DAM
PROGRAMACIÓN

UD3-POO
API DE JAVA Y MÉTODOS

Biblioteca vs API vs Framework

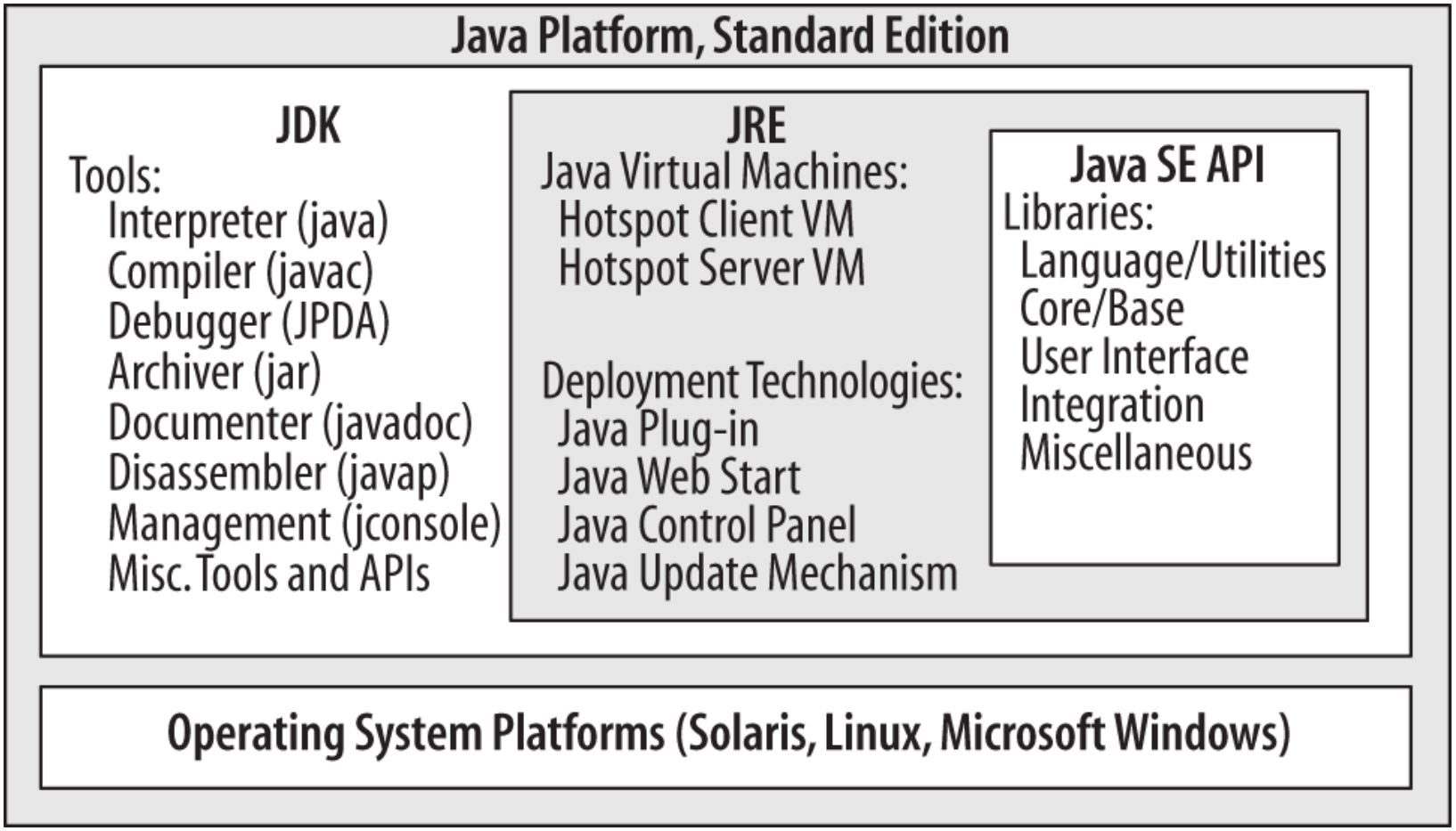
- Una **biblioteca** es un conjunto de clases que ofrece distintas funcionalidades que se pueden utilizar para desarrollar software.
- Una **API** es una *Interfaz para Programación de Aplicaciones*, es decir, es la parte de una biblioteca a la que accede el programa que usa la biblioteca. La API no incluye los detalles de la implementación, sino las herramientas que permiten utilizar la funcionalidad de las bibliotecas.
- Un **framework** es un conjunto integrado de herramientas que facilitan el desarrollo del software. Puede incluir APIs y bibliotecas, pero también puede incluir otros elementos como herramientas de depuración, diseño gráfico, edición ,etc.
- Muchas veces se utiliza el término biblioteca y API indistintamente, pero no es correcto. Por ejemplo, podemos tener una implementación diferente de librerías en C++ para Linux y Windows pero tener la misma API, que es la forma en que se utilizan esas librerías.

JDK. API de Java

- **Al instalar el paquete JDK**, dotamos a nuestro sistema de las herramientas necesarias para poder desarrollar programas en lenguaje Java.
- **El paquete JDK**, además del compilador y la máquina virtual de Java, **contiene** un conjunto de librerías que pueden ser utilizadas según la **API** (*Application Programming Interface*) de **Java**.
- **La API de Java** permite el uso de un gran número de clases como ***System***, ***String***, ***Scanner***, ***ArrayList***, etc. que contienen numerosos métodos para desarrollar software.
- Existen otras APIs que se pueden instalar y utilizar, creadas por empresas, programadores. Por ejemplo,
- Cualquier desarrollador Java puede implementar sus propias librerías y APIs.

JSE (Java Platform, Estándar Edition)

Para desarrollar software Java es necesario instalar el paquete JSE que, como podemos ver en el gráfico, incluye la API de Java



Core de Java

- **El core de Java es el conjunto de librerías integradas** en el paquete *Java Runtime Machine* (JRE) que debe estar instalado en toda máquina que ejecute o desarrolle programas Java.
- **El core de Java está organizado en paquetes**, cada uno de los cuales contiene distintas clases que pueden ser utilizadas desde los programas a través de la API de Java.

Paquete	Contenido
`Java.applet`	**Clases para implementar *applets*, correspondientes a aplicaciones que corren en los browsers.**
`Java.awt`	**Clases para gráficas, componentes Graphic User Interface (GUI) y administradores de control de ventanas, además de clases más especializadas como para procesamiento de imágenes Abstract Window Toolkit (AWT).**
`java.beans`	**Clases e interfaces para construir *JavaBeans*, correspondientes a GUI independientes de plataformas.**
`java.io`	**Clases para control de entradas y salidas, tales como archivos y streams.**
`java.lang`	**Clases que componen el núcleo del lenguaje.**
`java.math`	**Clases para aritmética avanzada, incluyendo manejo de precisión numérica arbitraria.**
`java.net`	**Clases relacionadas con el manejo de redes, tales como datagramas y *sockets*.**
`java.rmi`	**Clases para el manejo de métodos remotos.**
`java.security`	**Clases para aspectos de seguridad, tales como criptografía.**
`java.sql`	**Clases para acceso a base de datos con el lenguaje Standard Query Language (SQL).[1]**
`java.text`	**Clases para internacionalización del idioma, independiente del lenguaje particular.**
`java.util`	**Clases adicionales, tales como estructuras de datos avanzadas y compresión de datos.**

Uso de la API de Java

- Para usar la API de Java tenemos dos niveles:
  - Librerías que no necesitan ser importadas.
    - Clases pertenecientes al paquete **java.lang** como String.
  - Librerías que necesitan ser importadas.
    - Clases pertenecientes a otros paquetes como
      - **java.io**, que contiene clases relacionadas con la entrada (input) y salida de datos (out).
      - **java.math**, que contiene clases que permiten operaciones y cálculos matemáticos.
      - **java.util,** que contiene clases relacionadas con colecciones (listas, conjuntos, etc.), entre otras.
      - etc.

Enlace a un buscador de ejemplos de todos los componentes del API de Java

Clase Math

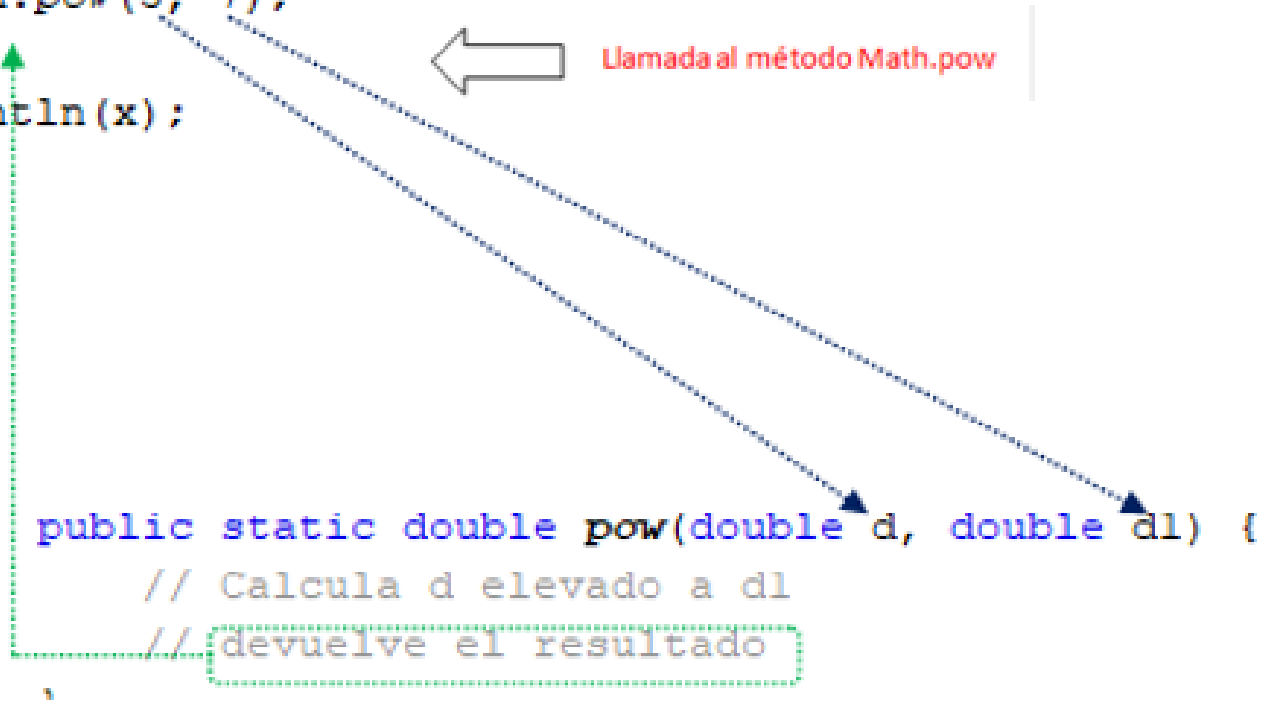
- Pertenece al paquete **java.lang**, por lo que no es necesario importar ningún paquete.
- Se utiliza para realizar operaciones matemáticas mediante métodos estáticos

MÉTODO	DESCRIPCIÓN	PARÁMETROS	TIPO DE DATO DEVUELTO
abs	Devuelve el valor absoluto de un número.	Puede ser un int, double, float o long	El mismo que introduces.
arcos	Devuelve el arco coseno de un ángulo en radianes.	Double	Double
asin	Devuelve el arco seno de un ángulo en radianes.	Double	Double
atan	Devuelve el arco tangente entre -PI/2 y PI/2.	Double	Double
atan2	Devuelve el arco tangente entre -PI y PI.	Double	Double
ceil	Devuelve el entero más cercano por arriba.	Double	Double
floor	Devuelve el entero más cercano por debajo.	Double	Double
round	Devuelve el entero más cercano.	Double o float	long (si introduces un double) o int (si introduces un float)
cos	Devuelve el coseno de un ángulo.	Double	Double
sin	Devuelve el seno de un ángulo.	Double	Double
tan	Devuelve la tangente de un ángulo.	Double	Double
exp	Devuelve el exponencial de un número.	Double	Double
log	Devuelve el logaritmo natural en base e de un número.	Double	Double
max	Devuelve el mayor de dos entre dos valores.	Pueden ser dos int, double, float o long	El mismo tipo que introduces.
min	Devuelve el menor de dos entre dos valores.	Pueden ser dos int, double, float o long	El mismo tipo que introduces.
random	Devuelve un número aleatorio entre 0 y 1.	Ninguno	Double
sqrt	Devuelve la raíz cuadrada de un número.	Double	Double
pow	Devuelve un número elevado a un exponente.	Double (base y exponente)	Double

```
public static void main(String[] args) {

    double x = Math.pow(3, 7);

    System.out.println(x);
}
```



Clase Math. Ejemplo

```
public class PruebaApp {
  public static void main(String[] args) {

    double operador1=25.5;
    double operador2=15.21;

    //Redondea al entero más cercano por arriba=> 26.0
    System.out.println(Math.ceil(operador1));

    //Redondea al entero más cercano por abajo=> 15.0
    System.out.println(Math.floor(operador2));

    //Devuelve operador1 elevado a operador 2 => 2.474435537975361E21
    System.out.println(Math.pow(operador1, operador2));

    //Devuelve el valor mayor entre dos valores 25.5
    System.out.println(Math.max(operador1, operador2));

    //Devuelve la raíz cuadrada de operador1 => 5.049752469181039
    System.out.println(Math.sqrt(operador1));
  }
}
```

Métodos

- Un método en Java es conjunto de instrucciones definidas dentro de una clase que puede ser invocado desde su misma clase o desde otras.
- La llamada a un método implica la ejecución de las instrucciones que contiene.
- En otros lenguajes de programación son denominados, subprogramas, funciones, subrutinas o procedimientos.

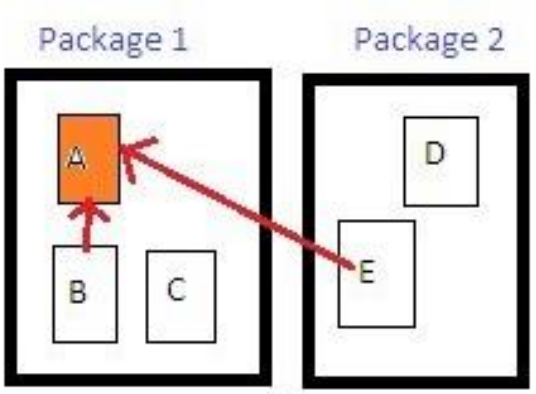
```
[modificador] tipo_resultado nombre_metodo ([lista de parámetros]) {
        declaración de variables locales;
        sentencias
        [return expresión]
}
```

Estructura general de un método

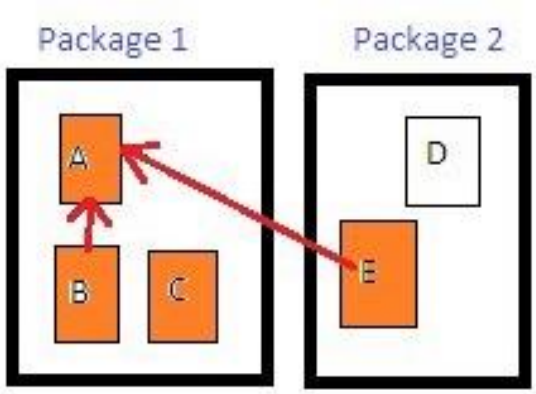
Métodos. Modificadores de acceso

- Los modificadores de acceso permiten establecer el nivel de visibilidad de una clase, método o atributo.
- Existen cuatro niveles de acceso:
  - **por defecto**: es el modificador que posee el método/clase/atributo cuando no especificamos ningún nivel de acceso. Solamente las propia clase o clases del mismo paquete podrán acceder a dichos miembros
  - **private**: Indica que los atributos o métodos solamente pueden ser llamados en la propia clase donde se definen
  - **protected**: Indica que los atributos o métodos solamente pueden ser llamados en la propia clase donde se definen o en clases derivadas (haciendo uso de herencia)
  - **public**: Indica que cualquier objeto en cualquier clase puede acceder a las clases, métodos o atributos etiquetadas con este modificador de acceso

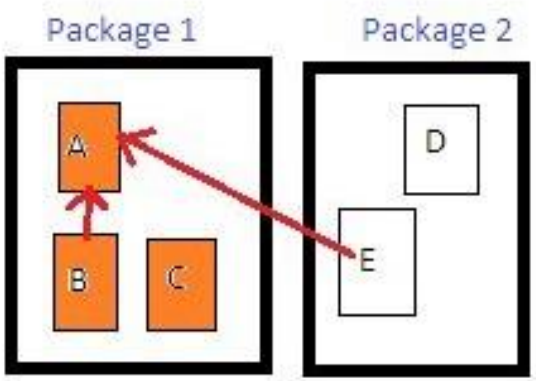
Modificador de acceso	Misma clase o anidada	Clase en el mismo paquete	Clase que hereda en otro paquete	Clase que no hereda en otro paquete
private	Sí	No	No	No
default	Sí	Sí	No	No
protected	Sí	Sí	Sí	No
public	Sí	Sí	Sí	Sí



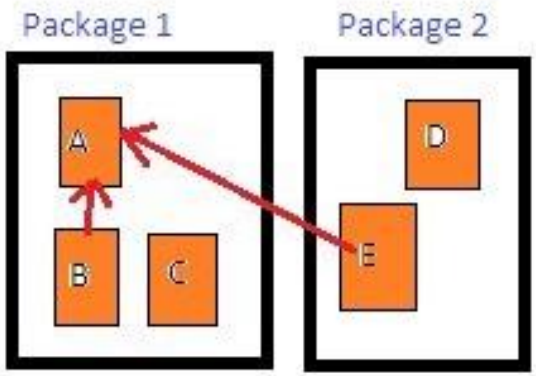
PRIVATE



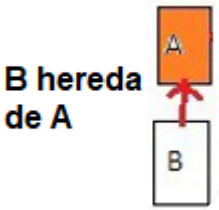
PROTECTED



DEFAULT



PUBLIC



Modificadores de acceso en Java
Fuente: wikipedia.org

Métodos. Valor de retorno

- Los métodos pueden devolver un valor, que puede ser de un tipo básico o una instancia de cualquier objeto.
- El tipo que se indica en **tipo_resultado** debe ser el de la expresión del **return**.

```
[modificador] tipo_resultado nombre_metodo ([lista de parámetros]) {
        declaración de variables locales;
        sentencias
        [return expresión]
}
```

```
int metodoEntero(){
    int suma = 5+5;
    return suma;
}
```

- Si el método no devuelve ningún valor, debemos de poner en **tipo_resultado** la palabra clave **void** y omitir la instrucción **return**.

Métodos. Paso de parámetros

- Un método puede recibir cero, uno o más parámetros.
- **Los parámetros, son datos de entrada que pueden ser rellenos en la llamada al método**.
- Dentro del cuerpo del método, los parámetros pueden ser usados como variables.
- Cuando se definen los parámetros de un método, se indica de qué tipo son.
- Cuando hay dos o más parámetros son separados por comas.

llamadas al metodo

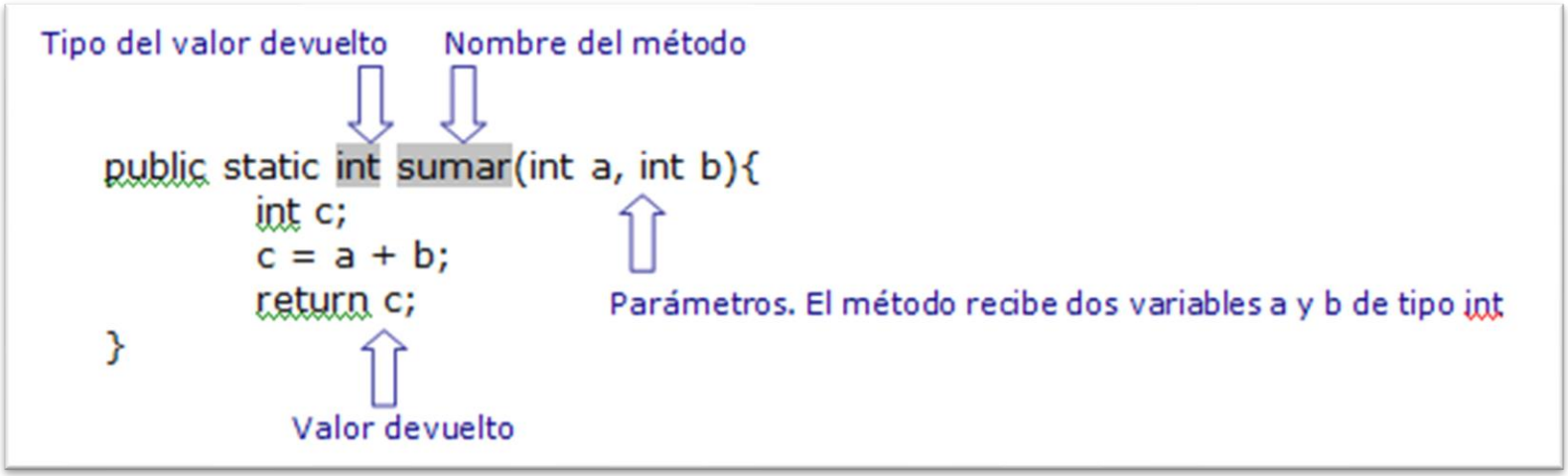
implementación del método

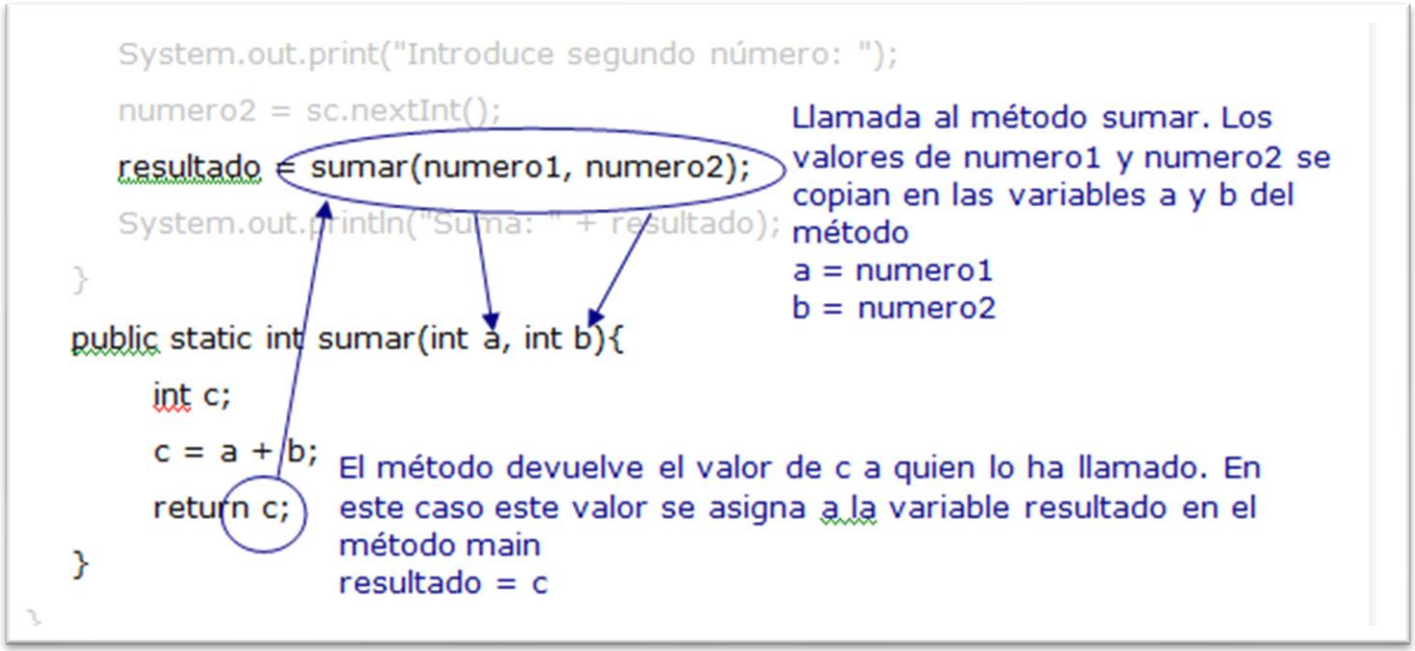
```
int metodoEntero(int a, int b){
    int suma = a+b;
    return suma;
}
```

```
int valor1=4, valor2=5;

//Llamada al método pasando variables como parámetros
int resultado1 = metodoEntero (valor1, valor2);

//Llamada al método pasando literales como parámetros
int resultado2 = metodoEntero (6, 7);
```

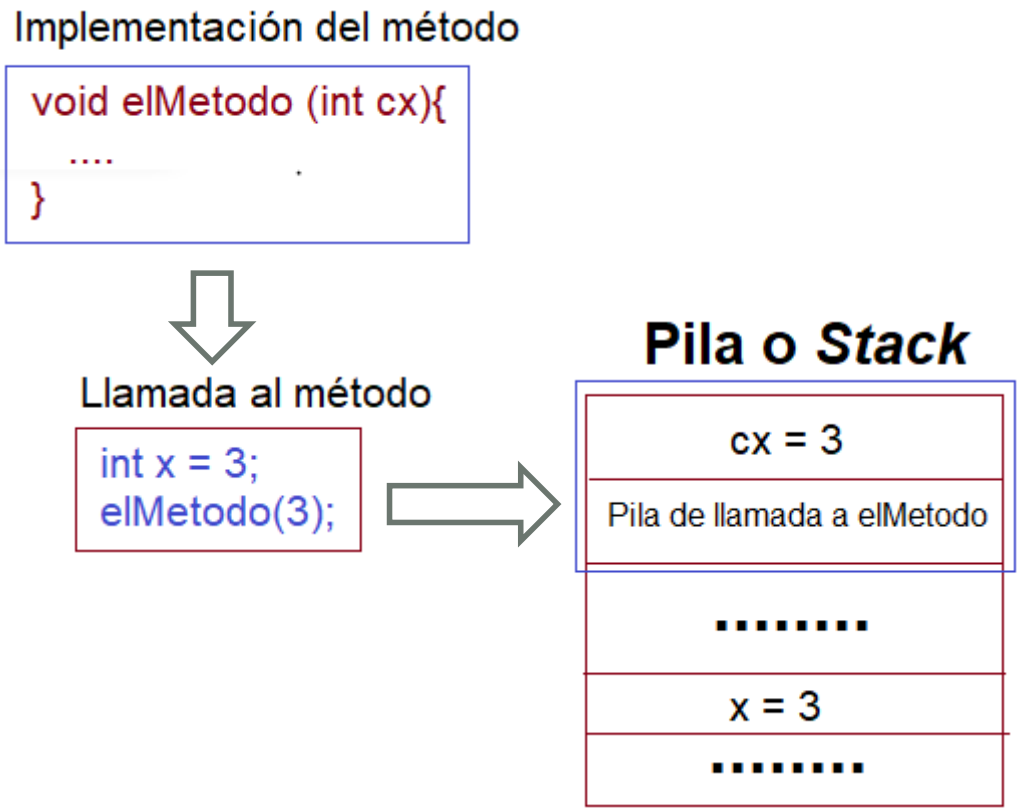

Tipo del valor devuelto
Nombre del método
public static int sumar(int a, int b){
int c;
c = a + b;
return c;
}
Parámetros. El método recibe dos variables a y b de tipo int
Valor devuelto


System.out.print("Introduce segundo número: ");
numero2 = sc.nextInt();
resultado = sumar(numero1, numero2);
System.out.println("Suma: " + resultado);
}
public static int sumar(int a, int b){
int c;
c = a + b;
return c;
}
Llamada al método sumar. Los valores de numero1 y numero2 se copian en las variables a y b del método
a = numero1
b = numero2
El método devuelve el valor de c a quien lo ha llamado. En este caso este valor se asigna a la variable resultado en el método main
resultado = c

Tipos de parámetros

- En Java podemos diferenciar dos tipos de parámetros
  - **Tipos primitivos**
  - **Objetos**
- **Los tipos primitivos** (*int, double, char, etc.*) se guardan directamente en la pila o *stack*. Cuando se llama a un método, se almacena en su espacio asignado una copia de los parámetros que se corresponden a tipos primitivos.
- **Los objetos** usan la pila para almacenar la dirección de memoria del *heap* donde están físicamente almacenados. Cuando se llama a un método se almacena, en su espacio asignado, una copia de la dirección de memoria del *heap* del objeto, que es lo que realmente se pasa por parámetro.
- El valor que se pasa por parámetro, en los dos casos, es el valor almacenado en la *pila* de esa variable, ya sea tipo primitivo (*almacena el valor de la variable*) u objeto (*almacena la dirección de memoria del heap donde está el objeto*).

Parámetros: Tipos primitivos



- Los cambios que se efectúan a los parámetros de los tipos primitivos dentro del método no afectan a la variable que se usa en la llamada al método, pues se trabaja con una copias.
- El método crea una copia de todos los parámetros de tipos primitivos y trabaja con ellos sin actuar sobre la variable/valor que se usa en la llamada al método.

Parámetros: Objetos

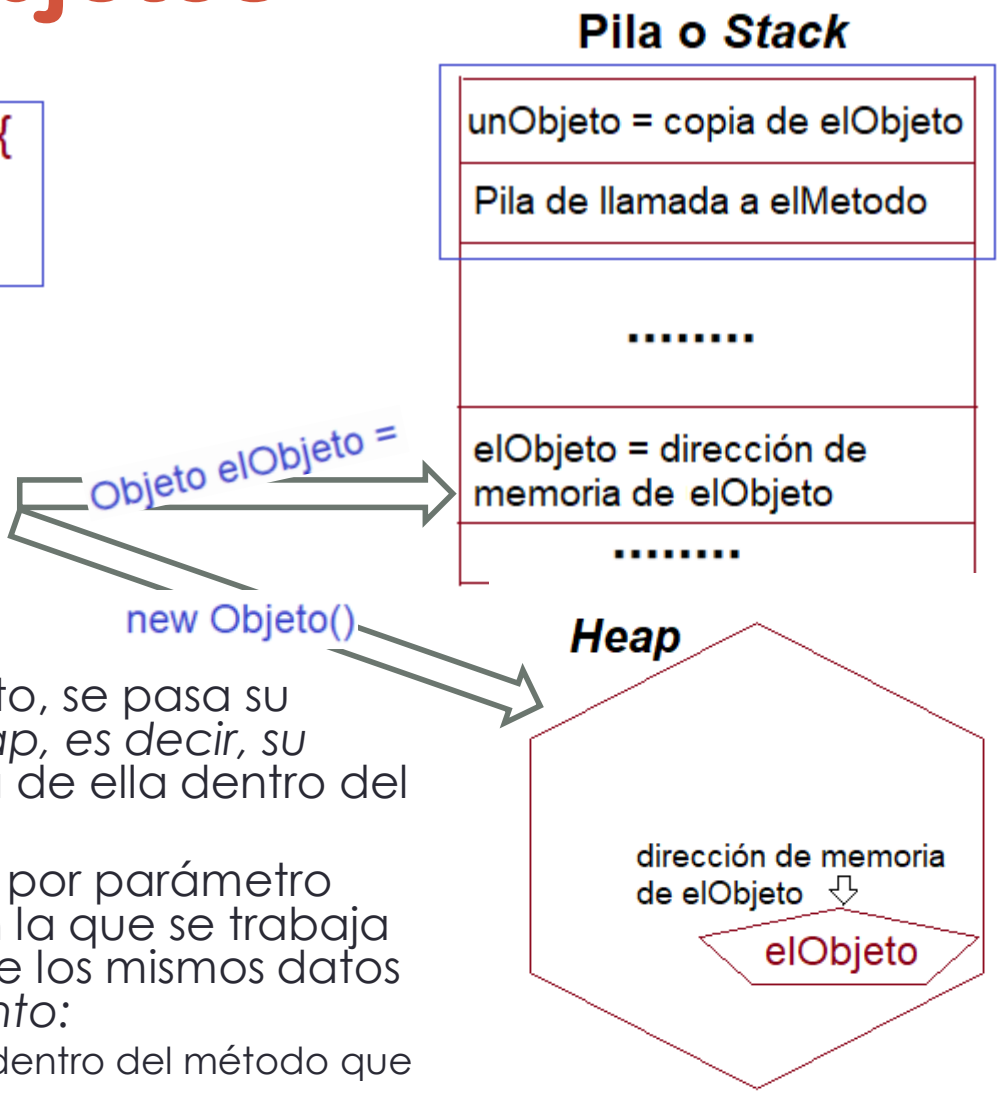
Implementación del método

```
void elMetodo (Objeto unObjeto){
    ......
}
```

Llamada al método

```
Objeto elObjeto = new Objeto();

elMetodo(elObjeto);
```



- Al pasar por parámetro un objeto, se pasa su dirección de memoria en el *heap, es decir, su referencia,* y se hace una copia de ella dentro del método.
- Tanto la referencia que se pasa por parámetro como la copia de la misma con la que se trabaja dentro del método actúan sobre los mismos datos del objeto en el *heap, por lo tanto:*
  - El objeto referenciado es el mismo dentro del método que fuera de él.
  - Cualquier modificación que se realice dentro del método por medio de la referencia del objeto, modifica el objeto.

Métodos estáticos o de clase

- **Métodos que están asociados a una clase y no a los objetos producidos por la clase.**
- Se identifican por la palabra **static** antes del valor de retorno.

```
[modificador] static tipo_resultado nombre_metodo ([lista de parámetros]) {
        declaración de variables locales;
        sentencias
        [return expresión]
}
```

- Para llamar a un método estático se utiliza la siguiente sintaxis.

NombreClase.nombreMétodo

```
class Persona{
   static int totalDePersonas;

   static int agregarUnaNuevaPersona(){
      Persona.totalDePersonas = Persona.totalDePersonas + 1;
   }
   ...
}
```

```
Persona.agregarUnaNuevaPersona();
```

llamada al método estático

implementación del método estático

Métodos de instancia

- **Métodos que están asociados a objetos producidos por la clase a la que pertenecen.**
- No tienen asociado ninguna palabra clave para su implementación.
- Es necesario crear un objeto de la clase para llamar a los métodos de la misma.
- Por ejemplo, la clase **StringBuilder** posee varios métodos de clase que requieren de la creación de un objeto para poder ser utilizados.

```
StringBuilder texto3 = new StringBuilder("Otra prueba");

texto3.append(" mas");
System.out.println("Texto 3 es: " + texto3);

texto3.insert(2, "W");
System.out.println("Y ahora es: " + texto3);

texto3.reverse();
System.out.println("Y ahora: " + texto3);
```

```
Texto 3 es: Otra prueba mas
Y ahora es: OtWra prueba mas
Y ahora: sam abeurp arWtO
```

Métodos de instancia. Ejemplo

```
public static void main(String[] args) {
  Vehiculo coche1 = new Vehiculo();
  Vehiculo coche2 = new Vehiculo();

  int km1, km2;

  coche1.pasajeros=9;
  coche1.capacidad=15;
  coche1.ckl=2;

  coche2.pasajeros=10;
  coche2.capacidad=25;
  coche2. ckl =3;

  km1=coche1.recorre();
  km2=coche2.recorre();

  System.out.println("El coche1 puede llevar " +
                coche1.pasajeros +" una distancia de : "+km1);
  System.out.println("El coche2 puede llevar " +
                coche2.pasajeros +" una distancia de : "+km2);
}
```

```
class Vehiculo {
    //nº de pasajeros
    int pasajeros;
    //capacidad del depósito en litros
    int capacidad;
    //en km x litro
    int ckl;
    //Km que puede recorrer con un depósito
    int recorre (){
        return capacidad / ckl;
    }
}
```