# Sesión 4: Métodos

# Métodos

Un **método** en Java es un **conjunto de instrucciones** definidas dentro de un bloque, que realiza una **tarea determinada**.

Existen dos tipos de métodos:

- Métodos que realizan procesos o cálculos sobre variables para obtener algún resultado o llevar a cabo alguna tarea. Son los que vamos a estudiar ahora..
- Métodos predefinidos en clases, que realizan procesos sobre variables definidas por la clase y, en muchos casos, brindan acceso a estos datos desde otra clase externa. Este segundo tipo se verán más adelante.

Algunos métodos que hemos utilizado:

```
System.out.println();

teclado.nextInt();

String.equals()
```

# Métodos: Forma general

**Forma o estructura general de un método**

```
[especificadores] tipoDevuelto nombreMetodo([lista parámetros]) {
    // instrucciones
   [return valor;]
}
```

- especificadores (opcional): determinan el **tipo de acceso** al método (*public, private, static...*). Se verán en detalle más adelante.
- tipoDevuelto: indica el **tipo del valor** que devuelve el método. En Java es imprescindible indicar el tipo de dato que se va a devolver cuando se declara un método. Habrá que escribir ***int*** **o** ***String*** **si devuelve un entero o un texto**. Si el método **no devuelve ningún valor**, se escribirá ***void***.
- nombreMetodo: es el nombre que se le da al método. Los métodos suelen **tener como nombre la acción que realizan**. El nombre del método suele empezar por minúscula y utilizar mayúsculas para separar frases (Scanner.nextInt).
- Un método empieza cuando se abre la llave ‘{‘ y termina cuando se cierra ‘}’ (bloque)

# Métodos: Forma general

- lista parámetros (opcional): tras el nombre del método, siempre entre paréntesis, se puede especificar una lista de parámetros. Son las **variables que se usarán en el método**.
  - Van **separados por comas**, en caso de que haya más de uno.
  - Estos parámetros son los **datos de entrada** que recibe el método para operar con ellos.
  - Un método puede recibir **cero o más argumentos**.
  - Para cada argumento, se debe **especificar su tipo de dato** (*int, double, String…*).
  - **Los paréntesis son obligatorios en los métodos**. Se escriben aunque no haya parámetros y estén vacíos.
- return: se utiliza para **devolver** un valor. La palabra clave *return* va seguida de un dato o una expresión que será que será el valor que devuelve el método.
  - Se puede devolver desde una variable de tipo primitivo hasta una expresión o un objeto.
  - El tipo de dato devuelto en *return* tiene que coincidir con el tipo tipoDevuelto que se ha especificado en el método. Por ejemplo, en un método que calcula una división, si al declarar el método especificamos que va a devolver un *double*, en *return* tendrá que devolverse un valor o expresión de tipo double.
  - En caso de que el método sea *void*, no se tiene que poner *return*.

# Métodos: cómo implementarlo

Los pasos a seguir y/o tener en cuenta al diseñar un método:

- Describir lo que el **método tiene que hacer**
- Determinar los **parámetros de entrada**
- Determinar el **tipo de los parámetros** de entrada
- Determinar si **devuelve algún valor**
- Si devuelve un valor, determinar el **tipo de valor retornado**.
- Escribir las **instrucciones** del método.
- **Probar** que funciona. Diseñar distintos casos de prueba.

# Métodos: Ejemplos

- Ejemplo 1: Método que no recibe parámetros y no devuelve nada.

  Método que imprime un texto de error.

```
public class Ejemplo1 {

  public static void main(String[] args) {

      int n = 0;
      if (n <= 0) {
          imprimirError();
      }
  }

  public static void imprimirError() {
      System.out.println("Error: el número no es mayor que 0");
  }
}
```

# Métodos: Ejemplos

- Ejemplo 2: Método que recibe parámetros y devuelve un valor..

  Método que suma dos números.

```
public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
      int numero1, numero2, resultado;
      System.out.print("Introduce primer número: ");
      numero1 = sc.nextInt();
      System.out.print("Introduce segundo número: ");
      numero2 = sc.nextInt();
      resultado = sumarNumeros(numero1, numero2);
      System.out.println("Suma: " + resultado);
  }
  public static int sumarNumeros(int a, int b){
     int c;
     c = a + b;
     return c;
  }
```

# Métodos: Ejemplos

- Ejemplo 3: Método que recibe parámetros con varios return

  Método que dice si el número es positivo.

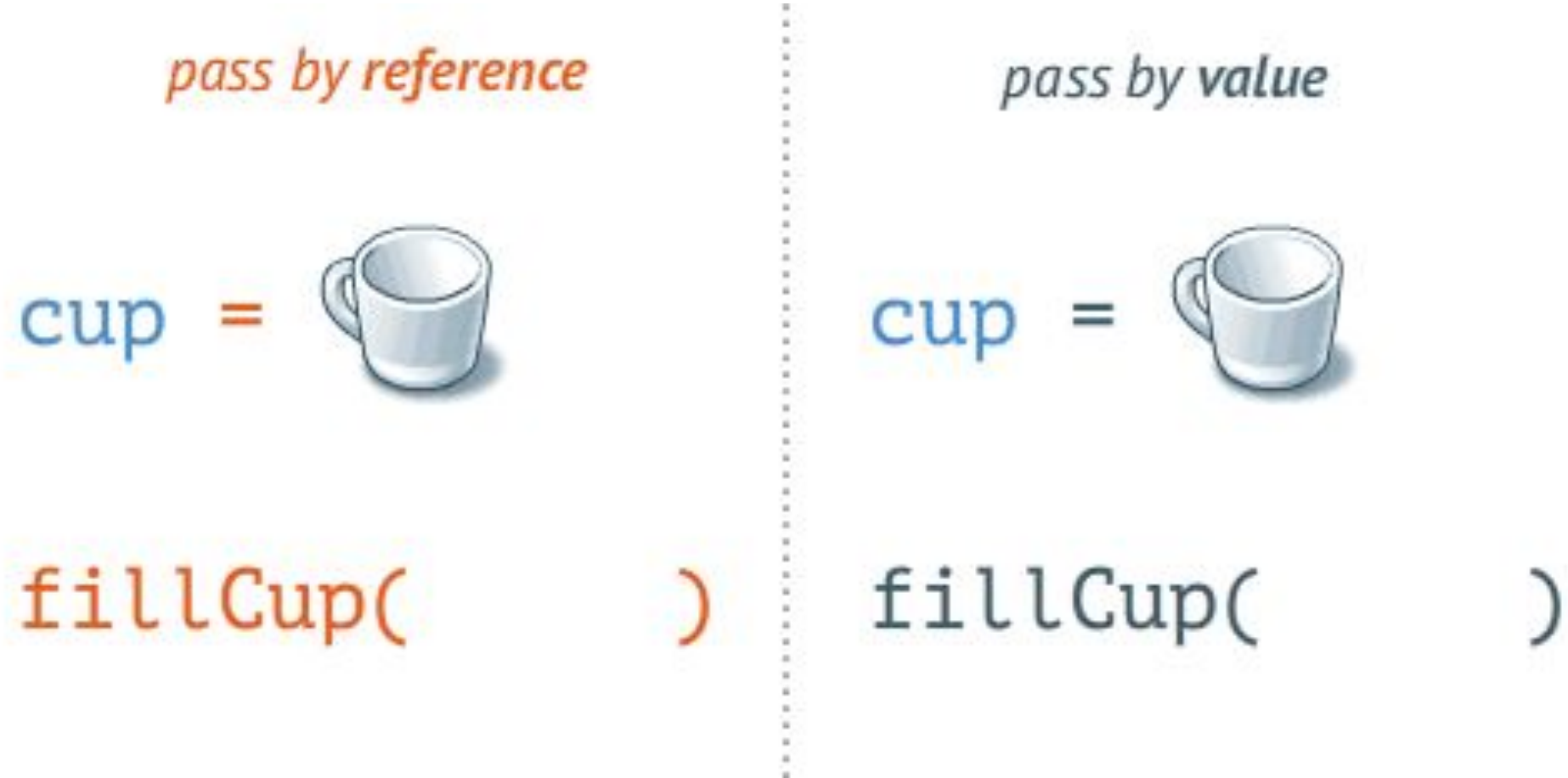
```
public static void main(String[] args) {
     Scanner sc = new Scanner(System.in);
      int numero1;
      boolean positivo;
      System.out.print("Introduce un número: ");
      numero1 = sc.nextInt();

      positivo = esPositivo(numero1);
          System.out.println("¿El  número  es  positivo?:  "  +
positivo);
  }
  public static boolean esPositivo (int a){
     if (a > 0)
        return true;
     else
        return false;
  }
```

# Paso de Parámetros en Java

En el mundo de la programación se conocen 2 formas de pasar parámetros a una función o método:

- paso por valor: se pasa una copia del valor que contiene la variable.
- paso por referencia: se pasa la dirección de memoria de la variable original.

**En Java sólo existe el paso por valor.**



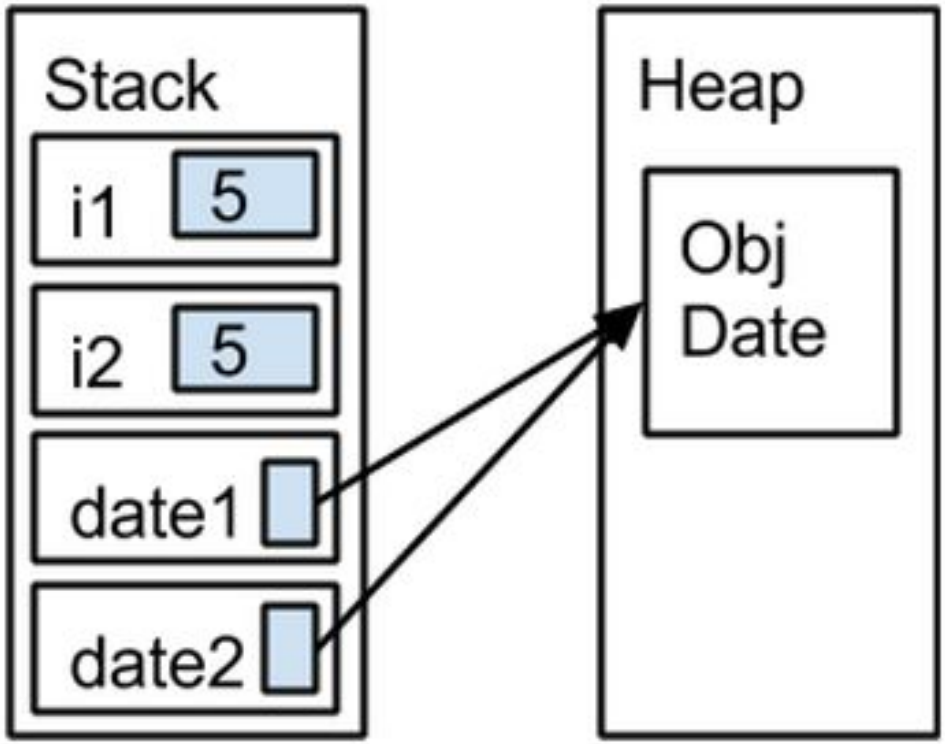
# Paso de Parámetros en Java

Todas las variables de tipos primitivos o de referencia se guardan en memoria de la pila *(stack).*

Además, los datos de una variable de tipo primitivo queda en el stack, pero los objetos que contienen los datos a los que apunta una variable de referencia se almacenan en el heap.

Esto tiene ciertas implicaciones a la hora de hacer asignaciones

```
int i1 = 5;
int i2 = i1;
Date date1 = new Date();
Date date2 = date1;
```



# Paso de Parámetros en Java

Durante la invocación a un método, los argumentos **siempre se pasan por valor:**

- Para los tipos primitivos o simples, el valor se copia simplemente en la memoria de la pila, que luego se pasa al método llamado.

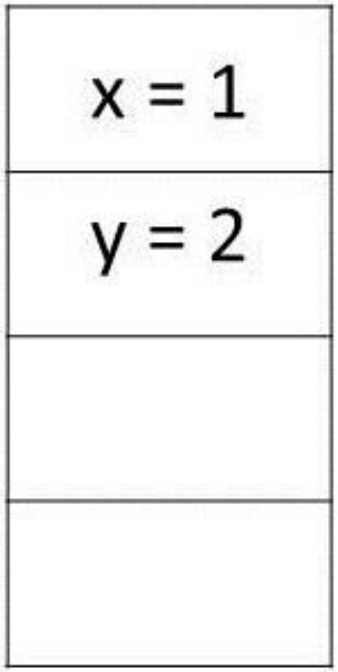
- En el caso de los no primitivos, en la memoria de la pila hay una referencia que apunta a los datos reales que residen en el (heap). Es esa nueva referencia la que se le pasa al método.

# Paso de tipos simples en Java: Ejemplo

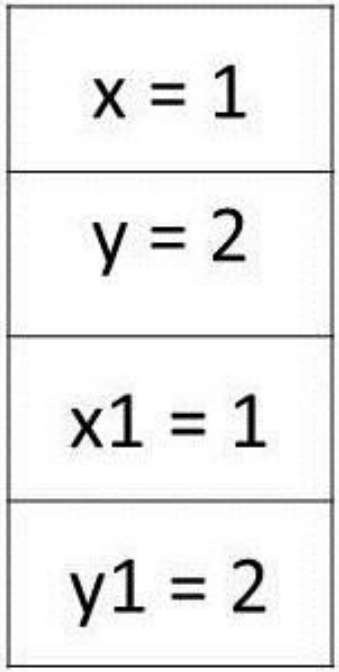
```
public class ArgumentosPrimitivos {
  public static void main(String[] args) {
      int x = 1;
       int y = 2;

       System.out.println("Antes de la Modificación");
       if(x == 1) { System.out.println("x ES 1"); }
       if(y == 2) { System.out.println("y ES 2"); }

       modificar(x, y);

       System.out.println("Después de la Modificación");
       if(x == 1) { System.out.println("x ES 1"); }
       if(y == 2) { System.out.println("y ES 2"); }
  }

  public static void modificar(int x1, int y1) {
        x1 = 5;
        y1 = 10;
  }
}
```

# Paso de tipos simples en Java: Ejemplo

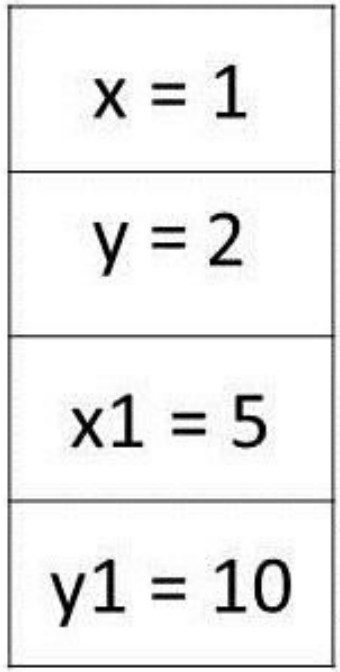
Initial Stack space




Stack space when *modify() method called*



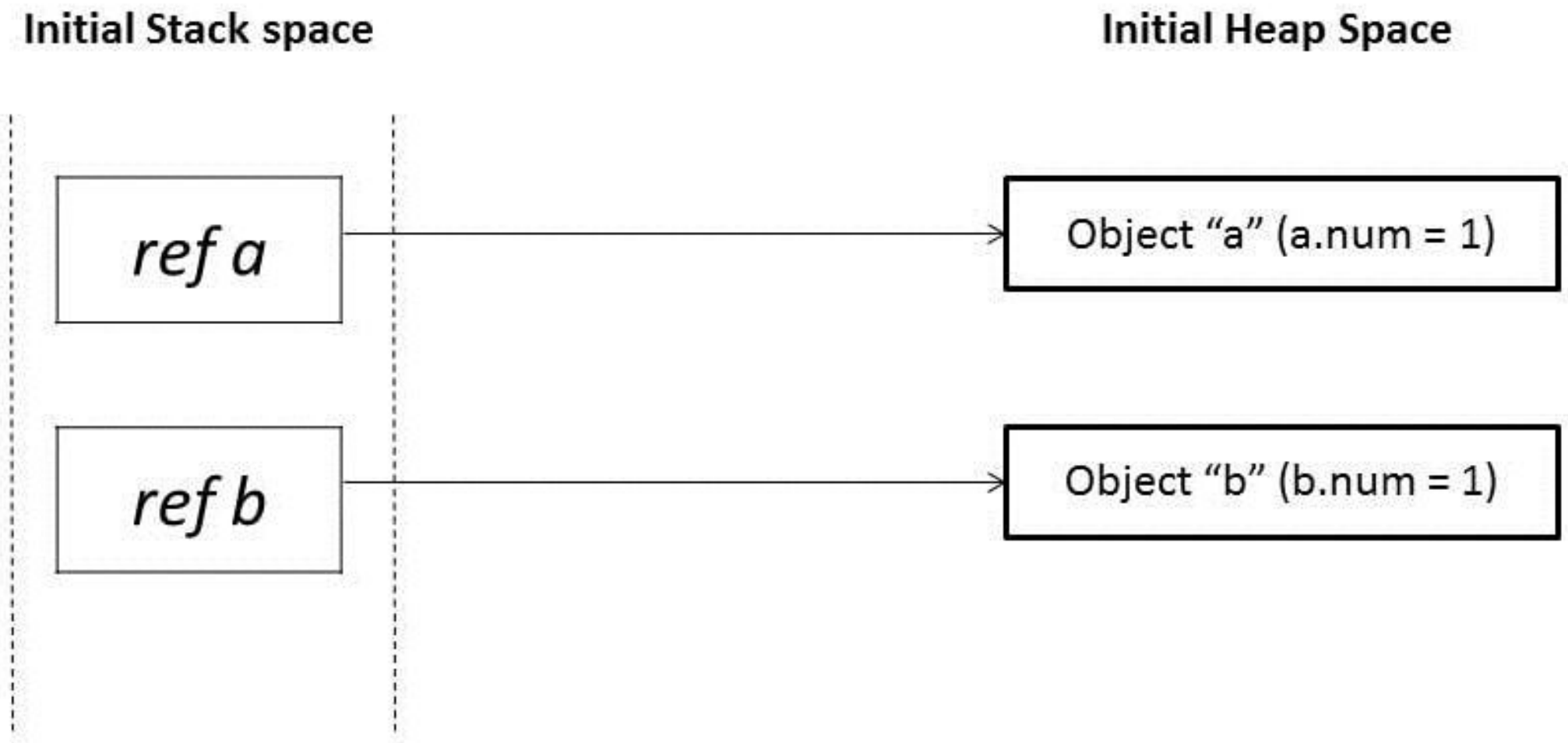

Stack space after *modify()* method call



# Paso de tipos no simples en Java: Ejemplo

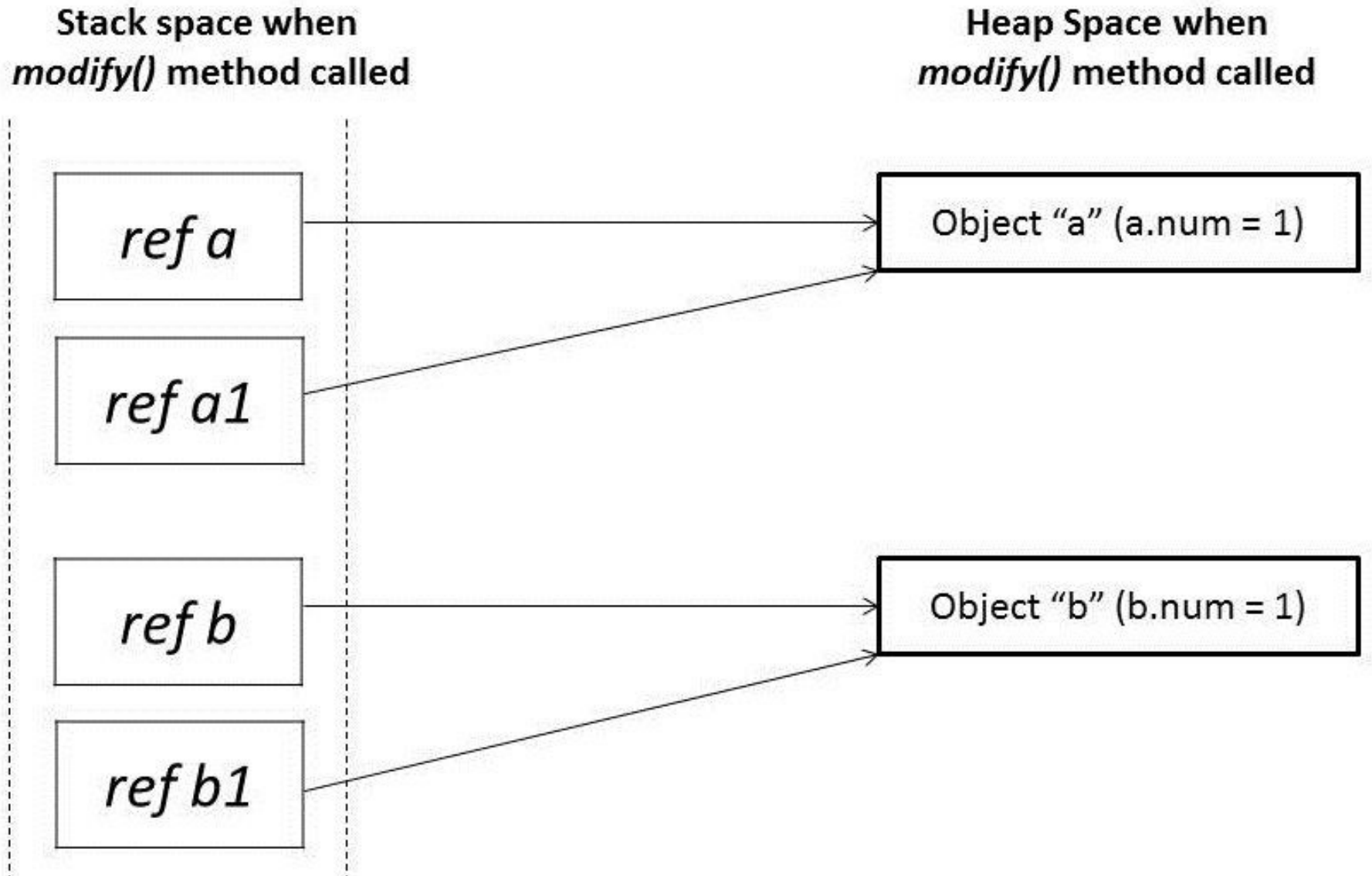
```
public class ArgumentosObjetos {
  public static void main(String[] args) {
      Foo a = new Foo(1);
      Foo b = new Foo(1);

      System.out.println("Antes de la Modificación");
      if(a.numero == 1) { System.out.println("a contiene 1"); }
      if(b.numero == 1) { System.out.println("b contiene 1"); }

      modificar(a, b);

      System.out.println("Después de la Modificación");
      if(a.numero == 2) { System.out.println("a contiene 2"); }
      if(b.numero == 1) { System.out.println("b contiene 1"); }
  }

   public static void modificar(Foo a1, Foo b1) {
          a1.numero++;

          b1 = new Foo(1);
          b1.numero++;
      }
}
```

# Paso de tipos no simples en Java: Ejemplo



# Paso de tipos no simples en Java: Ejemplo

En la llamada a modificar() se crean copias de esas referencias a1 y b1 que apuntan a los mismos objetos antiguos:



# Paso de tipos no simples en Java: Ejemplo

En el método *modificar*, cuando modificamos la referencia a1, cambia el objeto original. Como a b1 le hemos asignado un nuevo objeto que está ahora en la memoria del heap, Todo cambio realizado en b1 no se reflejará en el objeto original.