PROGRAMACIÓN

2ª TUTORÍA COLECTIVA

PROGRAMACIÓN MODULAR

* El resultado final de la programación modular es la **división** de un programa en un conjunto de módulos independientes que se comunican entre sí a través de llamadas.

* Un módulo representa **una tarea** determinada, identificada por un nombre (nombre del módulo) por el cual será invocada desde otros módulos.

* Siempre debe haber un módulo que describa la solución completa y que servirá de nexo de unión entre los demás, es el **programa principal**.

* Los módulos se **comunican** mediante variables de enlace a las que se denomina **parámetros** o argumentos.

* Los módulos se llaman funciones si devuelven algún argumento y procedimientos si no devuelven ningún valor. En Java hablamos de métodos.

VENTAJAS DE LA PROGRAMACIÓN MODULAR

* Los módulos se pueden desarrollar de manera independiente y paralélela por diferentes equipos, sin conocer casi unos de otros. Favorece el trabajo en equipo.
* Menor tiempo para la desarrollo de una aplicación.
* Los módulos, al ser independientes, son fácilmente modificables y/o sustituibles siempre y cuando se mantenga su interfaz con el resto, luego facilita la depuración y las pruebas, al poder hacerlo por separado cada módulo, y por tanto también facilita el mantenimiento.
* El cambio en uno de los módulos no afecta al conjunto de los demás.
* Simplifica el diseño. Disminuye la complejidad de los algoritmos y facilita su comprensión.
* Disminuye el tamaño total del programa.
* Ahorra en tiempo de **programación** porque promueve la reusabilidad del código.
* Al permitir la reutilización del código, disminuyen los costes.

PROGRAMACIÓN MODULAR con PSeInt

num es el parámetro

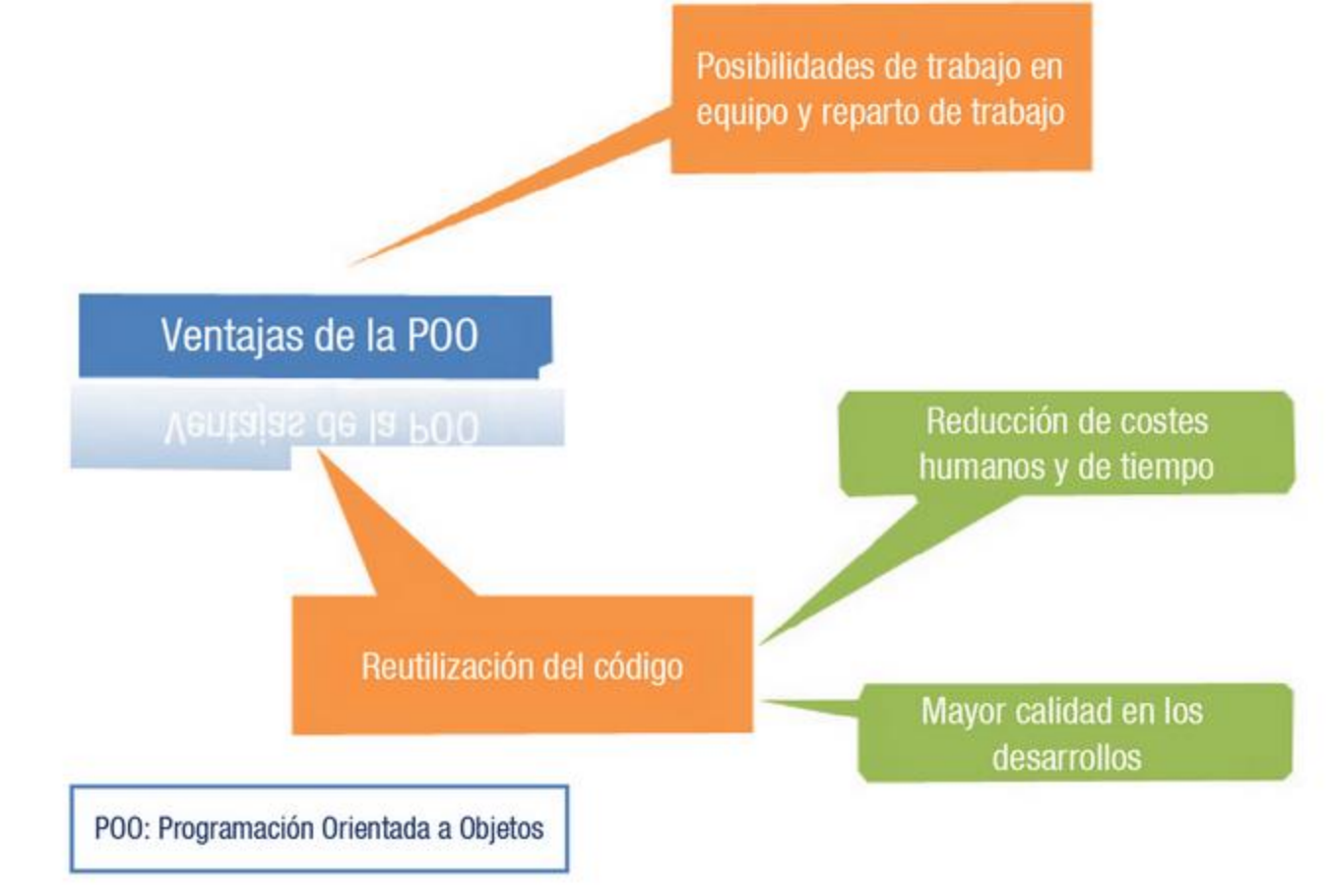
res1 es el valor que devuelve

```
1  // funcion que no recibe ni devuelve nada
2  SubProceso Saludar()
3    Escribir "Hola mundo!"
4  FinSubProceso
5
6  // funcion que recibe un parámetro y devuelve su doble
7  SubProceso res1 <- CalcularDoble(num)
8    res1 <- num*2 // retorna el doble
9  FinSubProceso
10
11 // funcion que recibe un parámetro y devuelve su triple
12 SubProceso res2 <-Triplicar(num)
13   res2 <- num*3
14 FinSubProceso
15
16 // proceso principal, que invoca a las funciones antes declaradas
17 Proceso PruebaFunciones
18   Escribir "Llamada a la funcion Saludar:"
19   Saludar() // como no recibe argumentos pueden omitirse los paréntesis vacios
20   Escribir "Introduce un valor numérico para x:"
21   Leer x
22   Escribir "Llamada a la función CalcularDoble"
23   Escribir "El doble de ",x," es ", CalcularDoble(x)
24   Escribir "Llamada a la función Triplicar"
25   Escribir "El tripleY de ",x," es ", Triplicar(x)
26 FinProceso
```

CARACTERÍSTICAS DE JAVA

* El código generado por el compilador Java es **independiente** de la arquitectura.
* Está totalmente **orientado a objetos**.
* Su sintaxis es **similar** a C y C++.
* Es distribuido, preparado para **aplicaciones TCP/IP.**
* Dispone de un amplio conjunto de **bibliotecas**.
* Es **robusto**, realizando comprobaciones del código en tiempo de compilación y de ejecución.
* La **seguridad** está garantizada, ya que las aplicaciones Java no acceden a zonas delicadas de memoria o del sistema.

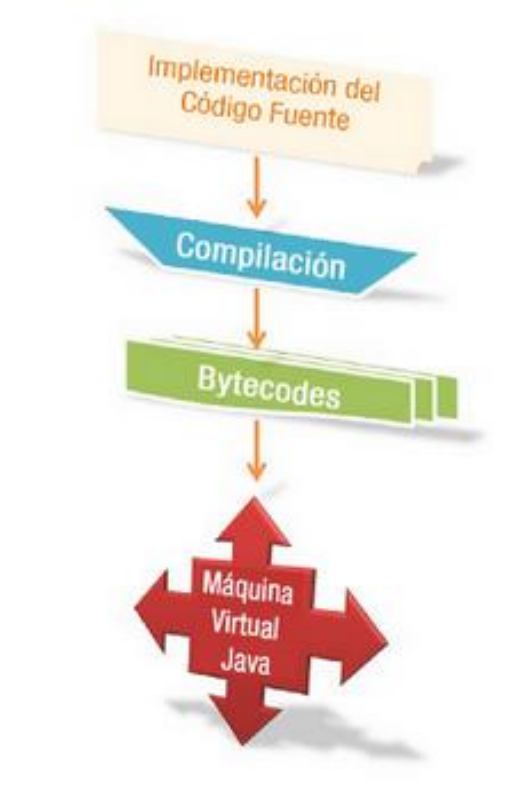
VENTAJAS DE LA POO



JAVA Y LOS BYTECODES

* Un programa escrito en Java no es directamente ejecutable, es necesario que el código fuente sea interpretado por la Maquina Virtual Java. (JVM)

* Una vez escrito el código fuente (archivos con extensión .Java), éste es precompilado generándose los códigos de bytes, Bytecodes (archivos con extensión .class) que serán interpretados directamente por la Maquina Virtual Java y traducidos a código nativo de la plataforma sobre la que se esté ejecutando el programa.

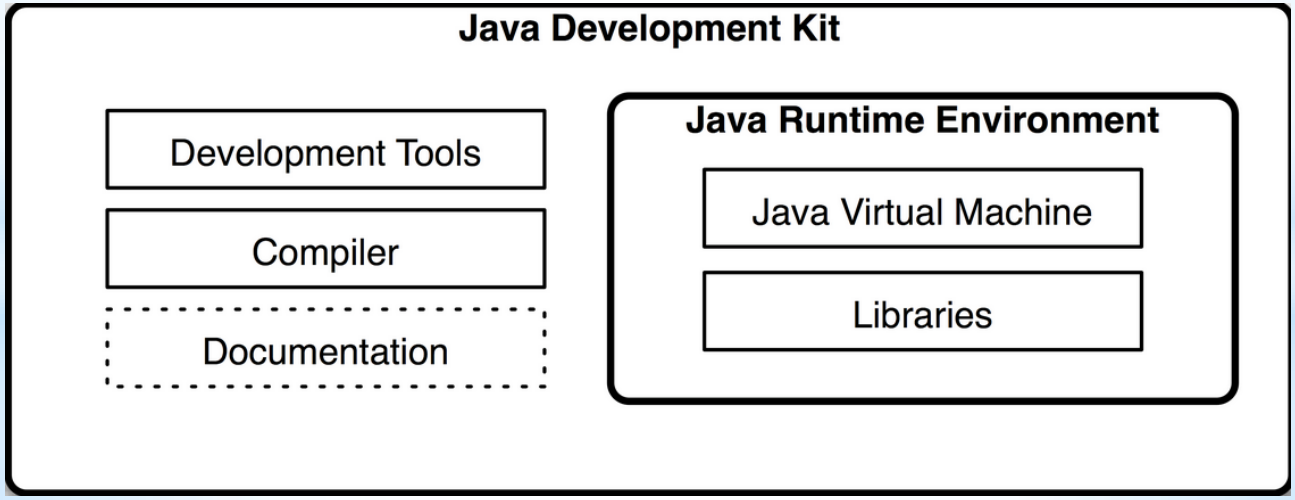


EL ENTORNO DE DESARROLLO JAVA

* La herramienta básica para empezar a desarrollar aplicaciones en Java es el **JDK** (Java Development Kit o Kit de Desarrollo Java), que incluye un compilador y un intérprete para línea de comandos.

* Así mismo, junto a JDK se incluye una implementación del entorno de ejecución Java, el **JRE** (Java Runtime Environment) para ser utilizado por el JDK. El JRE incluye la Máquina Virtual de Java (JVM - Java Virtual Machine) y la biblioteca de clases, la **API** (Aplication Programming Interface).

* La API incluye paquetes de clases útiles para la realización de múltiples tareas habituales.



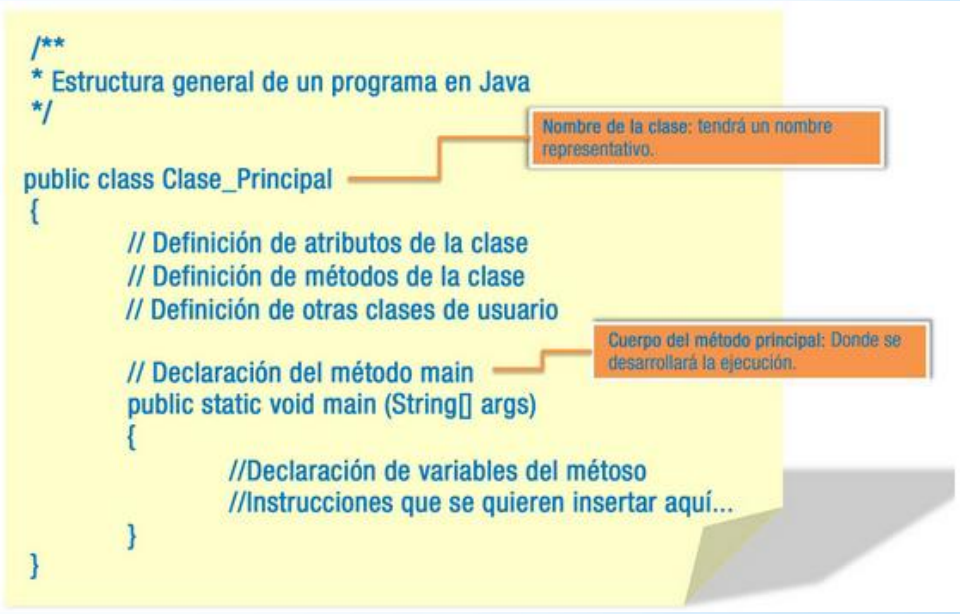
TIPOS DE APLICACIONES JAVA

* **Aplicaciones de consola:**
  * Son programas independientes al igual que los creados con los lenguajes tradicionales.
  * Se componen como mínimo de un archivo .class que debe contar necesariamente con el método main.
  * No necesitan un navegador web y se ejecutan cuando invocamos el comando Java para iniciar la Máquina Virtual de Java (JVM). De no encontrarse el método main la aplicación no podrá ejecutarse.
  * Las aplicaciones de consola leen y escriben hacia y desde la entrada y salida estándar, sin ninguna interfaz gráfica de usuario.
* **Aplicaciones gráficas:**
  * Aquellas que utilizan las clases con capacidades gráficas, como Swing que es la biblioteca para la interfaz gráfica de usuario avanzada de la plataforma Java SE.
  * Incluyen las instrucciones import, que indican al compilador de Java que las clases del paquete Javax.swing se incluyan en la compilación.
* **Applets:**
  * Son programas incrustados en otras aplicaciones, normalmente una página web que se muestra en un navegador. Cuando el navegador carga una web que contiene un applet, éste se descarga en el navegador web y comienza a ejecutarse. Esto nos permite crear programas que cualquier usuario puede ejecutar con tan solo cargar la página web en su navegador.
* **Servlets:**
  * Son componentes de la parte del servidor de Java EE, encargados de generar respuestas a las peticiones recibidas de los clientes.
  * Los servlets, al contrario de los applets, son programas que están pensados para trabajar en el lado del servidor y desarrollar aplicaciones Web que interactúen con los clientes.
* **Midlets:**
  * Son aplicaciones creadas en Java para su ejecución en sistemas de propósito simple o dispositivos móviles.
  * Son programas creados para dispositivos embebidos (se dedican a una sola actividad), más específicamente para la máquina virtual Java MicroEdition (Java ME).
  * Generalmente son juegos y aplicaciones que se ejecutan en teléfonos móviles.

ESTRUCTURA DE UN PROGRAMA

* public class Clase_Principal:
  * Es una clase general en la que se incluyen todos los demás elementos del programa.
  * Entre otras cosas, contiene el método o función main() que representa al programa principal, desde el que se llevará a cabo la ejecución del programa.
  * El nombre del fichero .java que contiene el código fuente de nuestro programa, coincidirá con el nombre de esta clase.

* public static void main (String[] args):
  * Es el método que representa al programa principal.
  * En él se podrán incluir las instrucciones que estimemos oportunas para la ejecución del programa.
  * Desde él se podrá hacer uso del resto de clases creadas.
  * Todos los programas Java tienen un método main.



IDENTIFICADORES

* **Java distingue las mayúsculas de las minúsculas**. Por ejemplo, Alumno y alumno son variables diferentes.
* **No se suelen utilizar identificadores que comiencen con «$» o «_»,** además el símbolo del dólar, por convenio, no se utiliza nunca.
* **No se puede utilizar el valor booleano** (true o false) **ni el valor nulo** (null).
* Los **identificadores deben ser lo más descriptivos posibles**. Es mejor usar palabras completas en vez de abreviaturas crípticas.
* **No podemos usar** como identificadores las **palabras reservadas** del lenguaje.

Convenciones sobre identificadores en Java		
Identificador	Convención	Ejemplo
Nombre de variable	Comienza por letra minúscula, y si tienen más de una palabra se colocan juntas y el resto comenzando por mayúsculas	numAlumnos suma
Nombre de constante	En letras mayúsculas, separando las palabras con el guión bajo, por convenio el guión bajo no se utiliza en ningún otro sitio	TAM_MAX PI
Nombre de una clase	Comienza por letra mayúscula	String MiTipo
Nombre de función	Comienza con letra minúscula	modificaValor obtieneValor

TIPOS DE VARIABLES

TIPOS DE DATOS PRIMITIVOS

Son aquéllos datos sencillos que constituyen los tipos de información más habituales: números, caracteres y valores lógicos o booleanos

Tipo	Descripción	Bytes	Rango
byte	Entero muy corto	1	-128 a 127
short	Entero corto	2	-32,768 a 32,767
int	Entero	4	-2,147,483,648 a 2,147,483,647
long	Entero largo	8	-9,223,372,036,854,775,808 a 9,223,372,036,854,775,807
float	Numero con punto flotante de precisión individual con hasta 7 dígitos significativos	4	+/-1.4E-45 (+/-1.4 times 10-45) a +/-3.4E38 (+/-3.4 times 1038
double	Numero con punto flotante de precisión doble con hasta 16 dígitos significativos	8	+/-4.9E-324 (+/-4.9 times 10-324) a +/-1.7E308 (+/-1.7 times 10308)
char	Carácter UNICODE	2	
boolean	Valor Verdadero o Falso	1	true o false

DECLARACIÓN E INICIALIZACIÓN DE VARIABLES

* Las variables se pueden declarar **en cualquier bloque** de código, dentro de llaves. Y lo hacemos indicando su identificador y el tipo de dato, separadas por comas si vamos a declarar varias a la vez.

```
int numAlumnos = 15;
double radio = 3.14, importe = 102.95;
```

* En ocasiones puede que al declarar una variable no le demos valor. En este caso:
  * Las variables **miembro** se inicializan automáticamente, si no les damos un valor.
    * De tipo numérico, se inicializan por defecto a 0,
    * De tipo carácter o referenciado, se inicializan a null ,
    * De tipo boolean se les asigna el valor por defecto false,
  * Las variables **locales** no se inicializan automáticamente. Debemos asignarles obligatoriamente un valor antes de ser usadas, sino se produce un error.

LITERALES DE TIPOS PRIMITIVOS

Un literal es un valor concreto para un tipo de datos.

* **Booleanos** tienen dos únicos valores que puede aceptar el tipo: true y false.

  encontrado = true;

* **Enteros** se pueden representar en tres notaciones:
  * **Decimal**: por ejemplo **20.** Es la forma más común.
  * **Octal**: por ejemplo **024**. Un número en octal siempre empieza por **0**, seguido de dígitos octales (0 al 7).
  * **Hexadecimal**: por ejemplo **0x14**. Un número en hexadecimal siempre empieza por **0x** seguido de dígitos hexadecimales (del 0 al 9, de la 'a' a la 'f' o de la 'A' a la 'F').

  A los literales de tipo **long** se les debe añadir detrás una **l** ó **L**, por ejemplo **873L**, sino se considera por defecto de tipo int.

* **Reales** o en coma flotante se expresan con punto decimal o en notación científica (E). El valor puede finalizarse con una f o una F para indica el formato float o con una d o una D para indicar el formato double (por defecto es double).

  Por ejemplo, éstas representan el mismo número: **13.2, 13.2D, 1.32e1, 0.132E2**

  Otras constantes literales reales : **.54, 31.21E-5, 2.f, 6.022137e+23f, 3.141e-9d**

* Un **literal carácter** puede escribirse como un carácter entre comillas simples como **'a', 'ñ', 'Z', 'p',** o por su código de la tabla Unicode, anteponiendo la secuencia de escape '****' si el valor lo ponemos en octal o '**\u**' si ponemos el valor en hexadecimal.

  Por ejemplo, la letra **'A'** (mayúscula) es el símbolo número 65. El número 65 en octal es 101 y en hexadecimal 41, podemos representar esta letra como **'\101'** en octal y **'\u0041'** en hexadecimal.

LITERALES DE TIPOS PRIMITIVOS

Existen literales que se representan utilizando secuencias de escape:

Secuencia de escape	Significado	Secuencia de escape	Significado
\b	Retroceso	\r	Retorno de carro
\t	Tabulador	\"	Carácter comillas dobles
\n	Salto de línea	\'	Carácter comillas simples
\f	Salto de página	\\	Barra diagonal

* **Cadenas de caracteres** se indican entre comillas dobles. Al construir una cadena de caracteres se puede incluir cualquier carácter Unicode excepto un carácter de retorno de carro, por ejemplo en la siguiente instrucción utilizamos la secuencia de escape \'' para escribir dobles comillas dentro del mensaje:

```
String  texto = "Juan dijo: \"Hoy hace un día fantástico…\" ";
```

OPERADORES ARITMÉTICOS

Son aquellos que combinados con los operandos forman expresiones matemáticas o aritméticas.

Operadores aritméticos básicos			
Operador	Operación Java	Expresión Java	Resultado
-	Operador unario de cambio de signo	-10	-10
+	Adición	1.2 + 9.3	10.5
-	Sustracción	312.5 - 12.3	300.2
*****	Multiplicación	1.7 * 1.2	1.02
/	División (entera o real)	0.5 / 0.2	2.5
%	Resto de la división entera	25 % 3	1

El resultado de este tipo de expresiones depende de los tipos de los operandos que utilicen:

Resultados de las operaciones aritméticas en Java	
Tipo de los operandos	Resultado
Un operando de tipo long y ninguno real (float o double)	long
Ningún operando de tipo long ni real (float o double)	int
Al menos un operando de tipo double	double
Al menos un operando de tipo float y ninguno double	float

OPERADORES ARITMÉTICOS

Otro tipo de operadores aritméticos son los operadores unarios **incrementales** y **decrementales**.

Producen un resultado del mismo tipo que el operando, y podemos utilizarlos con **notación prefija**, si el operador aparece antes que el operando, o **notación postfija**, si el operador aparece después del operando.

Operadores incrementales en Java		
Tipo operador	Expresión Java	

++ (incremental)	Prefija:
	x=3;
	y=++x;
	// x vale 4 e y vale 4
		Postfija:
		x=3;
		y=x++;
		// x vale 4 e y vale 3

--(decremental)	5-- // el resultado es 4	

EJEMPLO operadores aritméticos

```java
public class operadoresaritmeticos {
    public static void main(String[] args) {
        short x = 7;
        int y = 5;
        float f1 = 13.5f;
        float f2 = 8f;
        System.out.println("El valor de x es " + x + ", y es " + y);
        System.out.println("El resultado de x + y es " + (x + y));
        System.out.println("El resultado de x - y es " + (x - y));
        System.out.printf("%s\n%s%s\n","División entera:","x / y = ",(x/y));
        System.out.println("Resto de la división entera: x % y = " + (x % y));
        System.out.println("El valor de f1 es " + f1 + ", f2 es " + f2);
        System.out.println("El resultado de f1 / f2 es " + (f1 / f2));
    } // fin de main
} // fin de la clase operadoresaritmeticos
```

```
El valor de x es 7, y es 5
El resultado de x + y es 12
El resultado de x - y es 2
División entera:
x / y = 1
Resto de la división entera: x % y = 2
El valor de f1 es 13.5, f2 es 8.0
El resultado de f1 / f2 es 1.6875
```

OPERADORES DE ASIGNACIÓN

* El principal operador de esta categoría es el operador **asignación** "=" , que permite al programa darle un valor a una variable, y ya hemos utilizado varias ocasiones en esta unidad.

* Además de este operador, Java proporciona otros operadores de asignación **combinados** con los operadores aritméticos, que permiten abreviar o reducir ciertas expresiones.

Operadores de asignación combinados en Java		
Operador	Ejemplo en Java	Expresión equivalente
+=	op1 += op2	op1 = op1 + op2
-=	op1 -= op2	op1 = op1 - op2
***=**	op1 *= op2	op1 = op1 * op2
/=	op1 /= op2	op1 = op1 / op2
%=	op1 %= op2	op1 = op1 % op2

OPERADORES RELACIONALES

En la tabla siguiente aparecen los operadores relacionales en Java.

Operadores relacionales en Java		
Operador	Ejemplo en Java	Significado
==	op1 == op2	op1 igual a op2
!=	op1 != op2	op1 distinto de op2
>	op1 > op2	op1 mayor que op2
<	op1 < op2	op1 menor que op2
>=	op1 >= op2	op1 mayor o igual que op2
<=	op1 <= op2	op1 menor o igual que op2

OPERADOR CONDICIONAL ?

Sirve para evaluar una condición y devolver un resultado en función de si es verdadera o falsa dicha condición. Es el único operador ternario de Java, y como tal, necesita tres operandos para formar una expresión.

Operador condicional en Java	
Operador	Expresión en Java
? :	condición ? exp1 : exp2

OPERADORES LÓGICOS

Existen ciertos casos en los que el segundo operando de una expresión lógica no se evalúa para ahorrar tiempo de ejecución, porque con el primero ya es suficiente para saber cuál va a ser el resultado de la expresión.

Operadores lógicos en Java		
Operador	Ejemplo en Java	Significado
!	!op	Devuelve true si el operando es false y viceversa.
&	op1 & op2	Devuelve true si op1 y op2 son true
|	op1 | op2	Devuelve true si op1 u op2 son true
^	op1 ^ op2	Devuelve true si sólo uno de los operandos es true
&&	op1 && op2	Igual que &, pero si op1 es false ya no se evalúa op2
||	op1 || op2	Igual que |, pero si op1 es true ya no se evalúa op2

PROMOCIONES DE TIPOS DE DATOS

* **Conversiones automáticas**. Cuando a una variable de un tipo se le asigna un valor de otro tipo numérico con menos bits para su representación, se realiza una conversión automática.

* **Conversiones explícitas**. Cuando hacemos una conversión de un tipo con más bits a un tipo con menos bits. Este tipo de conversiones se realiza con el **operador cast**. El operador cast es un operador unario que se forma colocando delante del valor a convertir el tipo de dato entre paréntesis. Tiene la misma precedencia que el resto de operadores unarios y se asocia de izquierda a derecha. **(double) x** **(int) y**

Reglas de Promoción de Tipos de Datos:

* Cuando en una expresión hay datos o variables de distinto tipo, el compilador realiza la promoción de unos tipos en otros, para obtener como resultado el tipo final de la expresión.
  * Si uno de los operandos es de tipo **double**, el otro es convertido a **double**.
  * En cualquier otro caso:
    * Si el uno de los operandos es **float**, el otro se convierte a **float**
    * Si uno de los operandos es **long**, el otro se convierte a **long**
    * Si no se cumple ninguna de las condiciones anteriores, entonces ambos operandos son convertidos al tipo **int**.

```
public class Prueba
{
    public static void main(String[] args){
        int a;
        byte b;
        a=12;
        b=14;
        a=b;  //conversión automática en un int introduzco un byte
        System.out.println("a= "+a+" b= "+b);
        // b=a;  ********NO PERMITIDO
        b=(byte)a;   //conversión explícita
        System.out.println("a= "+a+" b= "+b);
    }
}
```

ESTRUCTURAS DE SELECCIÓN: if else

```
public static void main(String[] args){
    //Declaración e inicialización de variables
    int num_aciertos = 12;
    int num_errores = 3;
    int num_preguntas = 20;
    float nota = 0;
    String calificacion="";

    //Procesamos los datos
    nota = ((num_aciertos -(num_errores/2))*10)/num_preguntas;
    if (nota < 5){
        calificacion="INSUFICIENTE";
    }
    else{
        if (nota >=5 && nota <6 )
            calificacion = "SUFICIENTE";
        if (nota >=6 && nota < 7)
            calificacion = "BIEN";
        if (nota >=7 && nota < 9)
            calificacion = "NOTABLE";
        if (nota >=9 && nota <=10)
            calificacion = "SOBRESALIENTE";
    }

    //Salida de información
    System.out.println("La nota obtenida es: " + nota);
    System.out.println("y la calificación obtenida es: " + calificacion);
}
```

ESTRUCTURAS DE SELECCIÓN: switch case

```
public static void main(String[] args){
    //Declaración e inicialización de variables
    int num_aciertos = 12;
    int num_errores = 3;
    int num_preguntas = 20;
    int nota = 0;
    String calificacion="";

    //Procesamos los datos
    nota = ((num_aciertos -(num_errores/2))*10)/num_preguntas;

    switch (nota){
        case 5:  calificacion = "SUFICIENTE";
                 break;
        case 6:  calificacion = "BIEN";
                 break;
        case 7:
        case 8:  calificacion = "NOTABLE";
                 break;
        case 9:
        case 10: calificacion = "SOBRESALIENTE";
                 break;
        default: calificacion = "INSUFICIENTE";
    }

    //Salida de información
    System.out.println("La nota obtenida es: " + nota);
    System.out.println("y la calificación obtenida es: " + calificacion);
}
```

ESTRUCTURAS DE REPETICIÓN: for

Este tipo de bucle tiene las siguientes características:

* Se ejecuta un número determinado de veces.
* Utiliza una variable contadora que controla las iteraciones del bucle.

```
public static void main(String[] args) {
    // Declaración e inicialización de variables
    int numero = 7;
    int contador;
    int resultado=0;

    //Salida de información
    System.out.println ("Tabla de multiplicar del " + numero);
    System.out.println ("............................. ");

    //Utilizamos ahora el bucle for
    for (contador=1; contador<=10;contador++)
    /* La cabecera del bucle incorpora la inicialización de la variable de control, la conc
     * incremento de dicha variable de uno en uno en cada iteración del bucle.
     * En este caso contador++ incrementará en una unidad el valor de dicha variable.
     */
    {
        resultado = contador * numero;
        System.out.println(numero + " x " + contador + " = " + resultado);
        /* A través del operador + aplicado a cadenas de caracteres, concatenamos los valor
         * caracteres que necesitamos para representar correctamente la salida de cada mult
         */
    }
}
```

ESTRUCTURAS DE REPETICIÓN: while

Las instrucciones que forman el cuerpo del bucle podría ser necesario ejecutarlas o no. Es decir, en el bucle while siempre se evaluará la condición que lo controla, y si dicha condición es cierta, el cuerpo del bucle se ejecutará una vez, y se seguirá ejecutando mientras la condición sea cierta.

```
public static void main(String[] args) {
    // Declaración e inicialización de variables
    int numero = 7;
    int contador;
    int resultado=0;

    //Salida de información
    System.out.println ("Tabla de multiplicar del " + numero);
    System.out.println (".............................. ");

    //Utilizamos ahora el bucle while
    contador = 1; //inicializamos la variable contadora
    while (contador <= 10) //Establecemos la condición del bucle
    {
        resultado = contador * numero;
        System.out.println(numero + " x " + contador + " = " + resultado);
        //Modificamos el valor de la variable contadora, para hacer que el
        //bucle pueda seguir iterando hasta llegar a finalizar
        contador++;
    }
}
```

ESTRUCTURAS DE REPETICIÓN: do while

Las instrucciones que forman el cuerpo del bucle necesitan ser ejecutadas, al menos, una vez y repetir su ejecución mientras que la condición sea verdadera.

```
public static void main(String[] args) {
    // Declaración e inicialización de variables
    int numero = 7;
    int contador;
    int resultado=0;

    //Salida de información
    System.out.println ("Tabla de multiplicar del " + numero);
    System.out.println (".............................. ");

    //Utilizamos ahora el bucle do-while
    contador = 1; //inicializamos la variable contadora
    do
    {
        resultado = contador * numero;
        System.out.println(numero + " x " + contador + " = " + resultado);
        //Modificamos el valor de la variable contadora, para hacer que el
        //bucle pueda seguir iterando hasta llegar a finalizar
        contador++;
    }while (contador <= 10); //Establecemos la condición del bucle

}
```