# Java:Bloques de Construcción

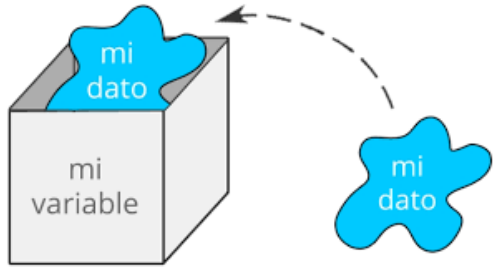
Dr. Mario Marcelo Berón

# Índice

# Tipos de Datos Simples

| Tipos de Java | | |
|---|---|---|
| **Tipo** | **Descripción** | **Rango de Valores** |
| byte | Enteros muy pequeños | -128 a 127 |
| short | Enteros pequeños | -32768 a 32767 |
| long | Enteros grandes | -9223372036854775808 a 9223372036854775808 |
| float | Números Reales | $\mp 1.4 * 10^{-45}$ a $3.4 * 10^{38}$ |
| double | Números Reales grandes | $\pm 4.9 * 10^{-324}$ a $1.8 * 10^{308}$ |
| char | Caracter | Conjunto de caracteres unicode |
| boolean | Verdadero o Flaso | No aplica |

# Variables



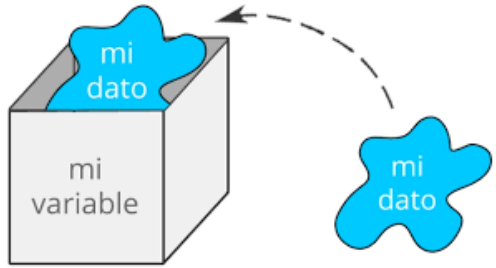

**Variables**: ubicaciones de memoria cuyo valor puede variar durante el transcurso del programa.
Para crear una variable:

- Dar el nombre a la variable.
- Decidir el tipo que mejor refleje la clase de valores que se desea almacenar en la variable.

## Atención

- Por convención los nombres de las variables comienzan con minúsculas y los nombres de las clases con mayúsculas.
- Los nombres tienen que describir el propósito de la variable.

# Variables - Identificadores




1. Un identificador es secuencia de uno o más caracteres (letras, números o guión bajo) que comienza con una letra o un guión bajo.
2. Los identificadores no tienen espacios en blanco.
3. Las letras mayúsculas y minúsculas se consideran diferentes.
4. Cuando un identificador está compuesto por varios nombres generalmente se pone en mayúscula el inicio de cada palabra, excepto posiblemente la primera (*camel case*).

# Variables


Una vez que el nombre y el tipo de dato de la variable se han decidido, la declaración tiene la siguiente forma:

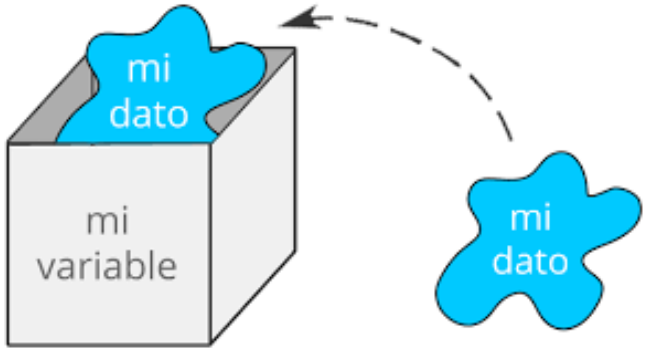
```
tipoDeDato nombreDeLaVariable;
```

donde *tipoDeDato* es el tipo primitivo elegido y *nombreDeLaVariable* es el nombre seleccionado para la variable.

## Ejemplos

```
int score;
double p;
```

# Variables




Cuando el nombre de la variable está compuesto por más de una palabras se pueden utilizar diferentes formas para escribirla:

- Separando las palabras con un guión bajo. Ejemplo: nivel_de_dificulta
- Comenzando la segunda palabra con mayúsculas.
  Ejemplo: nivelDeDificultad.

## Comentarios

Diferentes variables se pueden declarar en una sola línea:

```
int edad, piso, nroDeDomicilio;
```

# Asignación

La asignación permite que los valores sean colocados en las variables.

```
nombreDeLaVariable=valor;
```

## Ejemplo

```
edad=23;
```

Lo anterior se interpreta incializar la variable edad con el valor 23.

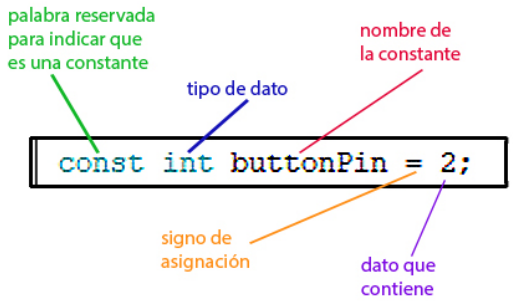
## Atención

- Es posible combinar una declaración con una asignación:

```
int edad = 23;
```

- Siempre es conveniente inicializar las variables explícitamente.
- El tipo del lado izquierdo de la asignación debe coincidir con el tipo del lado derecho de la misma.

# Constantes




Las constantes se declaran como una variable excepto que son precedidas por la palabra *final*.

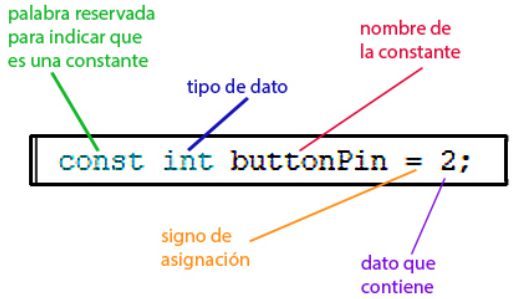
## Ejemplo

```
final int HORAS = 24;
```

## Atención

Por convención los nombres de las constantes se escriben con mayúsculas.

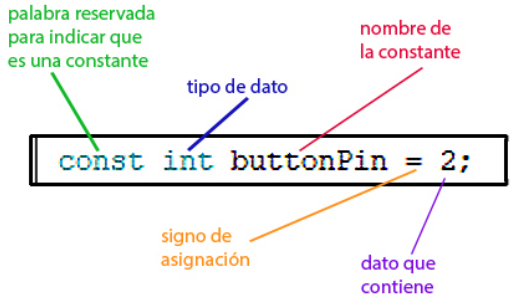
# Constantes Literales




## Características

1. Literal de tipo char y strings. Ejemplo de char 'A','*', etc. Ejemplo de string "Hola".
2. Literales numéricos:
   - 1.3e12 es decir $10^{12}$.
   - Cualquier literal que contenga un punto decimal es de tipo double. Si se desea que el literal sea de tipo float se le debe agregar el sufijo F así 1.2F es un literal de tipo float.

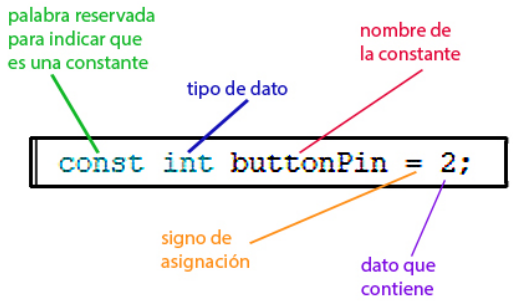
# Constantes Literales




## Características

1. Literal de tipo char y strings. Ejemplo de char 'A','*', etc. Ejemplo de string "Hola".
2. Literales numéricos:
   - En el caso de los literales enteros los mismos pueden ser short, int o long dependiendo de su tamaño. No obstante es importante señalar que un literal entero se puede hacer de tipo long agregando el sufijo L, así 17L es un literal entero Long.

# Constantes Literales




## Características

1. Literal de tipo char y strings. Ejemplo de 'A','*', etc. Ejemplo de string "Hola".
2. Literales numéricos:
   - También se pueden especificar literales enteros en formato binario,octal y hexadecimal.Los primeros empiezan 0b o 0B, los segundos con 0 los terceros con 0x o OX.
   - Los literales numéricos pueden tener un guión bajo para separar grupos de dígitos. Por ejemplo: 7_000_000_000 es mejor que 7000000000.

# Entrada Salida


Java permite que cualquier valor o expresiones de tipos primitivos se puedan imprimir por pantalla. Esta tarea la lleva a cabo convirtiendo implícitamente cada valor/expresión a un string antes de mostrarlo en la pantalla.

```
System.out.print(10*10);
```

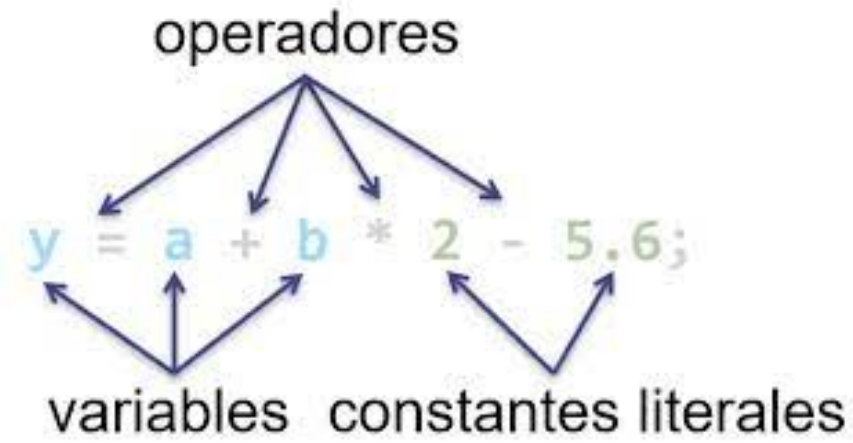
La sentencia imprime por pantalla el número 100.

# Scanner

Permite construir programas que posibilitan la entrada de datos por teclado. Esta tarea puede ser llevada adelante por la clase *Scanner*.

```
import java.util.Scanner;
public class EncontrarCosto {
 public static void main(String[] args ){
    Scanner teclado = new Scanner(System.in);
    double precio, impuesto;
    System.out.println("*** Verificación del Precio del
        Producto ***");
    System.out.print("Ingrese el precio inicial: ");
    precio = teclado.nextDouble();
    System.out.print("Ingrese la tasa de impuestos: ");
    impuesto = teclado.nextDouble();
    precio = precio * (1 + impuesto/100);
    System.out.println("Cost after tax = " + precio);
 }
}
```

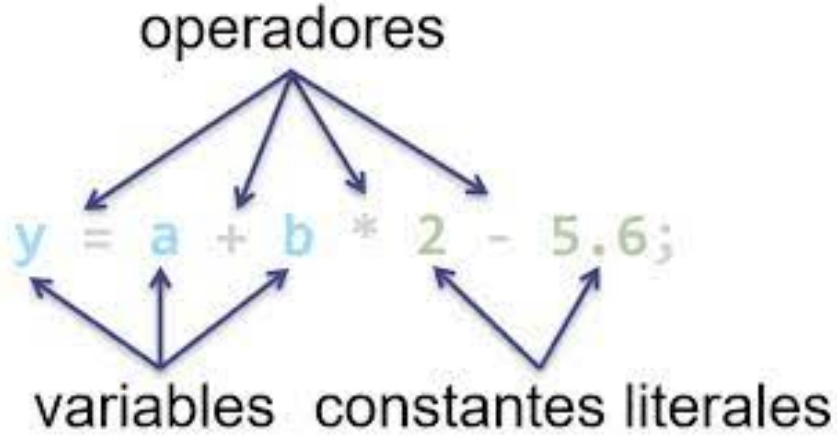
# Expresiones




## Concepción

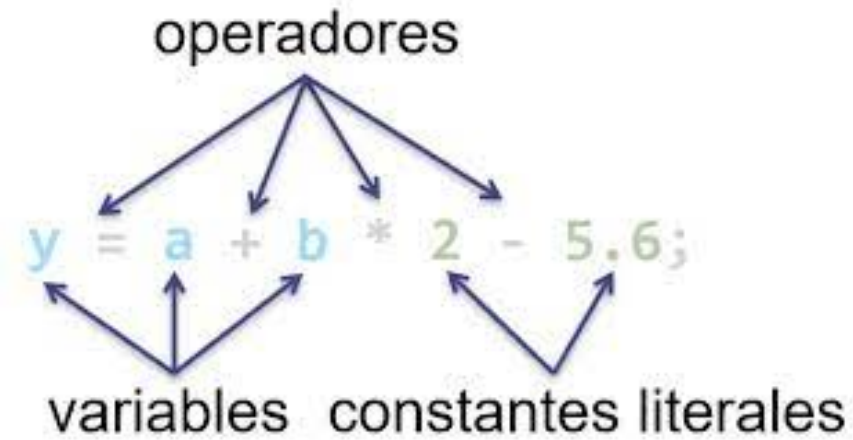
Trozo de código que representa o computa un valor. Una expresión puede ser un literal, una variable, una invocación a función, o combinaciones de los elementos antes mencionados usando operadores artiméticos, relacionales, lógicos,etc.

# Expresiones Artiméticas




1. Son aquellas que se construyen con los *Operadores Artiméticos* +,-,* /.
2. Las operaciones antes mencionadas se pueden usar con valores numéricos de tipo: *byte*, *short*, *int*, *long*, *float* y *double*.
3. Cuando se calcula una expresión todos los operandos deben ser del mismo tipo. En el caso de que los operandos sean de tipos diferentes se lleva a cabo una conversión de tipos.
4. El resultado de una expresión será del mismo tipo que la de sus operandos. Si una operación de suma adiciona dos enteros el resultado será entero.

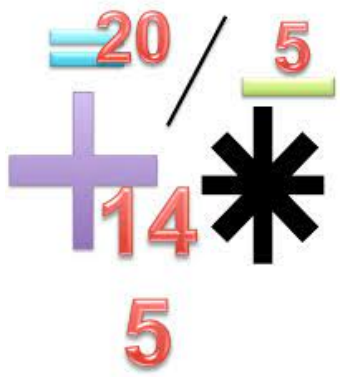
# Expresiones Aritméticas




## Notas y Comentarios

1. Operandos de tipo *char* pueden ser usados en las expresiones, solo se debe tener en cuenta que el valor numérico utilizado se corresponde con su código unicode.
2. Si se realiza una división con enteros la parte decimal se pierde ya que el resultado obtenido es de tipo entero.

# Expresiones Aritméticas

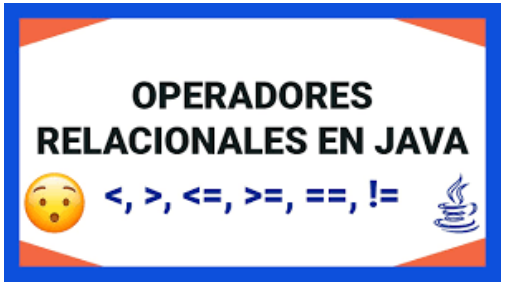



Java, al igual que C/C++, provee los operadores de pre y post incremento/decremento. El funcionamiento de dichos operadores es igual a los provistos por C/C++.

## Ejemplos de Expresiones Aritméticas

```
z=x+1;
w=x++;
k= - -x;
r=x+y+2.0;
r=Math.PI + 28 + z + x*x;
```

# Expresiones Relacionales

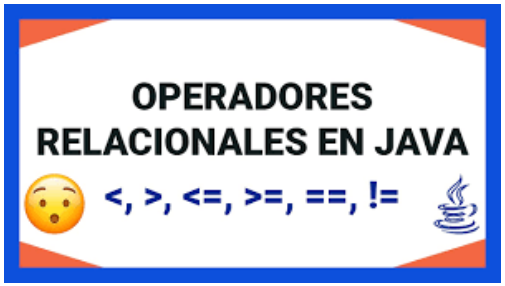



1. Se utilizan para expresar condiciones, el resultado retornando por este tipo de expresiones es *true* o *false*.
2. Las expresiones relacionales utilizan los operadores relacionales: $<, >, <=, >=, ==, != ;$

## Ejemplos de Expresiones Relacionales

1. A==B A es igual a B
2. A != B A es distinto de B
3. A > B A es mayor que B
4. A < B A es menor que B
5. A >= B A es mayor o igual que B

# Expresiones Relacionales




## Notas y Comentarios

1. Los operadores relacionales $<$ y $>$ también se pueden usar con operandos de tipo char. Ellos están definidos acordemente teniendo en cuenta el tipo char. Esta comparación no sigue el orden alfabético.
2. Los operadores $<, ><=, >=$ no pueden ser usados para comparar strings.
3. Los operadores $==, != $ pueden ser usados para comparar strings. Pero por el tipo de objeto que son los strings no proporcionan los resultados deseados.

# Expresiones Lógicas


Son aquellas que se construyen utilizando los operadores booleanos: && (and), || (or), ! (not).

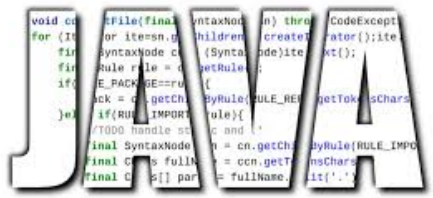
## Ejemplo de Expresiones Lógicas

```
x && y || Z
!z
!z || k && w
```

## Notas y Comentarios

Los operadores booleanos && y || son implementados usando *corto-circuito*.

# Expresiones Lógicas



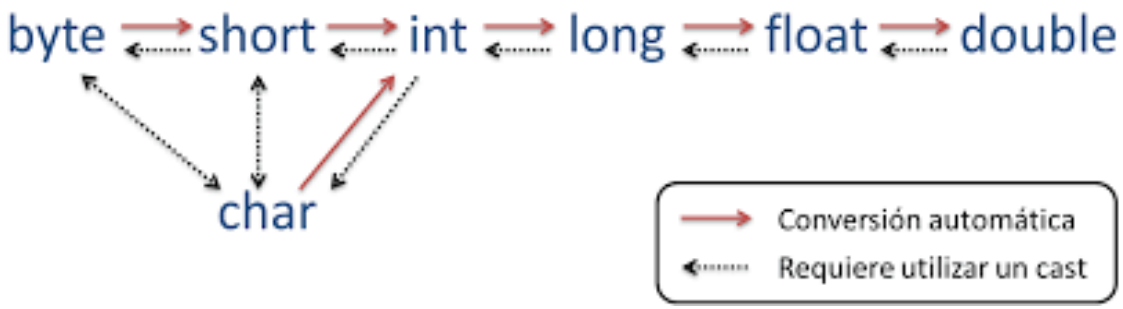

Java al igual que C/C++ provee el operador codicional

## Sintaxis

<expresión booleana >? < expresión 1 >: <expresión 2>

La computadora evalúa la expresión booleana, si la misma produce como resultado verdadero se ejecuta expresión 1 en caso contrario se ejecuta expresión 2.

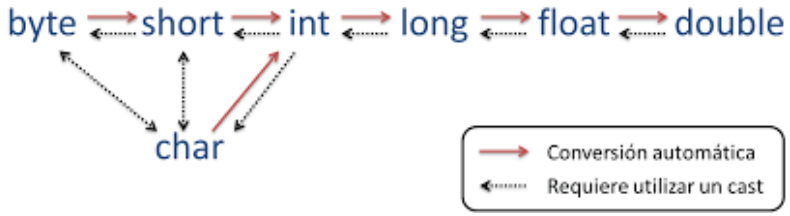
# Operador de Asignación y Conversiones de Tipo




Cuando se usa el operador de asignación dos tipos de conversiones pueden tomar lugar:

1. Automáticas.
2. Aquellas llevadas a cabo por medio de la utilización de un *cast*.

# Operador de Asignación y Conversiones de Tipo



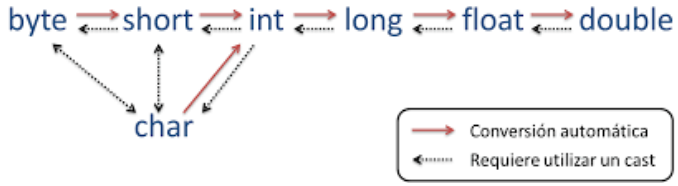

## Conversiones Automáticas

Aquellas realizadas por el compilador para que los tipos de las variables concuerden.

## Ejemplos de convesiones automáticas

```
int a;
double z;
short b;
a = 17;
z = a; // a se convierte a double
b = a; // Error a no se puede convertir a short
```

# Operador de Asignación y Conversiones de Tipo



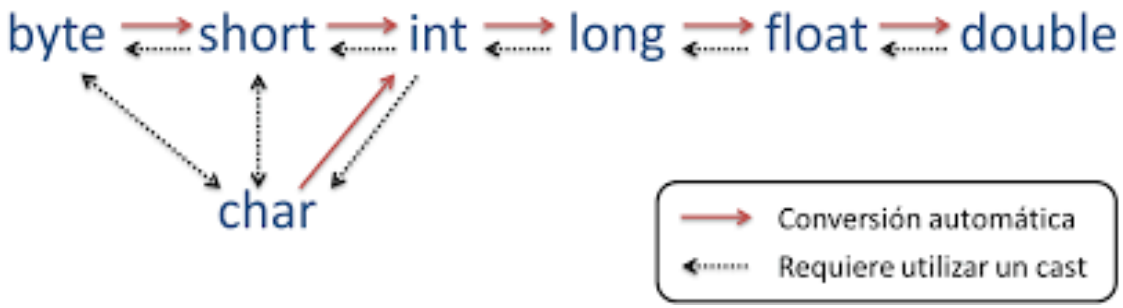

## Conversiones Automáticas

Una conversión automática solo se lleva a cabo si la misma no cambia la semántica del valor.

## Ejemplo de Conversión Legal e Ilegal

```
float a;
int b;
short c;
...
a=b; //Un entero se puede convertir a float
b=c; //Un short se puede convertir a int
c=b; //Un int no se puede convertir a short
//porque cae fuera de su rango.
```

# Operador de Asignación y Conversiones de Tipo




## Conversiones Forzadas por el Programador - Cast

5 Se utiliza cuando se desea realizar una conversión que se sabe no se realizará automáticamente. Para llevar a cabo esta conversión se debe usar un cast, el cual se indica colocando en nombre del tipo entre paréntesis.

# Operador de Asignación y Conversiones de Tipo

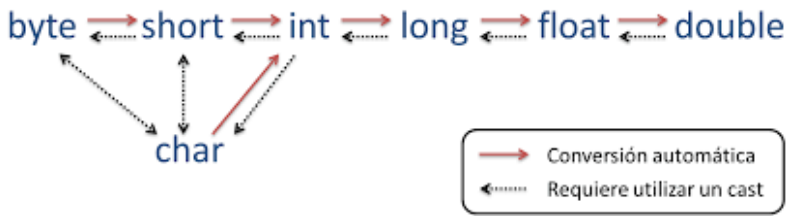
## Ejemplos de convesiones automáticas

```
int a;
short b;
a= 17;
b=(short)a;//a es convertido explícitamente a
//short
```

## Notas y Comentarios

Cuando se lleva a cabo una conversión forzada se debe tener en cuenta que se puede alterar el valor.

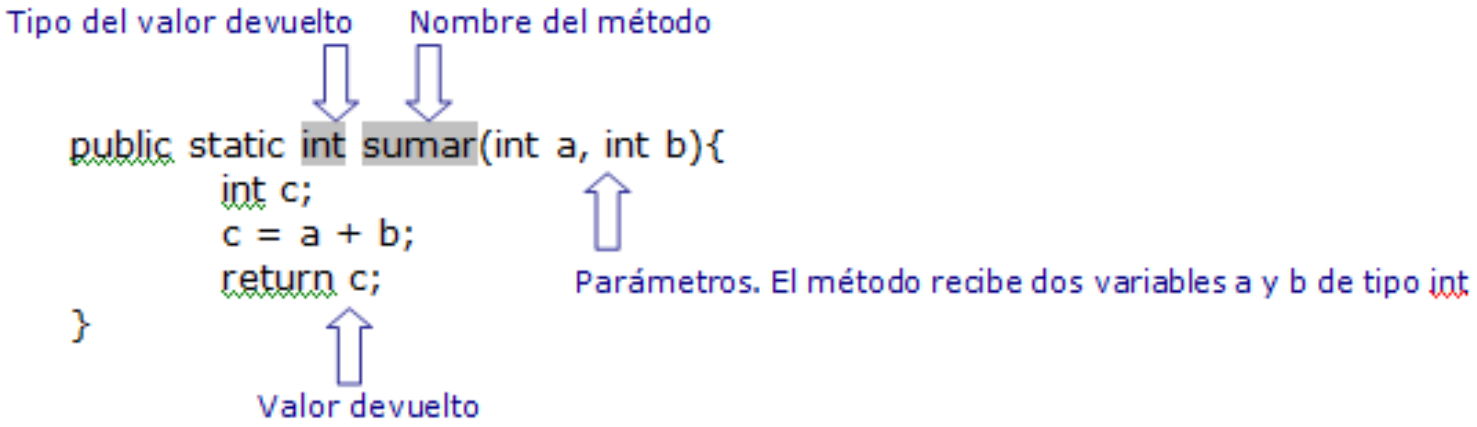
# Precedencia de Operadores




A continuación se presenta la precedencia de los operadores provistos por Java descriptos en esta clase:

**Operadores Unarios:** $++$, $--$, !, unario -, unario $+$, tipo-cast
**Multiplicación y División:** *, /, %
**Suma y Resta:** $+$, -
**Operadores Relacionales:** $<$, $>$, $<=$, $>=$
**Igualdad y Desigualdad:** $==$, $!=$
**Boolean and:** &&
**Boolean or:** $||$
**Operador Condicional:** ?:
**Operadores de Asignación:** $=$, $+=$, $-=$, $*=$, $/=$, $\%=$

# Métodos




Un método es parte de una clase y contiene un conjunto particular de sentencias. Por lo general realizará una tarea bien definida.

# Métodos-Definición

```
[mod–alc.][static] tipo id(lis. de param. formales) {
   sentencias
}
```

- *mod-alc* es opcional y se refiere a la forma en la que el método puede ser accedido. El concepto se verá en las próximas clases.
- *static* es un modificador opcional que indica que el método es un método de clase. Este concepto se estudiará más adelante en el curso.
- *tipo* se refiere a un tipo primitivo o a un objeto debe coincidir con el tipo del resultado del método.

# Métodos-Definición

```
[mod–alc.][static] tipo id(lis. de param. formales) {
  sentencias
}
```

- *id* es el nombre del método.
- *Lis. Parámetros Formales* son los datos de entrada del método. La lista de parámetros formales puede ser vacía.
- *sentencias* es el cuerpo del método el cual puede tener muchas sentencias.

# Métodos-Definición

## Ejemplo

```
public class Ejemplos {
 static double doble(int x){
  return 2 * x;
 }

  ...
}
```

## Atención

El método doble se tiene que definir como static porque para que se pueda invocar desde otro método static que en este caso es main.

# Métodos-Invocación

```
nombreDelMetodo(lista de parametros reales)
```

La invocación del método se lleva a cabo colocando el nombre del método seguido por la lista de parámetros reales.

## Atención

- La cantidad de parámetros reales debe coincidir con la cantidad de parámetros formales.
- La correspondencia entre parámetros reales y parámetros formales es posicional, es decir el primer parámetro real se corresponde con el primer parámetro formal y así siguiendo.

# Métodos-Invocación

## Ejemplo

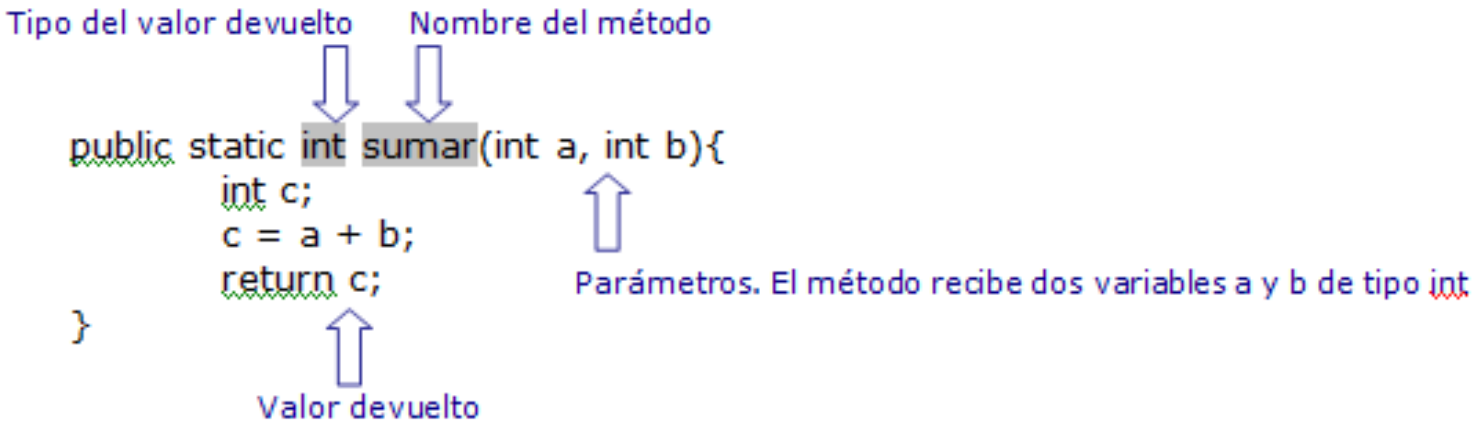
```
public class Ejemplos {
 static double doble(int x){
  return 2 * x;
 }

 public static void main(String []args){
  System.out.println("Doble de 2:"+doble(2));
 }
}
```

## Atención

Siempre que un método retorne un valor se puede invocar dentro de una expresión.

# Métodos-Invocación




## Atención

- Dentro de los métodos se pueden declarar variables y las mismas solo podrán ser referenciadas dentro del método es decir son locales a él.
- Las variables locales nacen cuando comienza la ejecución del método y mueren cuando el método termina su ejecución.

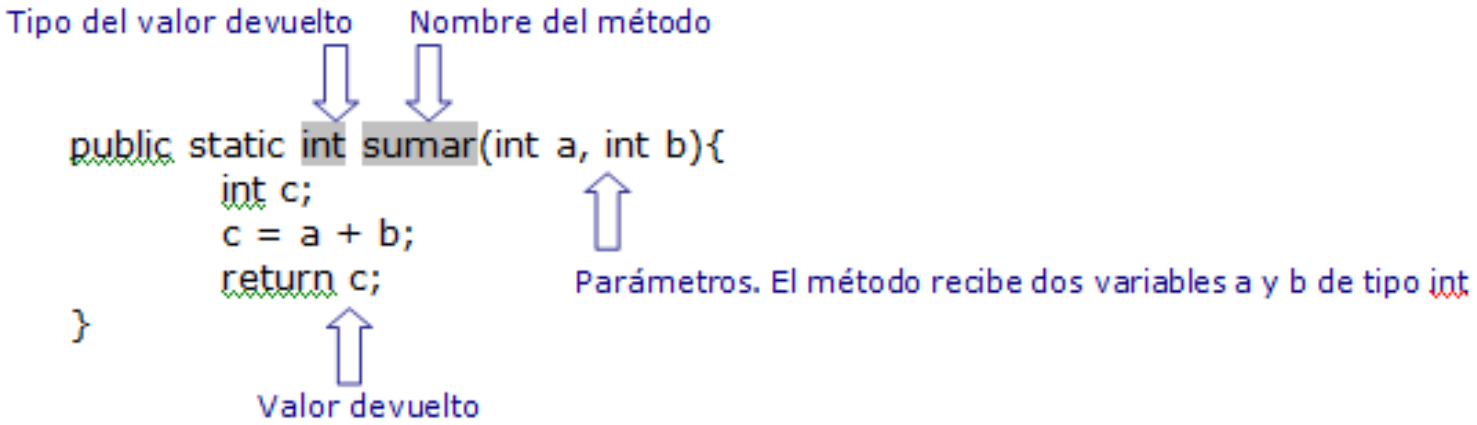
# Métodos-Invocación

```
public class Ejemplos{
  void f(){
    int x;  //nace x
      ...
      ...     //muere x
  }

  void g(){
    int x; //nace x
      ...
      ...    //muere x
  }

  void h(){
    x=.... //error x no esta definida en h
  }
  ...
}
```

# Métodos-Invocación




- Los operadores de división $(/)$ y suma $(+)$ son ejemplos de operadores sobrecargados porque funcionan para propósitos diferentes. La división puede dividir enteros o reales y la suma no solo se utiliza para sumar números sino también para la concatenación.
- Las funciones también se pueden sobrecargar colocando el mismo nombre pero tiene que haber diferencias en la lista de parámetros ya sea en el tipo de los parámetros o en la cantidad de los mismos.