UF.2
DISEÑO MODULAR

NF.1.1. Programación modular

Diseño modular

- Nuestro objetivo es implementar programas:
  - fiables
  - fáciles de entender
  - fáciles modificar
  - fáciles mantener
  - fáciles reutilizar

- La forma natural de evitar todos este problemas es dividir nuestro programa en módulos que juntos realicen la misma tarea que el programa monolítico.

Diseño TOP-DOWN

- El diseño descendente (top-down, en inglés) es la técnica que se basa al partir de un problema general y dividirlo en problemas más simples, denominados subproblemes. De entre todos estos, los considerados todavía demasiados complejos se vuelven a dividir en nuevos subproblemes. Se denomina descendente porque partiendo del problema grande se pasa a problemas más pequeños a los cuales dará solución individualmente.

Métodos

```
int sum = 0;
for (int i = 1; i <= 10; i++) {
    sum += i;
}
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 37; i++) {
    sum += i;
}
System.out.println("Sum from 20 to 37 is " + sum);

sum = 0;
for (int i = 35; i <= 49; i++) {
    sum += i;
}
System.out.println("Sum from 35 to
```

Problema: Calcular la suma de números enteros de 1 a 10, de 20 a 37 y de 35 a 49.

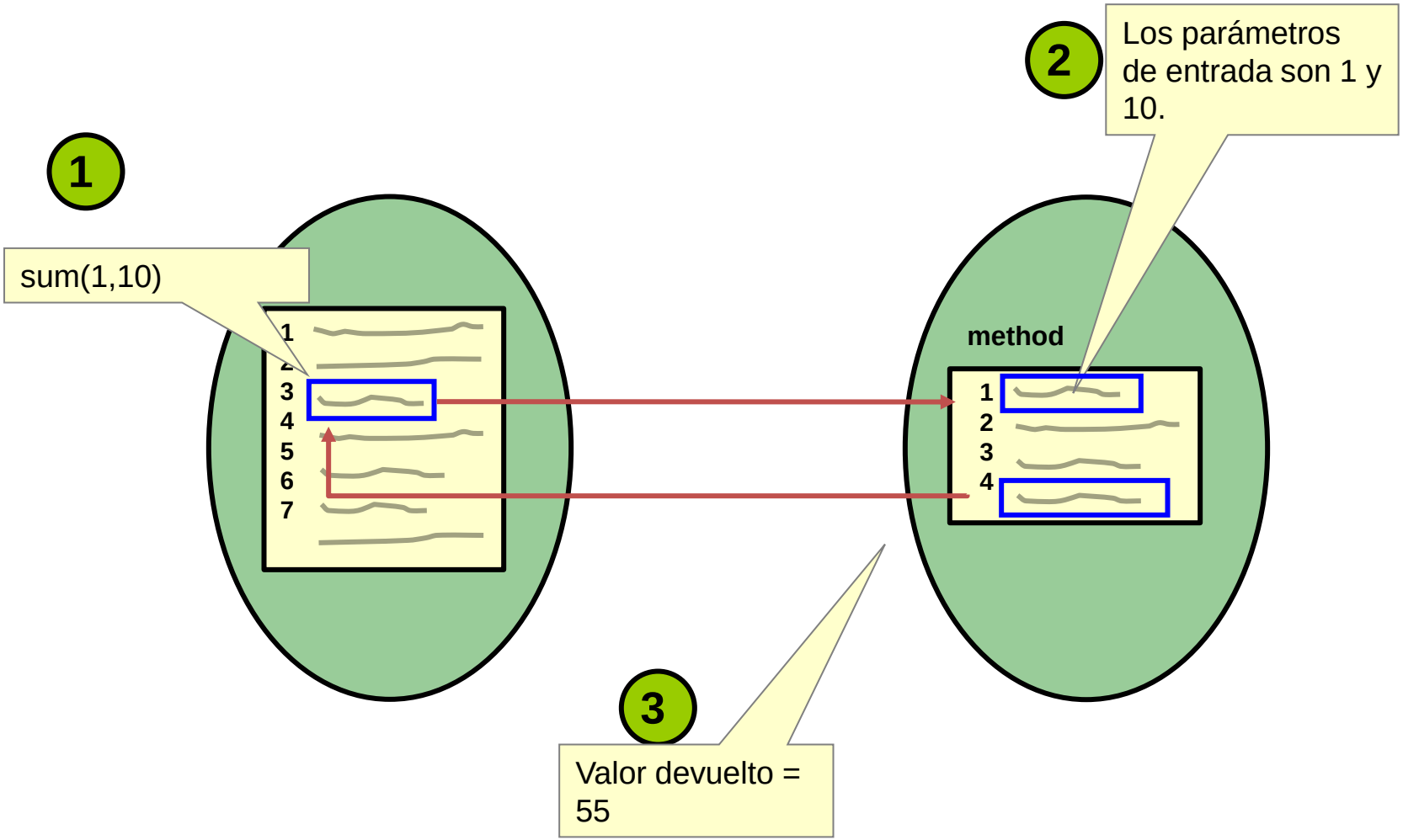
Es el mismo procedimiento para calcular las 3 sumas

MÉTODOS

```
    System.out.println("Sum from 1 to 10 is " + sum(1, 10));
    System.out.println("Sum from 20 to 37 is " + sum(20, 37));
    System.out.println("Sum from 35 to 49 is " + sum(35, 49));
}

public static int sum(int i1, int i2) {
    int result = 0;
    for (int i = i1; i <= i2; i++) {
        result += i;
    }
    return result;
}
```

Solución más efectiva: creando un método:
- Sirve para calcular cualquier suma a partir de dos valores.

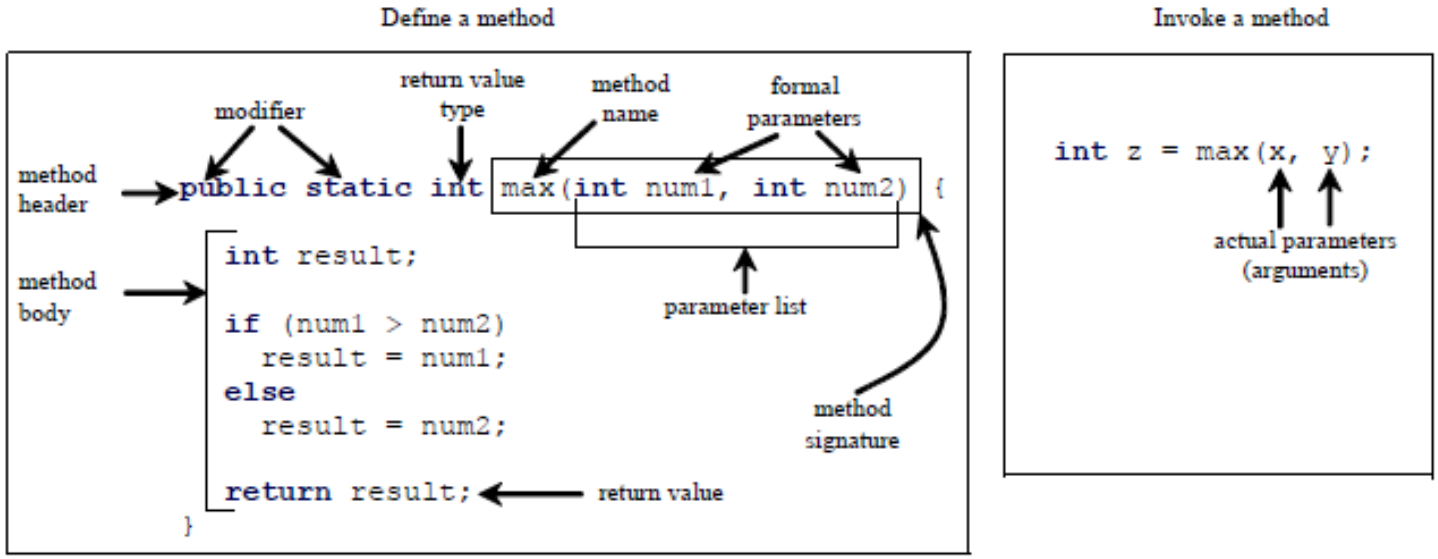
Passing Arguments and Returning Values



Creación de métodos
Sintaxi

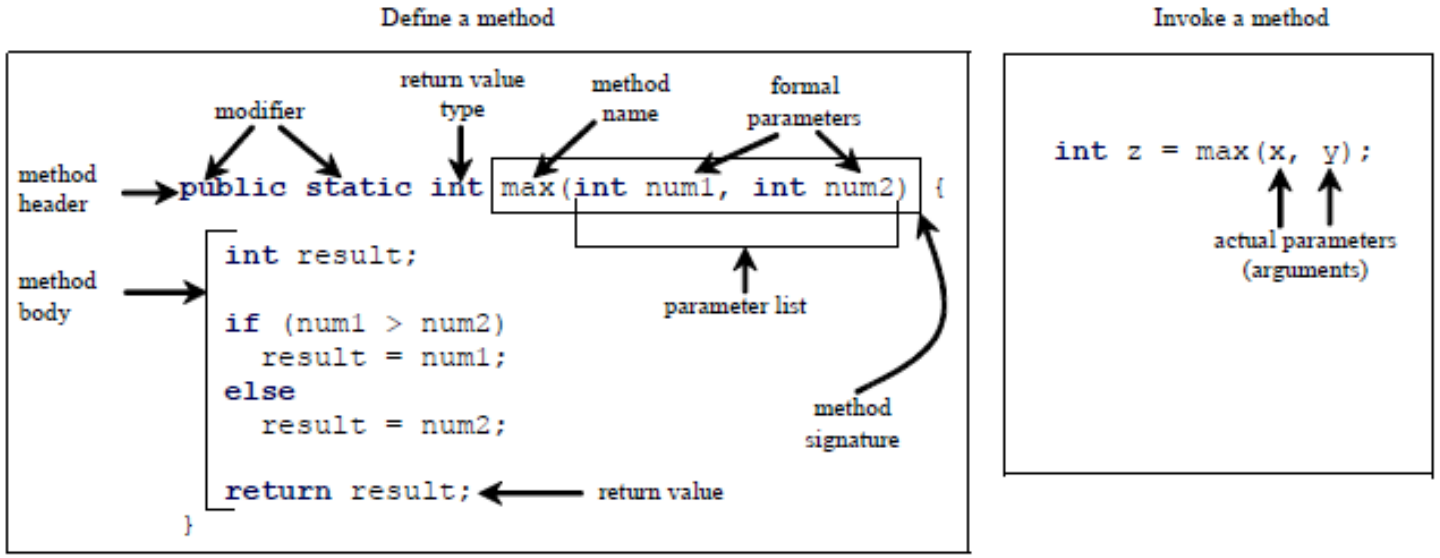
```
[modifiers] return_type method_identifier ([arguments]) {
        method_code_block
}
```

El siguiente método calcula a partir de dos enteros cual es el más grande. El método se llama max, tiene dos parámetros de entrada de tipo integer: num1 y num2, el más grande será devuelto por el método.



Creación de métodos
Sintaxi - Modifiers

```
[modifiers] return_type method_identifier ([arguments]) {
        method_code_block
}
```



MODIFIER: determinan el tipo de acceso al método.
PUBLIC: accesible por cualquier clase.
PRIVATE: sólo es accesible dentro de la clase que se ha declarado.
PROTECTED: es accesible dentro de la clase donde es declarado y desde una clase que herede de ella.
STATIC: es un método de clase. No requiere un objeto para invocarse.

Creación de métodos
Sintaxi – tipo devuelto

```
[modifiers] return_type method_identifier ([arguments]) {
        method_code_block
}
```

```
void
int
String
[] int
[] String
byte
double
```

Tipo Devuelto(return-type): indica el tipo del valor que devuelve el método.

Void indica que el método no devuelve ningun valor.

```
public  void  display ()  {
        System.out.println("Shirt ID: " + shirtID);
        System.out.println("Shirt description:" + description);
        System.out.println("Color Code: " + colorCode);
        System.out.println("Shirt price: " + price);
} // end of display method
```

Creación de métodos
Sintaxi – nombre del método

```
[modifiers] return_type method_identifier ([arguments]) {
        method_code_block
}
```

Nombre del método (method identifier): es el nombre que se le da al método.

Creación de métodos
Sintaxi – Lista de parámetros

```
[modifiers] return_type method_identifier ([arguments]) {
        method_code_block
}
```

Lista de parámetros (arguments): dentro de los paréntesis puede aparecer una lista de parámetros (también llamados argumentos) separados por comas.
Estos parámetros son los datos de entrada que recibe el método para operar con ellos.
Un método puede recibir cero o más argumentos.
Se debe especificar para cada argumento su tipo.
Los paréntesis son obligatorios aunque estén vacíos.

No hay parámetros de entrada.

```
public  void  display ()  {
        System.out.println("Shirt ID: " + shirtID);
        System.out.println("Shirt description:" + description);
        System.out.println("Color Code: " + colorCode);
        System.out.println("Shirt price: " + price);
} // end of display method
```

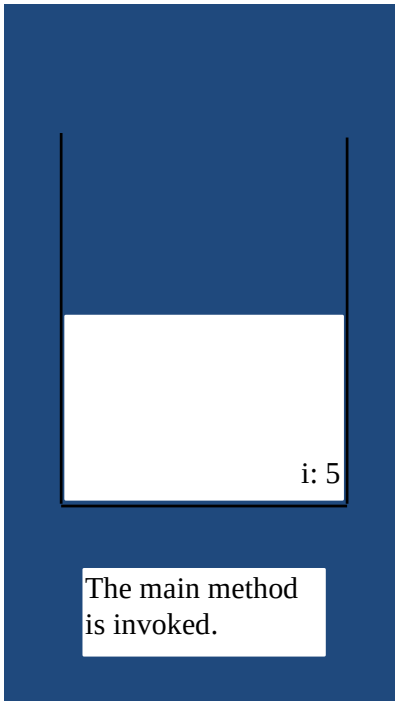
Trace Call Stack

Cada vez que se invoca un método, el sistema almacena los parámetros y variables locales en una área de memoria, conocida como PILA (Stack).

Se declara la variable i y se inicializa a 5.

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

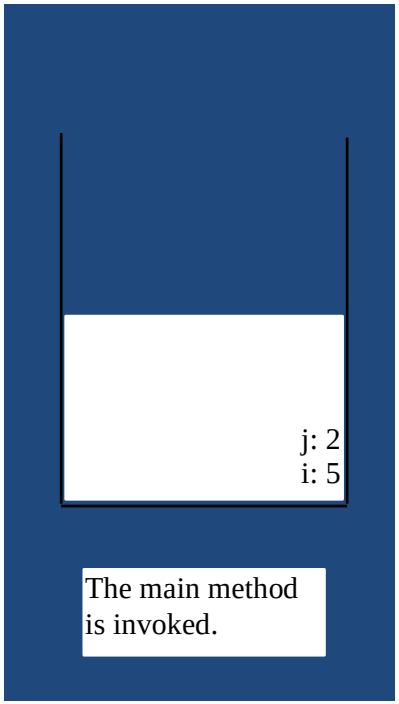


Trace Call Stack

Se declara la variable j y se inicializa.

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

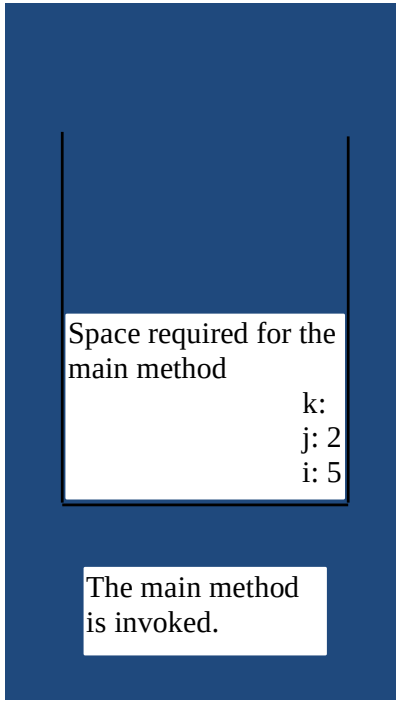


Trace Call Stack

Se declara la variable k, donde se almacenará el valor que devuelve la función max.

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
    "The maximum between " + i +
    " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

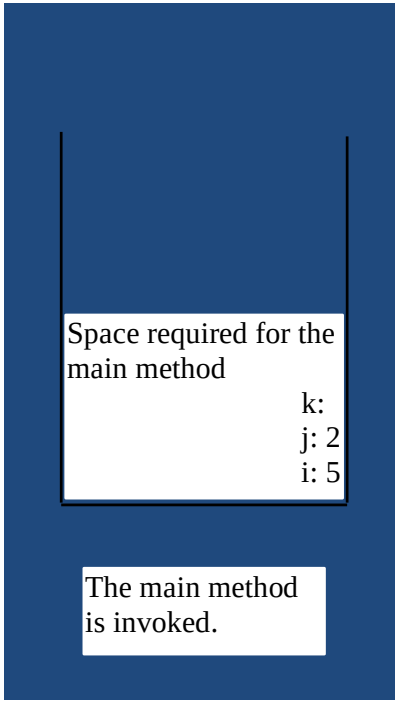


Trace Call Stack

Se llama al método max(i, j)

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

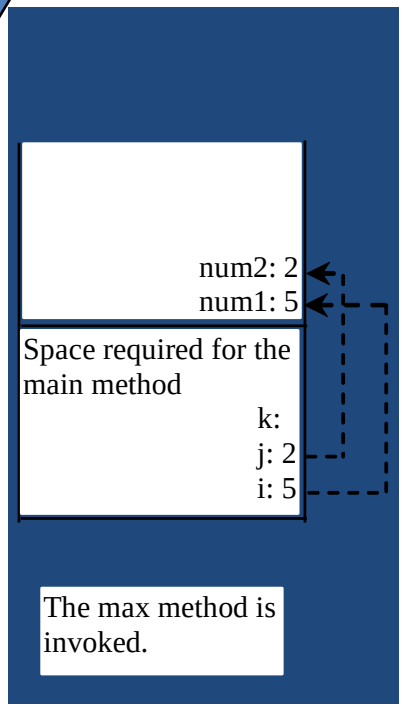


Trace Call Stack

Copia el valor de i a num1 y de j a num2

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

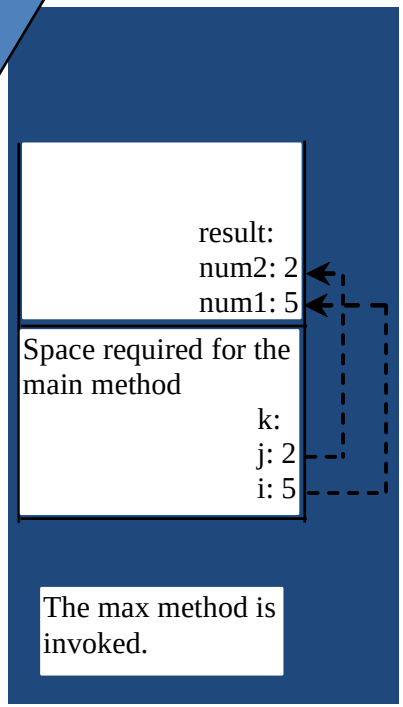


Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Se crea la variable result

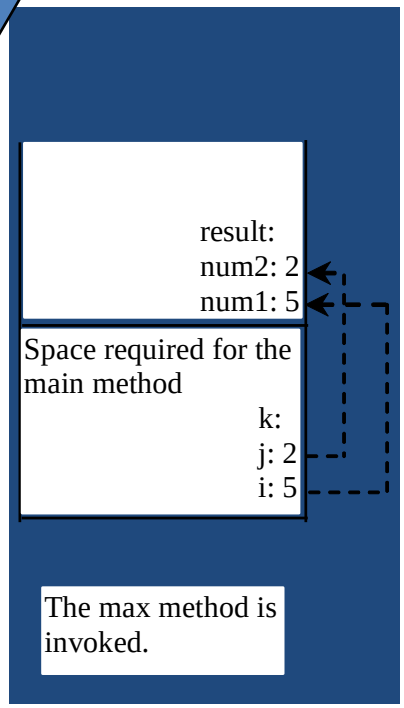


Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

(num1 > num2) es true

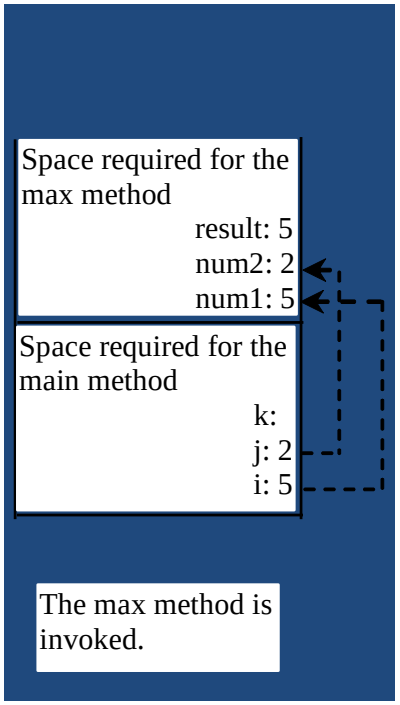


Trace Call Stack

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2)
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```

Se asigna a result el valor de num1

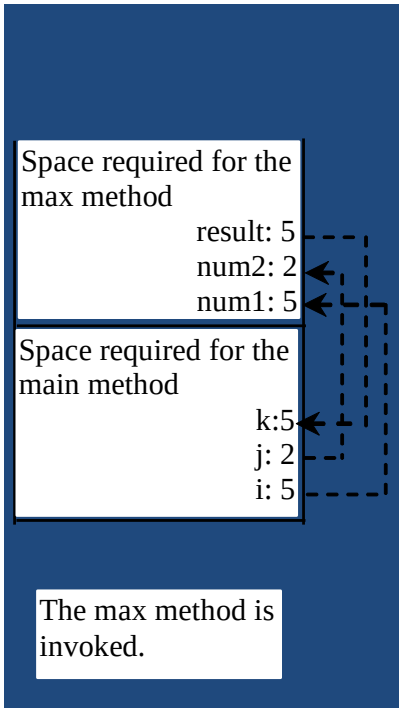


Trace Call Stack

Se devuelve la variable result, y se asigna a la variable k.

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
    "The maximum between " + i +
    " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```



Trace Call Stack

Se ejecuta la impresión de la cadena

```
public static void main(String[] args) {
  int i = 5;
  int j = 2;
  int k = max(i, j);

  System.out.println(
   "The maximum between " + i +
   " and " + j + " is " + k);
}
```

```
public static int max(int num1, int num2) {
  int result;

  if (num1 > num2)
    result = num1;
  else
    result = num2;

  return result;
}
```