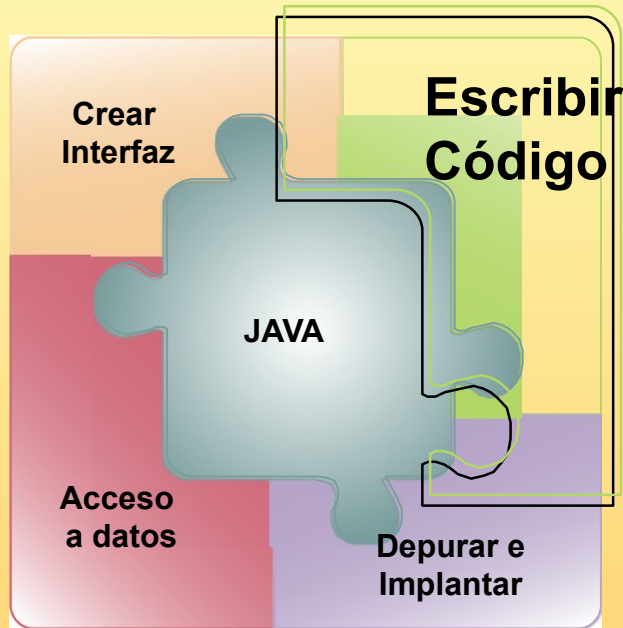


Presentación_Programa ción_Tema3 Métodos

Descripción



- **Qué es un método**
- **Cómo crear un método**
- **Cómo llamar a un método**
- **Paso de argumentos**
- **Métodos sobrecargados**
- **Ámbito de definición de métodos**

Qué es un método

- Un método son un conjunto de instrucciones referenciadas por un identificador
- Puede ser llamado desde diferentes puntos de un programa
- Opcionalmente puede devolver un valor.
Tradicionalmente:

- A los métodos que devuelven un valor se les denominan **funciones**
- A los métodos que no devuelven ningún valor se les llama **procedimientos**

Qué es un método

- **Hasta ahora hemos utilizado ciertos métodos que se definen en librerías del propio lenguaje:**
 - `double rx=Math.sqrt(78);`
 - `int i=entrada.nextInt();`
 - `System.out.println("Hola a todos");`
- **Podemos observar que:**
 - Todos los métodos tienen un identificador: `sqrt`, `nextInt`, `println`
 - Después del identificador, y entre paréntesis, figuran los parámetros del método: `78`, `"Hola a todos"`. Pueden no tener parámetros
 - Algunos métodos devuelven un resultado (`sqrt`), otros métodos no devuelven ningún resultado 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 frecuentemente una misma secuencia de acciones: Reutilización del código
 - Facilita la resolución en términos de sub-problemas más sencillos
 - Incrementa la legibilidad de los programas

Qué es un método

- **Un método puede ser invocado, o *llamado*, desde otro método:**
 - Cuando un método llama a otro método, se transfiere el control al segundo método
 - Cuando finaliza la ejecución del código del segundo método, éste devuelve el control método que lo invocó
- **En JAVA:**
 - Un programa siempre empieza a ejecutarse en el **método 'main'** o principal
 - El método main podrá invocar a otros métodos que, a su vez, pueden invocar a otros métodos
- **Un método puede invocarse a sí mismo, esto se llama recursividad**

Qué es un método

Llamada a una método

Método M1

Método invocador

M1 llama a M2

Opcionalmente se pueden transferir datos
(parámetros)

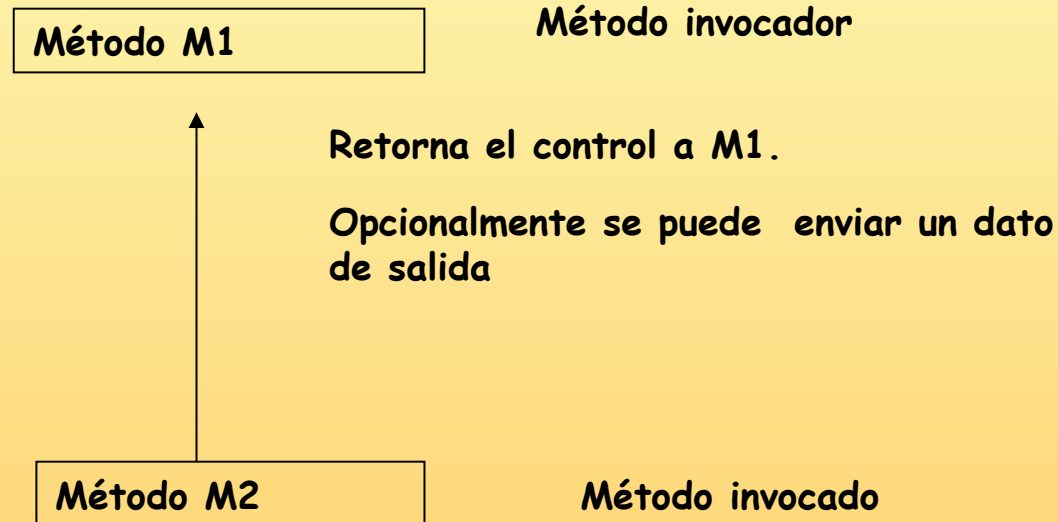
Método M2

Método invocado



Qué es un método

Retorno de un método



Cómo crear un método

```
static tipo_devuelto nombre( [argumentos] )  
{  
    ' Instrucciones de la función,  
    ' incluyendo opcionalmente la instrucción  
    return  
}
```

Ejemplo:

```
static int suma (int a, int b){  
    int r=0;  
    r=a+b;  
    return r;  
}
```

Cómo crear un método

```
static tipo_devuelto nombre( [argumentos ] )  
  
{  
    ' Instrucciones de la función,  
    ' incluyendo opcionalmente la instrucción Return  
}
```

- ***tipo_devuelto:***

Es el tipo de dato del valor que devuelve la función

Si no retorna ningún valor escribimos **void**

- ***argumentos:***

tipo1 variable1, tipo2 variable2,..., tipoN variableN

➤ entre [] significa que es opcional

Cómo crear un método. Valor de retorno

■ La instrucción return

- Especifica el valor que devuelve el método (***valor de retorno***)
- Devuelve el control inmediatamente al método que origina la llamada

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

Cómo llamar a un método

Método invocador

Método M1



Método M2

Método invocado(especificados en la cabecera de M2)

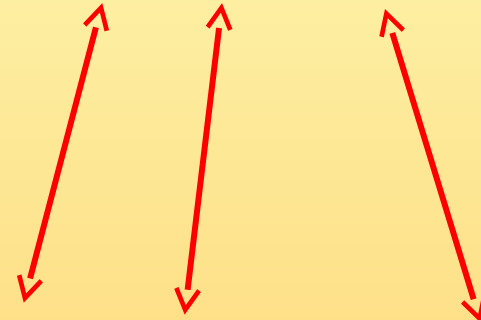
Argumentos actuales:

Son los valores con que M1 llama a M2

Argumentos formales

Han de coincidir en número
y tipo con los

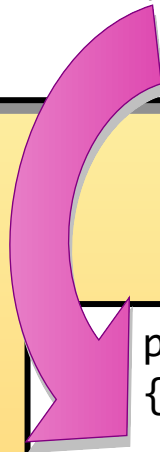
En M1: M2 (arg1, arg2, ..., argn);



M2 (tipo1 var1, tipo2 var2, ..., tipoN varN)

Cómo llamar a un método. Ejemplos

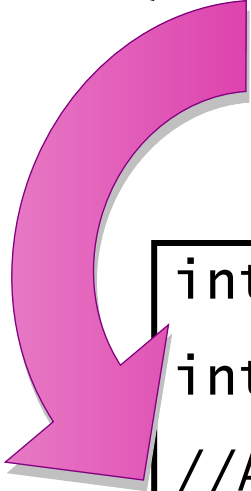
```
static int suma (int a, int b){  
    return a+b;  
}
```



```
public static void main(String[] args)  
{  
    .....  
    y=suma( x , x*67 );  
    z=suma( x+y , 45 );  
}
```

Cómo llamar a un método. Ejemplos

```
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);
```

Cómo llamar a un método. Ejemplos

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

Paso de Argumentos

- En programación podemos pasar argumentos por valor o por referencia
 - **Por valor:** El procedimiento no puede modificar el valor de la variable original
 - **Por referencia:** El procedimiento puede modificar el valor de la variable original
- Para los argumentos de tipos **básicos** (primitivos) en el lenguaje Java, solamente se pasan argumentos por **valor**
 - Los argumentos tipo byte, short, int, long, float, double, boolean, char nunca se modifican en el programa llamante, aunque sus copias varíen en el método llamado

Paso de Argumentos. Ejemplo

```
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=5;  
        System.out.println("dentro de la función: a="+x);  
    }  
}
```

Paso de Argumentos

Mecanismo de paso	Explicación	Implicaciones	Ventaja
Por valor	El procedimiento invocado recibe una copia de los datos cuando es invocado.	Si el procedimiento invocado modifica la copia, el valor original de la variable permanece intacto. Cuando la ejecución retorna al procedimiento de llamada, la variable contiene el mismo valor que tenía antes de que el valor se pasara.	Protege la variable de ser cambiada por el procedimiento invocado.
Por referencia	El procedimiento invocado recibe una referencia a los datos originales (la dirección de los datos en memoria) cuando es invocado.	El procedimiento invocado puede modificar la variable directamente. Cuando la ejecución retorna al procedimiento de llamada, la variable contiene el valor modificado.	El procedimiento invocado puede utilizar el argumento para devolver un nuevo valor al código de llamada.

Métodos Sobrecargados

- Cuando dos métodos se llaman igual pero difieren en el número, orden o tipo de los argumentos de entrada
 - en inglés "*overloaded*"
- No es suficiente diferenciación diferir en el tipo del valor devuelto o en las excepciones que pueden lanzar

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

Ámbito de definición de métodos

- Son objetos locales a un método los parámetros formales y las variables locales que en él se definen
 - Estos sólo son accesibles desde el método en el que se han definido
- Una clase Java puede definir y usar sus propios métodos:

```
class ... {  
    public static void main (...) {  
        ...  
        ...  
    }  
  
    //Declaración de 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

Ámbito de definición de métodos

- Una clase Java puede usar métodos static de otra clase:

```
public class Matematicas {  
    static long factorial(int num){  
        long resultado=1;  
        while(num>0){  
            resultado*=num;  
            num--;  
        }  
        return resultado;  
    }  
    static boolean esPrimo(int numero){  
        if((numero!=2)&&(numero%2==0))  
            for(int i=3; i<numero/2; i+=2){  
                if(numero%i==0){  
                    return false;  
                }  
            }  
        return true;  
    }  
}
```

```
public class MatesApp {  
    public static void main(String[] args) {  
        //Número combinatorio de m sobre n  
        int m=8, n=3;  
  
        long numerador=Matematicas.factorial(m);  
        long denominador=Matematicas.factorial(m-n);  
  
        System.out.println(" vale "+numerador + " / " + denominador);  
        System.out.println("*****");  
  
        //números primos comprendidos entre 100 y 200  
        System.out.println("Números primos comprendidos entre 10  
        for(int num=100; num<200; num++){  
            if(Matematicas.esPrimo(num)){  
                System.out.print(num+" - ");  
            }  
        }  
    }  
}
```