UD5. Métodos

Módulo: Programación

Unión Europea
Fondo Social Europeo
El FSE invierte en tu futuro

Germán Gascón Grau
g.gascongrau@edu.gva.es

Contenidos

- ¿Qué es un método?
- ¿Cómo crear un método?
- ¿Como invocar a un método?
- Paso de argumentos
- Métodos sobrecargados
- Ámbito de definición de métodos

¿Qué es un método?

- Conjunto de instrucciones agrupadas bajo un identificador.
- Puede ser invocado desde diferentes puntos de un programa.
- Opcionalmente puede devolver un valor. Tradicionalmente (programación estructurada):
  - Los métodos que devuelven un valor se denominan **funciones**
  - Los métodos que no devuelven ningún valor se denominan **procedimientos**

¿Qué es un método?

- Hasta ahora hemos utilizado algunos métodos definidos en las librerías propias de Java. Por ejemplo:
  - `int i = entrada.nextInt();`
  - `double rx = Math.sqrt(78);`
  - `System.out.println("Hola a todos");`
- Podemos observar que:
  - Todos los métodos tienen un identificador (nombre): `sqrt`, `nextInt`, `println`
  - Después del identificador, y entre paréntesis, aparecen los parámetros (`78`), (`"Hola a todos"`).
  - Pueden no tener parámetros.
  - Algunos métodos devuelven un resultado (`nextInt, sqrt`), otros métodos no explícitamente (`println`).

¿Qué es un método?

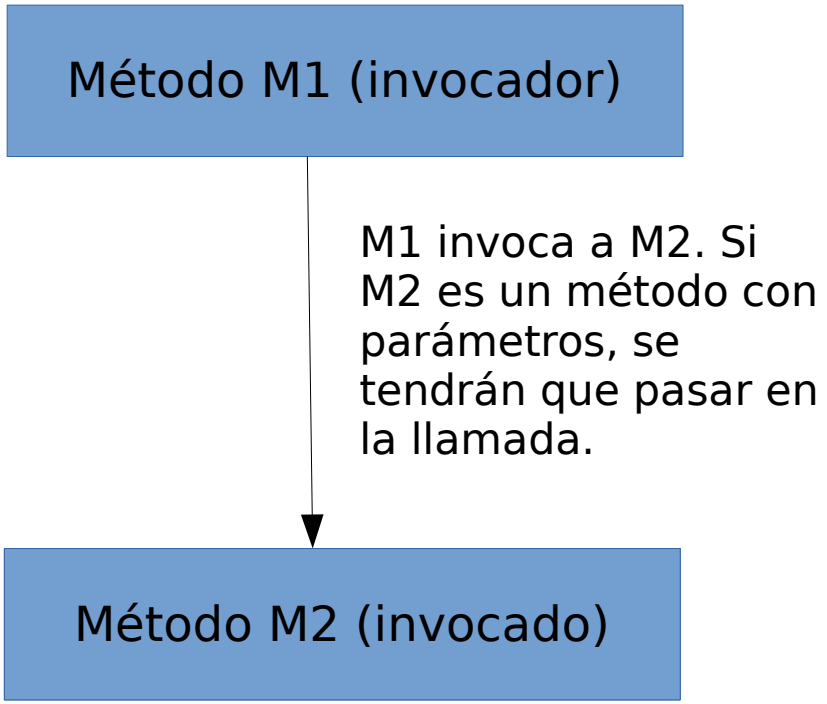
- El programador también puede definir sus métodos propios.
- Ventajas:
  - Ahorra **esfuerzo** y **tiempo** cuando en la resolución de un problema se repite con frecuencia una misma secuencia de acciones: reutilización de código.
  - Facilita la **resolución de problemas grandes** a través de la descomposición en problemas más sencillos.
  - **Incrementa la legibilidad** de los programas.

¿Qué es un método?

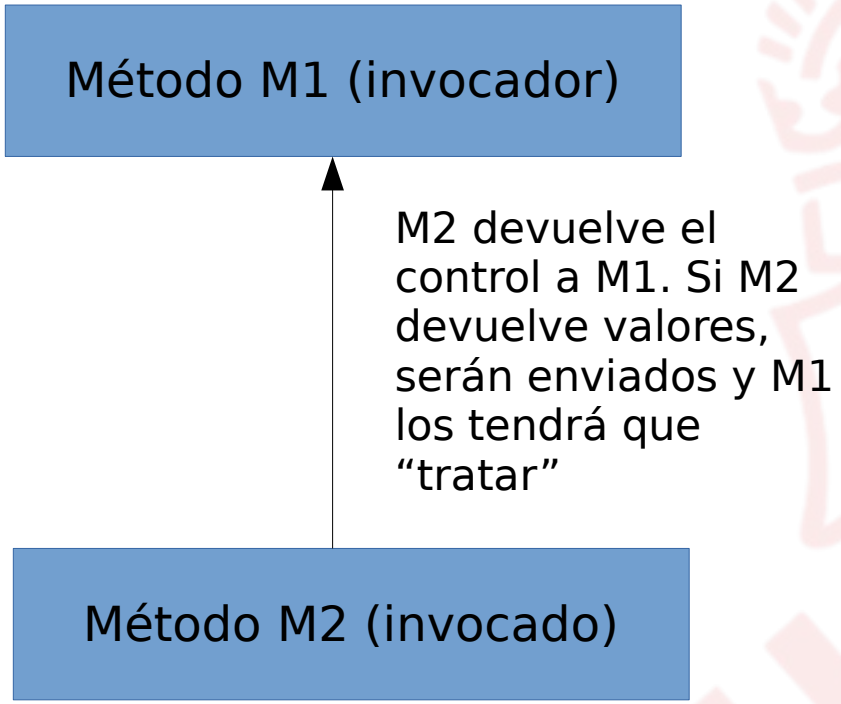
- Un método puede ser invocado (llamado) desde otro método:
  - Cuando un método(M1) llama a otro método(M2), el método invocador(M1) transfiere el control al método invocado(M2).
  - Cuando el método invocado(M2) finaliza la ejecución de su código, devuelve el control al método invocador(M1).
- En Java:
  - Un programa empieza a ejecutarse siempre por el método `main()`.
  - El método `main()` puede invocar a otros métodos que, a su vez, pueden invocar a otros.
- Un método también se puede invocarse así mismo: **recursividad**.

¿Qué es un método?

LLAMADA A UN MÉTODO



RETORNO DE UN MÉTODO



¿Cómo crear un método?

```
[static] tipo_devuelto|void nombre([tipo_par1 par1, tipo_par2 par2, …]) {
        //Instrucciones del método
        //si devuelve algún valor, se tiene que incluir la instrucción return

}
```

```
public static int suma(int a, int b) {
    return a + b;
}
```

¿Cómo crear un método?

```
[static] tipo_devuelto|void nombre([tipo_par1 par1, tipo_par2 par2, …]) {
        //Instrucciones del método
        //si devuelve algún valor, se tiene que incluir la instrucción return

}
```

tipo_devuelto

tipo de datos que devuelve el método

si no devuelve nada, se tiene que indicar con **void**

parámetros (argumentos)

son opcionales

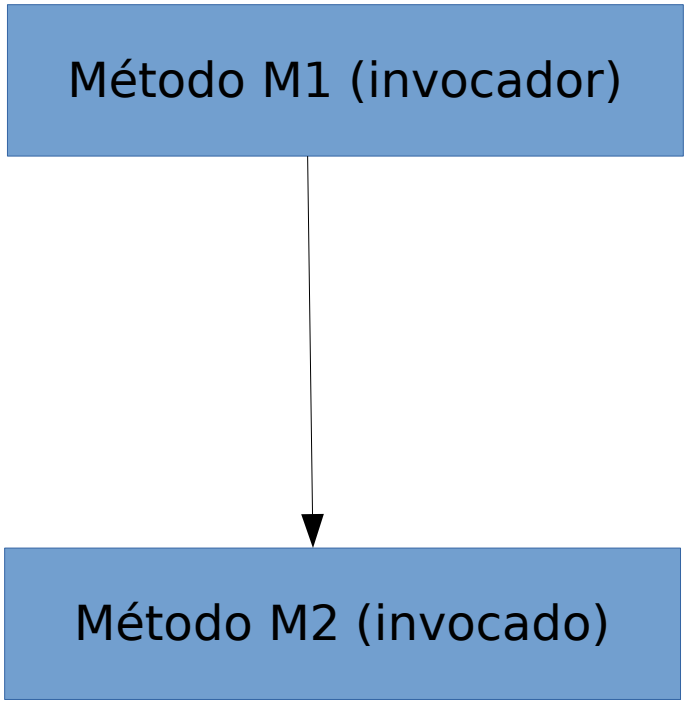
si aparecen, se tiene que indicar, para cada uno de ellos, su tipo y su nombre

¿Cómo crear un método?

- Instrucción **return**
  - especifica el valor que devuelve el método
  - devuelve el control inmediatamente al método invocador

```
public static int maximo(int x, int y) {
    if (x >= y)
        return x;
    else
        return y;
}
```

¿Cómo invocar a un método?



En M1, cuando se hace la llamada:

M2 (par1, par2, …, parN)

Parámetros actuales: contienen los valores con los cuales M1 llama a M2.

Tienen que coincidir la cantidad de parámetros y el tipo con:

Parámetros formales: especificados en la cabecera de M2

M2 (tipo_par1 par1, tipo_par2 par2, …, tipo_parN parN)

Invocar a un método. Ejemplo

```
public static int suma(int a, int b) {
    return a + b;
}
```

Parámetros formales

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

    double y = suma(x, x * 67);
    double z = suma(x + y, 45);
}
```

Parámetros actuales

Invocar a un método. Ejemplo

```
public static int maximo(int x, int y) {
    if (x >= y)
        return x;
    else
        return y;
}
```

```
int a,b,c,d;
int max1,max2,max;

//Asignación de valores a las variables
max1=maximo(a,b);
max2=maximo(c,d);
max=maximo(max1,max2);
```

Invocar a un método. Ejemplo

Cabecera del método	Llamada al método
int suma(**int a, int b**) /* a y b son parámetros **formales** */	suma(**2 , 4**); /* 2 y 4 parámetros **actuales** */
int suma(**int a, int b**) /* parámetros **formales** */	suma(**num1, 3 + num2**); /* parámetros **actuales** */
void imprime(**int a, float b, char c**) /* parámetros **formales** */	imprime(numero, 3.14 , 'x'); /* parámetros **actuales** */

Invocar a un método. Ejemplo

```
System.out.print("El resultado es " + suma(a, 74));
```

```
if (suma(a,b) < 20)
        ...
```

```
imprime('a', 40);
```

Paso de parámetros

- En Java, toda la información que se le pasa a un método cuando se invoca se pasa por **valor**. Pero se tiene que distinguir si los parámetros son de tipos **primitivos** o de tipos **referencia**.
- Cuando los parámetros formales son de tipos **primitivos** (tipos básicos), al método le llega un **valor** (parámetro actual) que se guardará dentro de una variable. Este valor es independiente de la variable que se envió en la llamada.
- Los parámetros de tipos `byte`, `short`, `int`, `long`, `float`, `double`, `boolean` y `char` **nunca** se modifican en el método invocador, aunque sus copias varían en el método invocado.
- Cuando los argumentos son de tipos **referencia** (array, objeto...), al método le llega una **dirección de memoria** (referencia). El método no podrá modificar la referencia, pero si los valores que hay en la dirección de memoria. (*)

  (*): lo veremos con más detalle en la orientación a objetos.

Paso de parámetros

```
public class ValorApp {
    public static void main(String[] args) {
        int a = 3;
        System.out.println("antes de la llamada: a = " + a);
        funcion(a);
        System.out.println("después de la llamada: a = " + a);
    }

    public static void funcion(int x){
        x = x + 3;
        System.out.println("dentro de la función: a = " + x);
    }
}
```

Métodos sobrecargados

- Cuando dos métodos tienen el mismo nombre, pero son diferentes en la cantidad, orden o tipo de los parámetros, se dice que están **sobrecargados** (overloaded en inglés)
- No hay suficiente diferencia con el tipo de valor devuelto o en las excepciones (errores a controlar) que se puedan lanzar.

```
public static int suma(int a, int b) {
    return a + b;
}

public static int suma(int a, int b, int c) {
    return a + b + c;
}
```

Ámbito de definición de métodos

- Las **variables** y los **parámetros formales** que se definen dentro de un método son **locales** a él.
- Únicamente son accesibles dentro del método.
- Una clase Java puede definir y emplear sus propios métodos:
  - No hay restricciones en el orden en el que se escriben los métodos.
  - El método main() puede estar antes o después de cualquier otro método.

```
class ... {
public static void main (...) {
...
...
}

//Declaración de métodos
}
```

Ámbito de definición de métodos

- Una clase puede emplear métodos de otra clase.

```
public class App {
    public static void main(String[] args) {
        double resultado = Matematicas.area(4.5);
    }
}
```

```
public class Matematicas {
    public static double area(double radio) {
        return Math.PI * radio * radio;
    }
}
```

Métodos con parámetros variables

- Son métodos que podemos invocar con un número de parámetros variable.
- Para ello debemos apoyarnos del operador tres puntos (…). Al asignar el operados tres puntos a un tipo conseguimos indicar que podrán llegar N parámetros, donde N dependerá de la cantidad de parámetros actuales que indiquemos en la llamada.
- Veamos un ejemplo:

```
public static double suma(double... numeros) {
}
```

- Ejemplos de llamadas válidas:

```
suma(4, 5, 8);
suma(4, 5, 8, 10, 11, 15, 21);
```

- Cuando veamos los Arrays profundizaremos más sobre este tipo de métodos.

Recapitulando

- Nuestros programas van a consistir, a partir de ahora, en una serie de métodos, donde cada uno de ellos va a aportar una pequeña parte de la solución que queremos implementar.
- **¿En qué circunstancias debemos definir métodos?**
  - Cuando detectemos que **tenemos código repetido** en varias partes de nuestro programa.
  - Cuando tengamos fragmentos de código **susceptibles de ser reutilizados**.
  - Los métodos deberían realizar **tareas simples**. Cuando identifiquemos métodos que realizan “demasiadas tareas” es conveniente dividirlos en varios métodos.

Recapitulando

- ¿Cómo definir los métodos?
  - Al definir un método siempre debemos identificar:
    - ¿**qué tarea** necesitamos que realice el método?
    - ¿**qué datos de entrada** necesita?
    - ¿qué **resultado/s** producirá?
- Una vez identificados los requisitos anteriores, puede que se nos ocurran **diferentes variantes de métodos**.
  - La norma es **elegir** siempre aquel método que sea **susceptible** de poder ser **utilizado en más situaciones**, siempre y cuando, el hecho de hacerlo más reutilizable no suponga tener que hacer mucho más código y complicar demasiado el método, lo que produciría el efecto contrario al que queremos conseguir.