String.valueOf(dni);
  StringBuffer s = new StringBuffer();
  int i;

            for (i=sAux.length()-1;i>2;i-=3) {
                s.insert(0,sAux.substring(i-2,i+1));
                s.insert(0,".");
            }

            s.insert(0,sAux.substring(0,i+1));
            s.append('-');
            s.append(letra);
            return (s.toString());
}
```

```
static char letraNif(int ndni) {
        return tabla[ndni%23];
}
static char letraNif(String sDni) throws NifException {
        Nif j = new Nif(sDni);
        return j.obtenerLetra();
 }
}
class NifException extends Exception {
  public NifException() { super(); }
  public NifException(String s) { super(s); }
}

class LetraNifException extends NifException {
        public LetraNifException() { super(); }
        public LetraNifException(String s) { super(s); }
}
```

En el ejemplo anterior, la clase `Nif` tiene tres constructores:

- public **Nif**(int ndni,char nletra) throws LetraNifException
- public **Nif**(int ndni)
- public **Nif**(String sNif) throws NifException, LetraNifException

Para inicializar un objeto de una determinada clase se llama a su constructor después de la palabra reservada `new`**.**

```
class EjecutaNif {
public static void main(String argumentos[]) {
        Nif n;
        int dni;
        if (argumentos.length!=1) {
                System.out.println("Uso: EjecutaNif dni");
                return;
        }
        else {
                dni = Integer.valueOf(argumentos[0]).intValue();
                n = new Nif(dni);
                System.out.println("Nif: "+n.toStringConFormato());
        }
   }
}
```

En este ejemplo, se está llamando al segundo constructor, aquel que acepta como parámetro un entero.

En él se acepta como argumento en la línea de comandos un DNI y muestra el NIF correspondiente:

```
java EjecutaNif 18957690
```

mostraría la siguiente salida por pantalla:

```
Nif: 18.957.690-D
```

Es bastante habitual cuando se sobrescribe el constructor o constructores de la superclase el realizar una llamada al constructor de la superclase y después realizar otras operaciones de inicialización de la clase hija. Esto, como ya se ha visto en el punto anterior, se realiza utilizando el identificador de clase `super`. Así, si se declarara una subclase de `Nif`, y se sobrescribiera alguno de sus constructores, se podría realizar en primer lugar, una llamada al constructor de `Nif`.

```
class HijaNif extends Nif {
public static numNifs = 0;
…
        public Nif(int ndni) {
        super.Nif(ndni);
        numNifs++;
        }
…
}
```

En este caso se ha declarado una clase `HijaNif` que añade un atributo de clase[27] que sirve de contador del número de objetos que se instancian de la clase `HijaNif`. Se rescribe el constructor (constructores) de forma que se incrementa este contador por cada objeto declarado.

Tenemos otro ejemplo donde definimos un constructor de la clase Ficha

```
class Ficha {
private int casilla;
        Ficha() { //constructor
                casilla = 1;
        }
        public void avanzar(int n) {
                casilla += n;
        }
        public int casillaActual(){
                return casilla;
        }
}
public class app {
        public static void main(String[] args) {
                Ficha ficha1 = new Ficha();
                ficha1.avanzar(3);
                System.out.println(ficha1.casillaActual());//Da 4
      }
}
```

En la línea *Ficha ficha1 = new Ficha();* es cuando se llama al constructor, que es el que coloca inicialmente la casilla a 1. Pero el constructor puede tener parámetros como en el siguiente caso:

```
class Ficha {
private int casilla; //Valor inicial de la propiedad
        Ficha(int n) { //constructor
                casilla = n;
```

```
        }
        public void avanzar(int n) {
                casilla += n;
        }
        public int casillaActual(){
                return casilla;
        }
}

public class app {
public static void main(String[] args) {
        Ficha ficha1 = new Ficha(6);
        ficha1.avanzar(3);
        System.out.println(ficha1.casillaActual());//Da 9
   }
}
```

4.6 Destructores.

Un destructor es un método de la clase que sirve para realizar una serie de operaciones cuando un objeto perteneciente a la clase deja de existir. Operaciones típicas en los objetos cuando desaparecen son la liberación de recursos del sistema que tuviera asignados el objeto: liberación de memoria que pudiera tener reservada el objeto, cierre de los ficheros y sockets que tuviera abiertos, etc..

En Java existe un *Thread* del sistema "*Garbage collector*" literalmente: Recolector de Basura, que se ejecuta regularmente para liberar la memoria asignada a objetos que ya no se necesitan. A pesar de ello, puede ser necesario realizar algunas operaciones adicionales. Para ello hay que declarar un método de la siguiente forma:

protected void finalize() throws throwable

Por ejemplo, en la clase `HijaNif`, se utiliza un contador para saber el número de objetos instanciados de la clase `HijaNif`. Para decrementar `numNifs`, habría que declarar el método `finalize()`:

```
protected void finalize() throws throwable {
        numNifs--;
        super.finalize();
}
```

Es conveniente llamar al método `super.finalize()`, el destructor de la superclase, para liberar recursos que pudiera tener asignados la clase heredados transparentemente de la clase padre y de los cuales no se tuviera conocimiento.

4.7 INTERFACES.

La limitación de que sólo se puede heredar de una clase, hace que haya problemas ya que muchas veces se deseará heredar de varias clases. Aunque ésta no es la finalidad directa de las interfaces, sí que tiene cierta relación.

Mediante interfaces se definen una serie de comportamientos de objeto. Estos comportamientos puede ser "implementados" en una determinada clase. No definen el tipo de objeto que es, sino lo que puede hacer (sus capacidades). Por ello lo normal es que el nombre de las interfaces terminen con el texto "**able"** (*configurable, modificable, cargable*).

Por ejemplo en el caso de la clase Coche, esta deriva de la superclase Vehículo, pero además puesto que es un vehículo a motor, puede implementar métodos de una interfaz llamada por ejemplo **arrancable.** Se dirá entonces que la clase Coche es *arrancable*. utilizar interfaces.

Para hacer que una clase utilice una interfaz, se añade detrás del nombre de la clase la palabra **implements** seguida del nombre del interfaz. Se pueden poner varios nombres de interfaces separados por comas (solucionando, en cierto modo, el problema de la herencia múltiple).

Definimos la interfaz de pila

```
public interface Stack {
     public int size();
     public boolean isEmpty();
     public void push(Object o);
     public Object pop() throws StackEmptyException;
     public Object top()  throws StackEmptyException;
  }
```

Tenemos una clase para el manejo de las excepciones.

```
public class StackEmptyException extends Exception{
    public StackEmptyException(){
      super();
    }
    public StackEmptyException(String mensaje){
      super(mensaje);
    }
  }
```

Realizamos la implementación de la pila utilizando la interfaz definida anteriormente utilizando listas.

```
public class LinkedStack implements Stack
  {
     private Node top;
     private int size;

     public int size() {
       return size;
```

```
    }
    public boolean isEmpty() {
      return (top==null);
    }
    public void push(Object e) {
      Node n=new Node();
      n.setElem(e);
      n.setNext(top);
      top=n;
      size++;
    }
    public Object top()
    throws StackEmptyException{
      if (isEmpty())
        throw new StackEmptyException("vacia");
      return top.getElem();
    }

    public Object pop()
    throws StackEmptyException{
      Object temp;
      if (isEmpty())
        throw new StackEmptyException("vacia");
      temp=top.getElem();
      top=top.getNext();
      size--;
      return temp;
    }
  }
```

Para el mismo caso de pilas realizamos la implementación de la interfaz utilizando la clase vector.

```
import java.util.Vector;
  public class VectorStack implements Stack{
    private Object top;
    private Vector pila;

    public VectorStack(){
      pila = new Vector();
    }

    public int size(){
      return pila.size();
    }
    public boolean isEmpty(){
      return (pila.size() == 0);
    }
    public void push(Object o){
      pila.addElement(o);
    }
    public Object pop() throws StackEmptyException{
      Object o;

      if (pila.size() == 0) throw new StackEmptyException();
      o = pila.remove(pila.size() - 1);
      return o;
```

```
    }
    public Object top() throws StackEmptyException{
      if (pila.size() == 0)
        throw new StackEmptyException();
      else
        return pila.lastElement();
    }
}
```

Bibliografía

- Descubre Java 1.2, Morgan Mike , 1999 , Editorial Prentice Hall.
- Aplicaciones de Pascal en Ciencias , Richard E. Crandall, Serie Instrucción Programada Editorial Limusa.
- Estructura de Datos en C, Aaron M. Tenenbaum- Yedidyah Langam, editorial Prentice Hall 1993.
- Algoritmos y su codificación C++, César Liza Avila , Grupo Creadores (*Motivando tu naturaleza Creatividad*).
- Estructura de datos, Osvaldo Cairó-Silvia Guardati , McGrawHill Tercera Edición 2006.
- Programación Estructurada en C. Problemas resueltos , Walter Lazo Aguirre.
- Pensando en java, Bruce Eckel, President, Prentice Hall, segunda edición 2005
- Java 2 , Steven Holzner, Anaya Multimedia 2005 Segunda Edición 2004.
- Java 2 , Francisco Javier Cevallos, editorial AlfaOmega 2004.
- Java 2, Jorge Sánchez año 2004