Capítulo 5

Funciones

Problema de apertura

Encuentre la suma de enteros de 1 a 10, de 20 a 30 y de 35 a 45, respectivamente.

```
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println(" Suma de 1 a 10 es " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
  sum += i;
System.out.println(" Suma de 20 a 30 es " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
  sum += i;
System.out.println("Suma de 35 a 45 es " + sum);
```

Problema

```
int sum = 0;
for (int i = 1; i <= 10; i++)
  sum += i;
System.out.println("Suma de 1 a 10 es " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
  sum += i;
System.out.println("Suma de 20 a 30 es " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
  sum += i;
System.out.println("Suma de 35 a 45 es" + sum);
```

Solución

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

```
public static void main(String[] args) {
  System.out.println("Sum from 1 to 10 is " + sum(1, 10));
  System.out.println("Sum from 20 to 30 is " + sum(20, 30));
  System.out.println("Sum from 35 to 45 is " + sum(35, 45));
}
```

Objetivos

- Definir métodos con parámetros formales.
- Invocar métodos con parámetros reales (es decir, argumentos).
- Definir métodos con un valor de retorno .
- Definir métodos sin un valor de retorno .
- Pasar argumentos por valor.
- Desarrollar código reutilizable que sea modular, fácil de leer, fácil de depurar y fácil de mantener .
- Usar la sobrecarga de métodos y comprender la sobrecarga ambigua.
- Determinar el alcance de las variables.
- Aplicar el concepto de abstracción de método en el desarrollo de software.
- Diseñar e implementar métodos utilizando el refinamiento escalonado.
- Funciones recursivas.

Definición de Funciones

Las funciones son fragmentos de código reutilizables a los que se les da nombre y pueden ser invocados tantas veces como se quiera desde otros puntos del programa. Cada vez que se invoca a una función se ejecutan todas las instrucciones que contiene.

Son muy adecuadas para contener secuencias de código que tienen un significado propio o que realizan una tarea específica, de forma que podemos dividir el programa en funciones de una forma más racional y fácil de mantener.

En Java, las funciones se definen como métodos estáticos, es decir, métodos que pertenecen a una clase. En Java todo el código debe estar contenido en una o varias clases. Para ello se usa la palabra clave static.

Sintaxis Definición de Funciones

La sintaxis de la definición de una función en java es

public static tipo_devuelto nombre_función(tipo1 parámetro1, tipo2 parámetro2, ...) {
bloque_de_instricciones;
}

La palabra reservada ***public*** (que es opcional) se usa para que la función sea accesible desde cualquier clase.
La palabra reservada ***static*** indica que la función (o método) pertenece a la clase y no a sus objetos.
El ***tipo_devuelto*** puede ser cualquier tipo válido en Java e indica el tipo del valor devuelto por la función. Además también puede ser void, que indica que la función no devuelve nada.

Sintaxis llamada a Funciones

La llamada a una función tiene la sintaxis

nombre_función(expresión1, expresión2,..)

y puede aparecer en una expresión (si su tipo no es void) o como una sentencia en sí misma.

Si la llamada se usa dentro de una expresión, será sustituida en ésta por el valor devuelto.

Si la llamada se usa como una sentencia, si devuelve un valor, éste se pierde.

Ejemplo de Métodos

Funciones en Java son Métodos estáticos.

Vemos el ejemplo del cálculo del máximo de dos enteros.

Definir un método

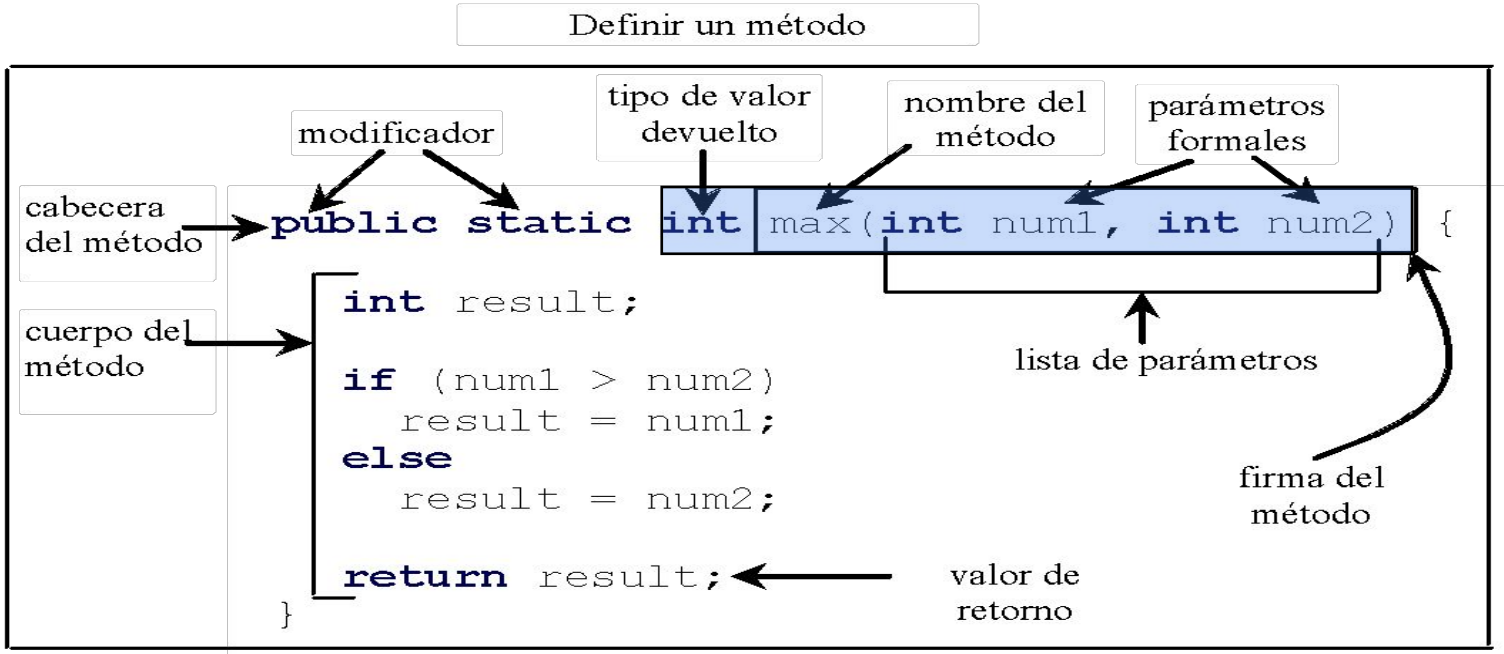
```
public static int max(int num1, int num2) {

   int result;

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

   return result;
 }
```

Invocar un método

```
int z = max(x, y);
```

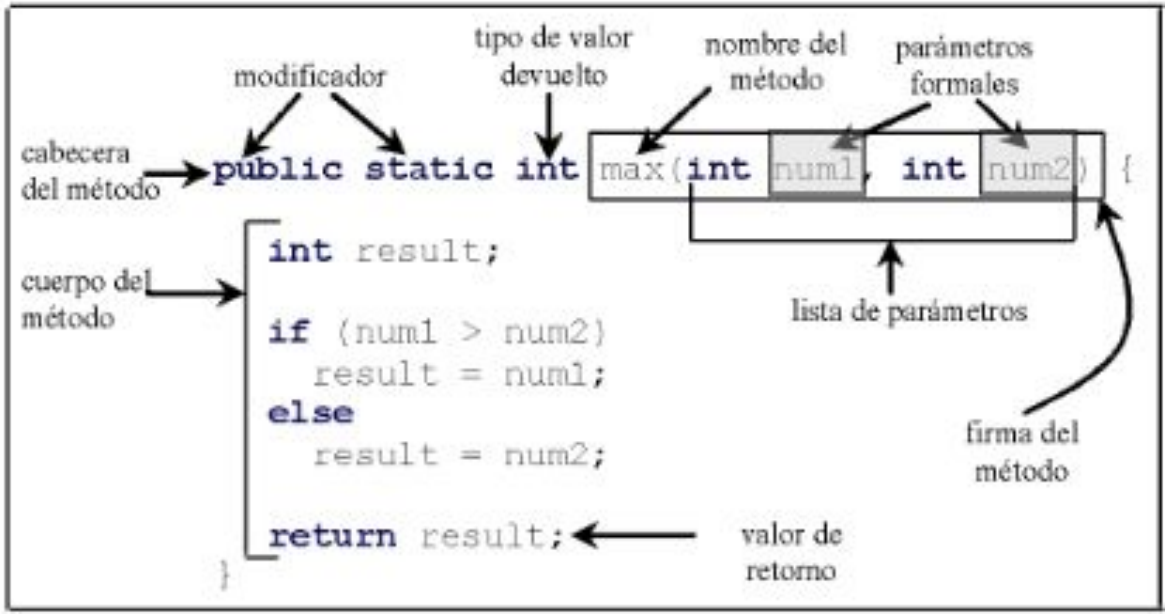
Definición del Método

La firma del método es la combinación del nombre del método y la lista de parámetros.



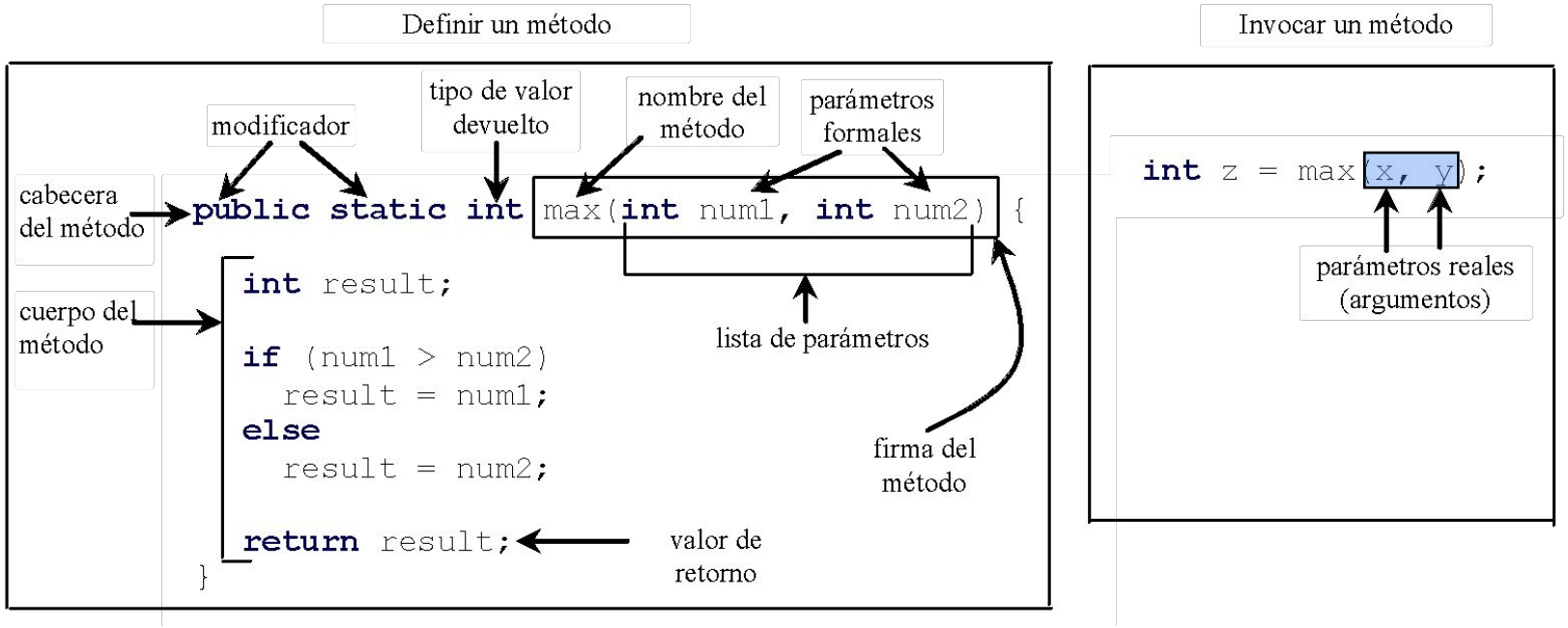
Parámetros formales

Las variables definidas en el encabezado del método se conocen como parámetros formales. La lista de parámetros, que aparece entre paréntesis, puede contener cualquier número de parámetros, que a su vez pueden ser de cualquier tipo válido. También puede estar vacía, en cuyo caso se ponen los paréntesis sin nada dentro.



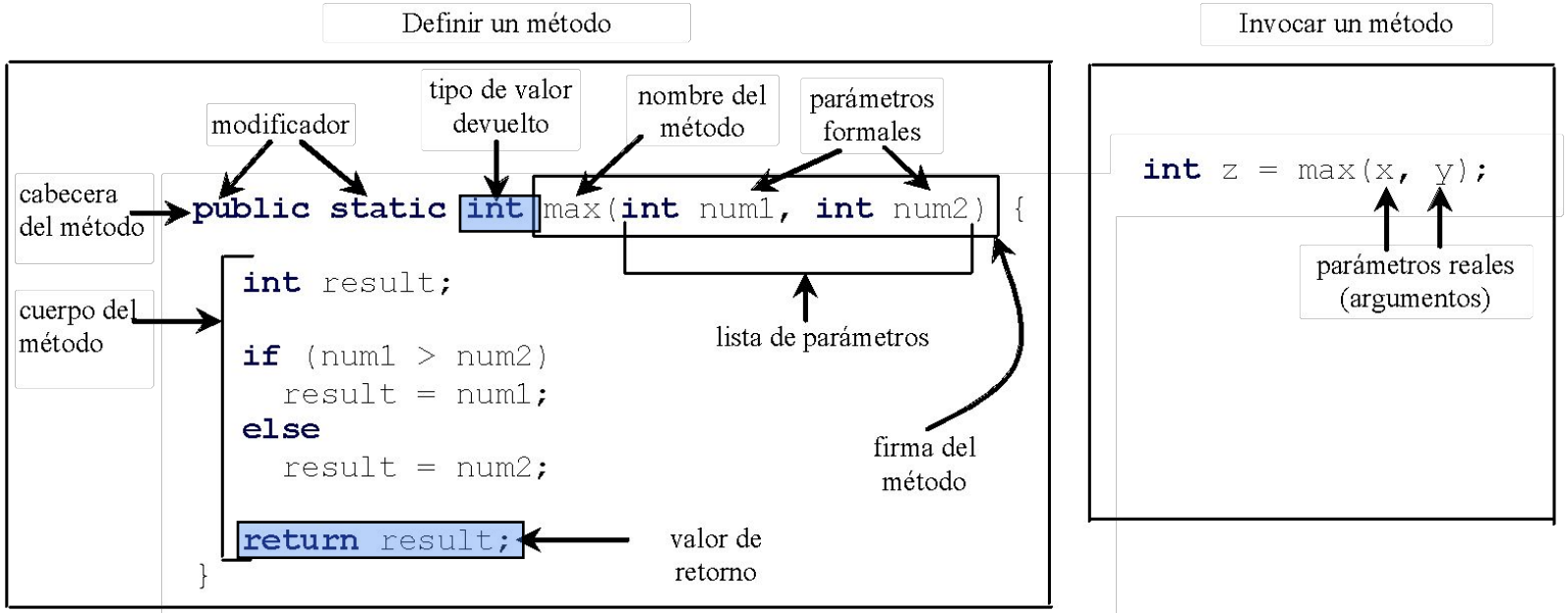
Parámetros reales

Cuando se invoca un método, pasa un valor al parámetro. Este valor se conoce como parámetro o argumento real.



Tipo de valor devuelto

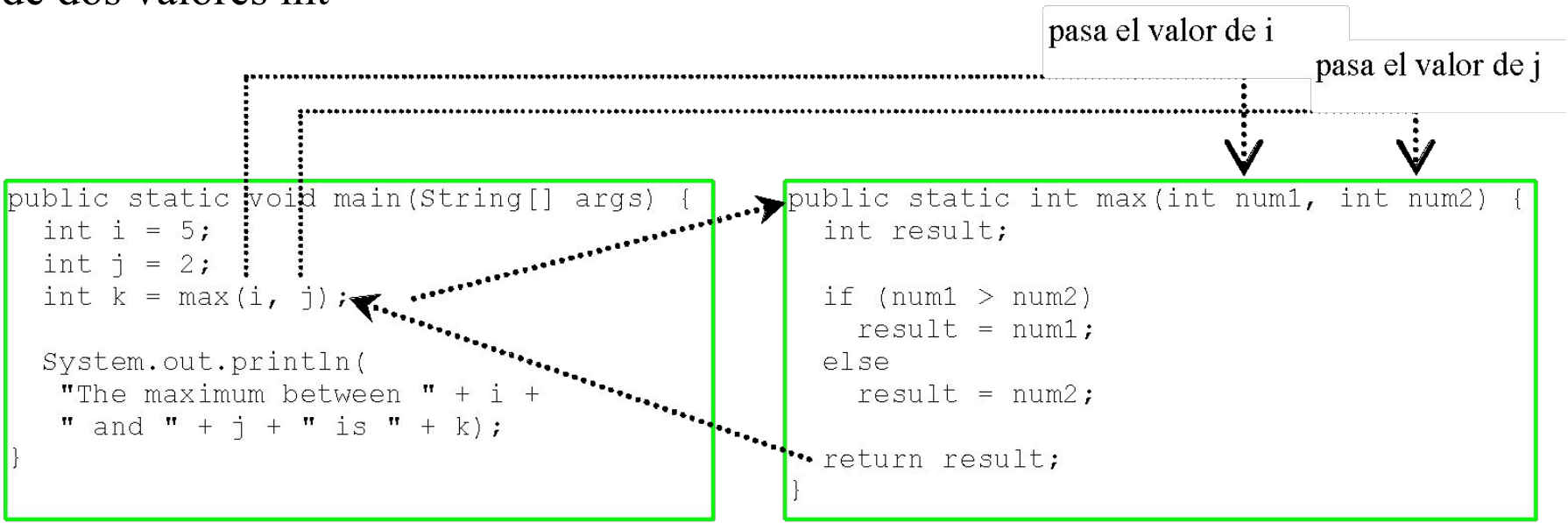
Un método puede devolver un valor. TipoValorDevuelto es el tipo de datos del valor que devuelve el método. Si el método no devuelve un valor, TipoValorDevuelto es la palabra clave void. Por ejemplo, TipoValorDevuelto en el método main es void.



Ejemplo Llamando a Métodos

Probando el método max

Este programa muestra cómo llamar a un método max para devolver el mayor de dos valores int



Seguimiento de invocación de método

i es ahora 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 Method Invocation

j is now 2

```
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;
}
```

Seguimiento de invocación de método


```
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;
}
```

Seguimiento de invocación de método

invoca max(i, j)
Pasa el valor de i a num1
Pasa el valor 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;
}
```

Seguimiento de invocación de método

Declara la variable result

```
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;
}
```

Seguimiento de invocación de método

(num1 > num2) es cierto, ya que num1 es 5 y num2 es 2

```
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;
}
```

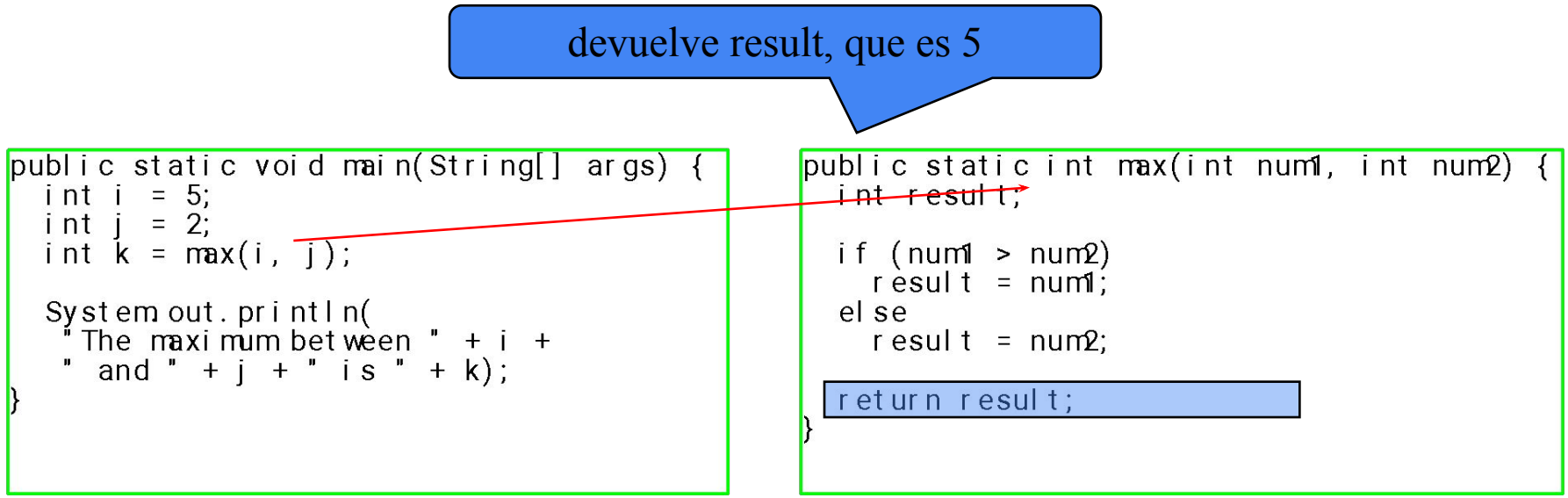
Seguimiento de invocación de método

result es ahora 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;
}
```

Seguimiento de invocación de método



Seguimiento de invocación de método

devuelve max(i, j) y asigna el valor de retorno a 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;
}
```

Seguimiento de invocación de método

Ejecuta la instrucción de impresión

```
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;
}
```

ATENCIÓN

Un método de devolución de valor requiere de una sentencia de devolución. El método que se muestra a continuación en (a) es lógicamente correcto, pero tiene un error de compilación porque el compilador de Java cree que es posible que este método no devuelva ningún valor.

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else if (n < 0)
    return -1;
}
```

(a)

Debiera ser

```
public static int sign(int n) {
  if (n > 0)
    return 1;
  else if (n == 0)
    return 0;
  else
    return -1;
}
```

(b)

Para solucionar este problema, eliminar *if (n < 0)* en (a), de modo que el compilador verá una declaración de devolución que se alcanzará independientemente de cómo se evalúe la sentencia if.

Paso de parámetros

El tipo de método void no devuelve un valor. El método realiza algunas acciones, pero no devuelve ningún valor.

```
public static void nPrintln(String message, int n) {
  for (int i = 0; i < n; i++)
    System.out.println(message);
}
```

Supongamos que invocas el método usando nPrintln(“Welcome to Java”, 5);
¿Cuál es el resultado?

Supongamos que invocas el método usando nPrintln(“Computer Science”, 15);
¿Cuál es el resultado?

¿ Puedes invocar el método usando nPrintln(15, “Computer Science”); ?

Paso por valor y por referencia

Cuando se pasa un parámetro por valor, se pasa una copia de la variable, únicamente importa el valor. Cualquier modificación que se haga a la variable que se pasa como parámetro dentro de la función no tendrá ningún efecto fuera de la misma.

Cuando se pasa un parámetro por referencia, por el contrario, si se modifica su valor dentro de la función, los cambios se mantienen una vez que la función ha terminado de ejecutarse.

En Java, todos los parámetros que son de tipo *int, double, float, char o String* se pasan siempre por valor mientras que los *arrays* se pasan siempre por referencia.

Modularización de Código

Los métodos se pueden usar para reducir la codificación redundante y permitir la reutilización del código. Los métodos también se pueden usar para modularizar código y mejorar la calidad del programa.

Métodos de reutilización de otras clases
NOTA: Uno de los beneficios de los métodos es la reutilización. El método max se puede invocar desde cualquier clase además de desde TestMax. Si crea una nueva clase Test, puede invocar el método max usando NombreClase.NombreMetodo (e.g., TestMax.max).

La sobrecarga de métodos

La sobrecarga de métodos es una técnica de la orientación a objetos que permite definir diferentes versiones de un método, todos con el mismo nombre pero con diferentes listas de parámetros.
El compilador de Java sabrá cuál es el que se quiere utilizar por el número y/o tipo de parámetros que se le pasen, y buscará la versión del método con la lista de parámetros correcta.
Para sobrecargar un método, sólo hay que definirlo más de una vez, especificando una nueva lista de parámetros en cada una de ellas.

Sobrecarga del método `max`

```
public static double max(double num1, double num2) {
  if (num1 > num2)
    return num1;
  else
    return num2;
}
```

Invocación ambigua

A veces puede haber dos o más coincidencias posibles para la invocación de un método, y el compilador no puede determinar la coincidencia más específica. Esto se conoce como invocación ambigua. La invocación ambigua es un error de compilación.

```
public class AmbiguousOverloading {
  public static void main(String[] args) {
    System.out.println(max(1, 2));
  }

  public static double max(int num1, double num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }

  public static double max(double num1, int num2) {
    if (num1 > num2)
      return num1;
    else
      return num2;
  }
}
```

Alcance de las variables locales

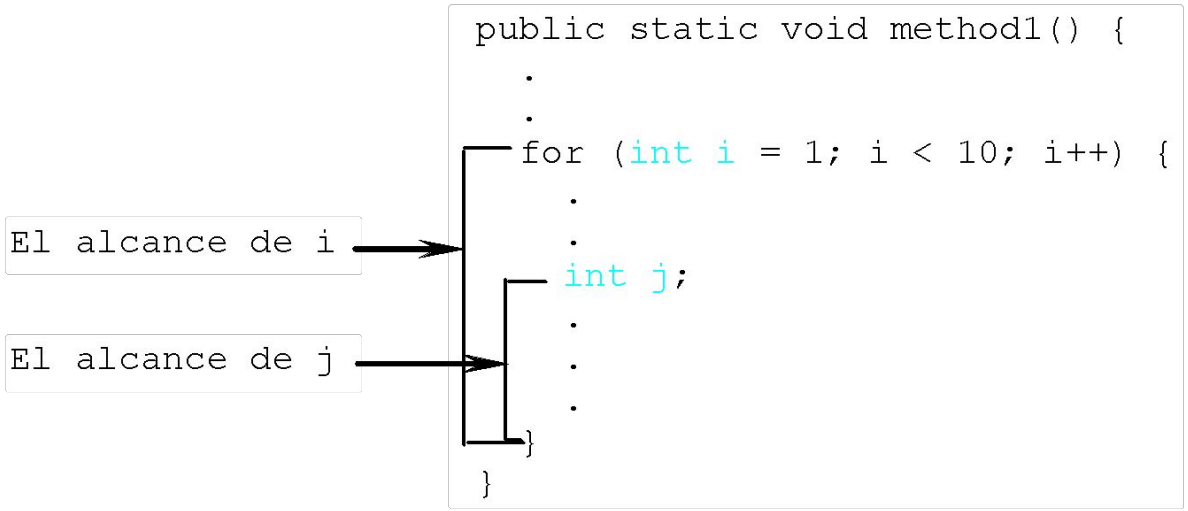
Una variable local: una variable definida dentro de un método.

Alcance: la parte del programa donde se puede hacer referencia a la variable.
El alcance de una variable local comienza desde su declaración y continúa hasta el final del bloque que contiene la variable. De hecho se pueden definir variables en ámbitos aún más locales, en los bloques, por ejemplo, el bloque de una sentencia condicional o el de un bucle.

Una variable local debe declararse antes de que pueda ser utilizada. Podemos declarar una variable local con el mismo nombre varias veces, en diferentes bloques que no anidan en un método, pero no puede declarar una variable local dos veces en bloques anidados.

Alcance de las variables locales.

Una variable declarada en la parte de acción inicial de un encabezado de bucle for tiene su alcance en todo el ciclo. Pero una variable declarada dentro de un cuerpo de bucle for tiene su alcance limitado en el cuerpo del bucle desde su declaración y hasta el final del bloque que contiene la variable.



Alcance de las variables locales.

Está bien declarar i en dos bloques que no anidan

```
public static void method1() {
  int x = 1;
  int y = 1;

  for (int i = 1; i < 10; i++) {
    x += i;
  }

  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

Es incorrecto declarar i en dos bloques anidados

```
public static void method2() {

  int i = 1;
  int sum = 0;

  for (int i = 1; i < 10; i++)
    sum += i;
  }

}
```

Alcance de las variables locales.

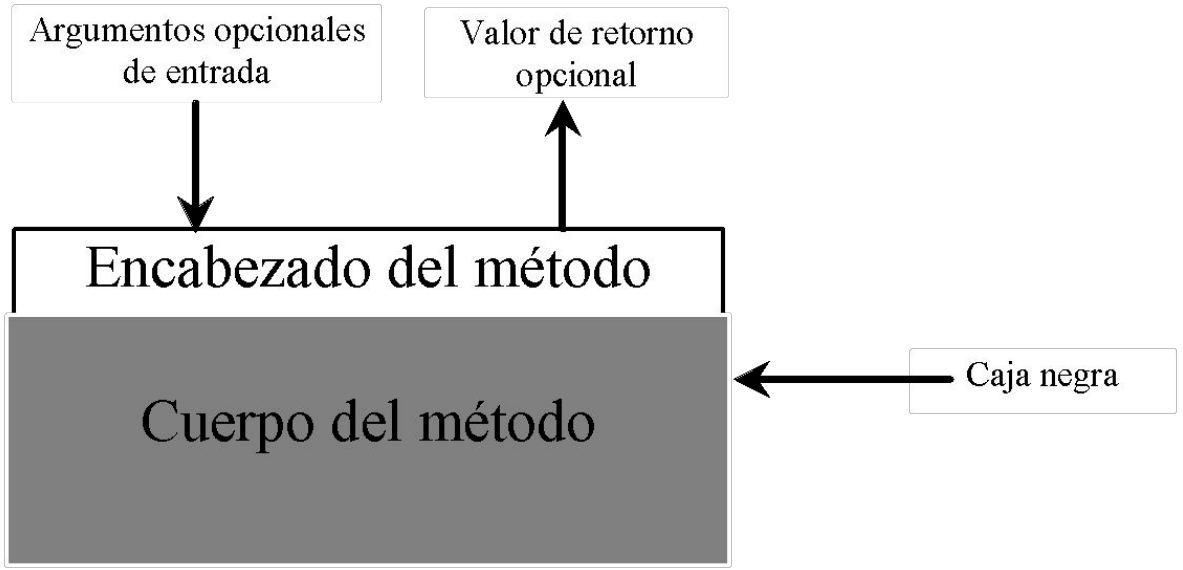
```
// Correcto sin errores
public static void correctMethod() {
  int x = 1;
  int y = 1;
  // i es declarada
  for (int i = 1; i < 10; i++) {
    x += i;
  }
  // i es declarada de nuevo
  for (int i = 1; i < 10; i++) {
    y += i;
  }
}
```

Alcance de las variables locales.

```
// Con errores
public static void incorrectMethod() {
  int x = 1;
  int y = 1;
  for (int i = 1; i < 10; i++) {
    int x = 0;
    x += i;
  }
}
```

Abstracción del método

Puedes pensar en el cuerpo del método como una caja negra que contiene la implementación detallada del método.



Beneficios de los métodos

- Escribe un método una vez y vuelve a utilizarlo en cualquier lugar..
- Ocultación de información. Ocultar la implementación a el usuario.
- Reducir la complejidad.

Caso práctico: Generando caracteres aleatorios.

Ahora consideremos cómo generar una letra minúscula al azar. El Unicode para letras minúsculas son enteros consecutivos comenzando desde el Unicode para 'a', después para 'b', 'c', ..., y 'z'. El Unicode para 'a' es (int)'a'

Entonces, un entero aleatorio entre (int) 'a' y (int) 'z' es

```
(int)((int)'a' + Math.random() * ((int)'z' - (int)'a' + 1)
```

Refinamiento por pasos (opcional)

El concepto de abstracción de métodos se puede aplicar al proceso de desarrollo de programas. Al escribir un programa grande, puede usar la estrategia "divide y vencerás", también conocida como refinamiento gradual, para descomponerla en sub-problemas. Los sub-problemas se pueden descomponer aún más en problemas más pequeños y más manejables.

Beneficios del refinamiento paso a paso

- Programas sencillos
- Reutilizando métodos
- Desarrollo, depuración y prueba más sencillos
- Facilita el trabajo en equipo

Funciones recursivas

Existe un tipo particular de funciones que son las funciones recursivas. Son funciones que se llaman a sí mismas, es decir, que durante la ejecución de sus instrucciones realizan (condicionalmente) una (o más) llamadas a la propia función.

Cada vez que se realiza una llamada se crea un nuevo contexto de ejecución de la función, con sus propios parámetros y sus propias variables locales, que no interfieren con el contexto anterior.

Dicha llamada debe ser condicional porque una característica imprescindible es que haya un caso de salida en el que no se realiza la llamada recursiva, ya que de lo contrario, la recursividad se volvería infinita.

La ejecución de las funciones recursivas no es en general la más óptima, pero existen determinados algoritmos que tienen una solución sencilla de forma recursiva.

Ejercicios

EJERCICIO1: Conversiones entre pies y metros.

Escribe una clase que contenga los siguientes dos métodos:

/** Convert from feet to meters */
public static double piesAMetros(double pies)

/** Convert from meters to feet */
public static double metrosAPies(double metros)

La fórmula para la conversión es:

metros = 0.305 * pies

pies = 3,279 * metros

Escriba un programa que llame a estos métodos para mostrar la siguiente tabla:

Pies	Metros		Metros	Pies
1.0	0.305	|	20.0	65.574
2.0	0.610	|	25.0	81.967
...				
9.0	2.745	|	60.0	196.721
10.0	3.050	|	65.0	213.115

Ejercicios

EJERCICIO2: Escriba un programa que llame a las funciones que creas convenientes para calcular la suma y multiplicación de los n primeros números impares y de los n primeros números pares. Y muestre los mensajes con los valores devueltos.

EJERCICIO 3: Realiza un programa que pida al usuario dos enteros, estos enteros se pasan como parámetros a una función que muestra todos los números comprendidos entre ellos, inclusive.

EJERCICIO 4: Calcular el factorial de n un número entero positivo introducido por el usuario de forma iterativa y recursivamente.

Nota: Recordar que por definición el factorial de 0(0!) es 1.