

10

Trabajar con métodos y sobrecarga de métodos

fundacion@proydesa.org) has a
this Student Guide.

Creación y llamada a métodos

Sintaxis:

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

Forma básica de un método

La palabra clave `void` indica que el método no devuelve un valor.

Los paréntesis vacíos indican que no se ha transferido ningún argumento al método.

```
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
```

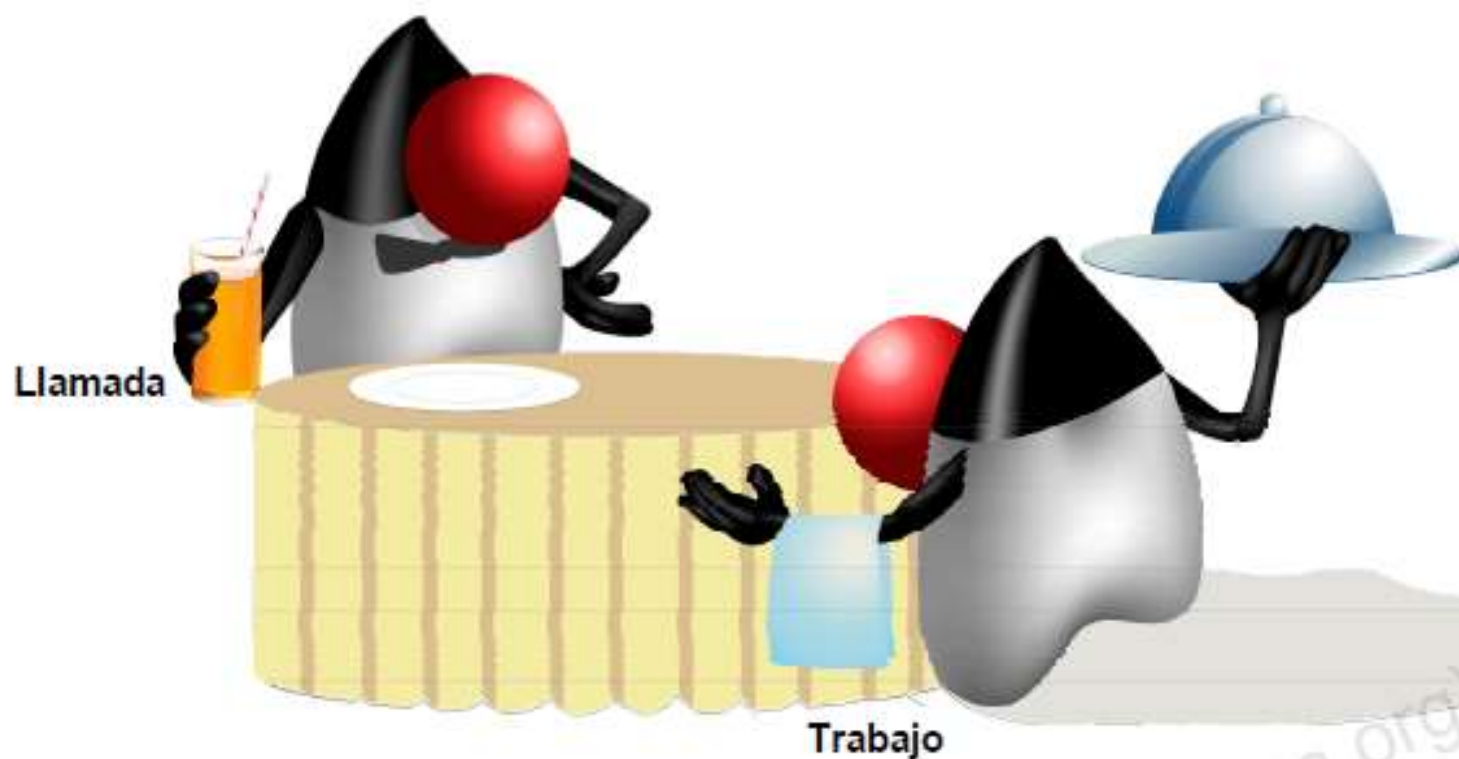
Llamada a un método en una clase diferente

```
public class ShirtTest {  
    public static void main (String[] args) {  
        Shirt myShirt;  
        myShirt = new Shirt();  
        myShirt.display();  
    }  
}
```

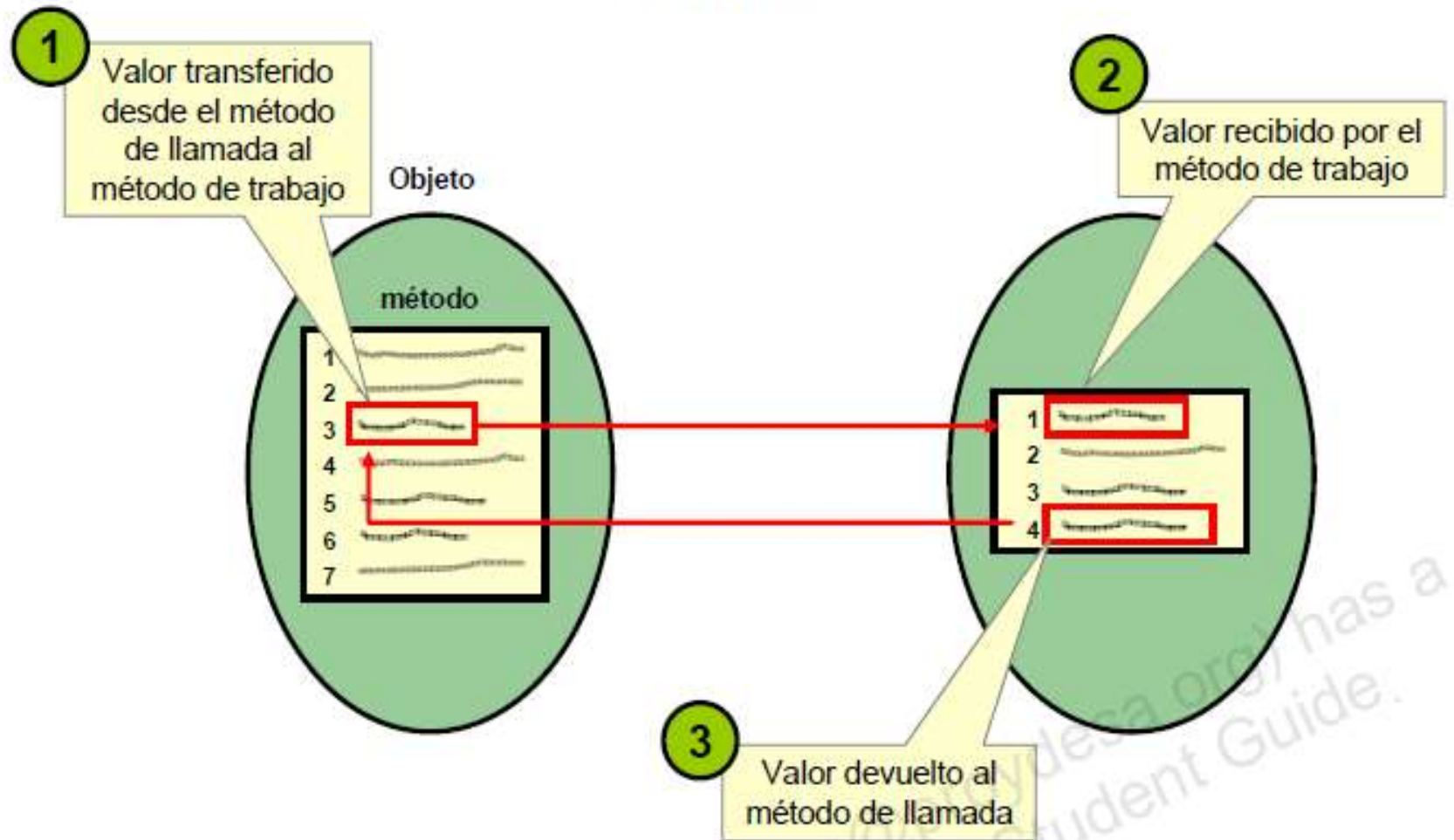
Salida:

```
Item ID: 0  
Item description:-description required-  
Color Code: U  
Item price: 0.0
```

Métodos de llamada y de trabajo



Transferencia de argumentos y devolución de valores



Creación de un método con un parámetro

Llamada:

```
Elevator theElevator = new Elevator();  
  
theElevator.setFloor(4); // Send elevator to the fourth floor
```

Llamada al método
setFloor(), transfiriendo el
valor 4, de tipo int.

Trabajo:

```
public void setFloor(int desiredFloor) {  
    while (currentFloor != desiredFloor){  
        if (currentFloor < desiredFloor){  
            goUp();  
        }  
        else {  
            goDown();  
        }  
    }  
}
```

El método setFloor()
recibe un argumento de
tipo int y le asigna el
nombre desiredFloor.

Creación de un método con un valor de retorno

Llamada:

```
... < lines of code omitted > ...
```

```
boolean isOpen = theElevator.checkDoorStatus() // Is door open?
```

La variable local `isOpen` indica si la puerta del ascensor está abierta.

Trabajo:

```
public class Elevator {  
    public boolean doorOpen=false;  
    public int currentFloor = 1;  
  
    ... < lines of code omitted > ...  
  
    public boolean checkDoorStatus() {  
  
        return doorOpen;  
    }  
}
```

El valor `Elevator` tiene el campo `doorOpen` para indicar el estado de la puerta del ascensor.

El tipo devuelto por el método se define después del nombre del método.

La sentencia `return` devuelve el valor en `doorOpen`.

Llamada a un método en la misma clase

```
public class Elevator {  
  
    public boolean doorOpen=false;  
    public int currentFloor = 1;  
  
    public final int TOP_FLOOR = 5;  
    public final int BOTTOM_FLOOR = 1;  
  
    public void openDoor() {  
  
        // Check if door already open  
        if ( !checkDoorStatus() ) {  
  
            // door opening code  
        }  
    }  
}
```

Se evalúa como true si la puerta está cerrada.

Transferencia de argumentos a métodos

```
public class ShirtTest {  
    public static void main (String[] args) {  
        Shirt myShirt = new Shirt();  
        System.out.println("Shirt color: " + myShirt.colorCode);  
        changeShirtColor(myShirt, 'B');  
        System.out.println("Shirt color: " + myShirt.colorCode);  
    }  
    public static void changeShirtColor(Shirt theShirt, char color) {  
        theShirt.colorCode = color;    }  
}
```

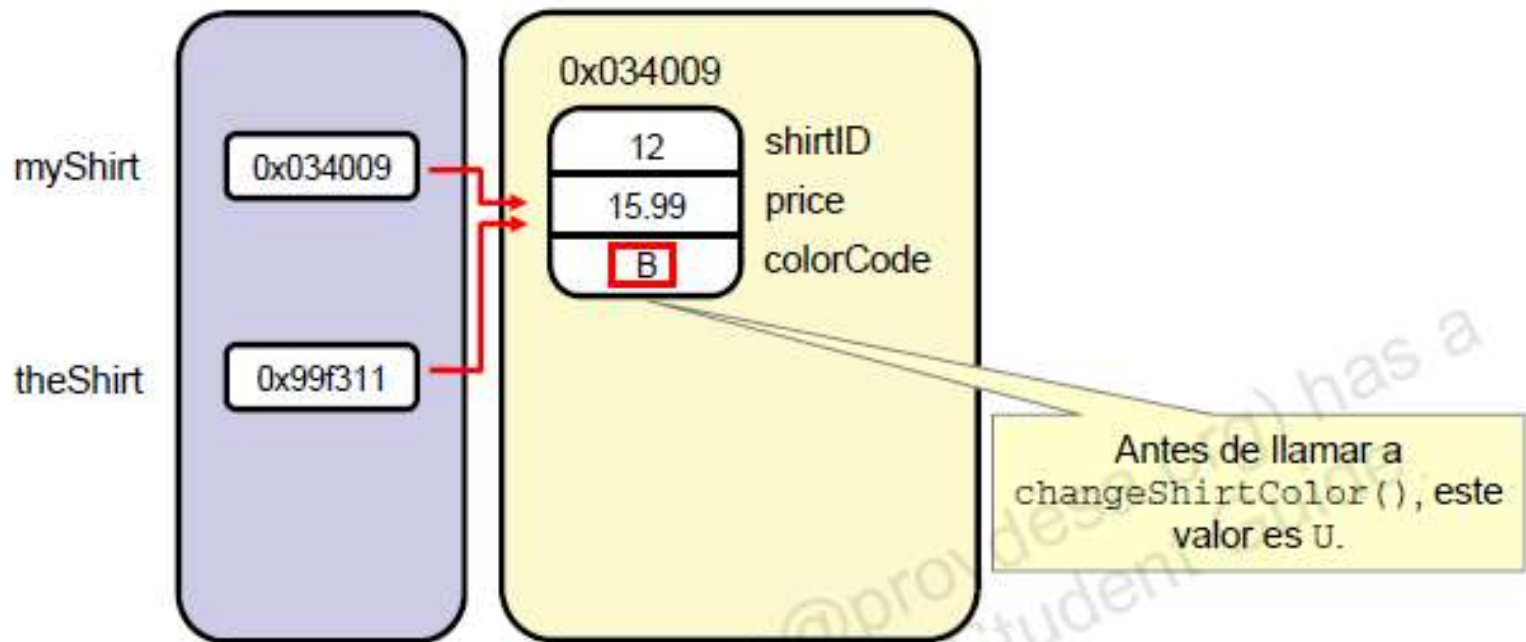
theShirt es una
nueva referencia de
tipo Shirt.

Salida:

```
Shirt color: U  
Shirt color: B
```

Transferencia por valor

```
Shirt myShirt = new Shirt();  
changeShirtColor(myShirt, 'B');
```



Transferencia por valor

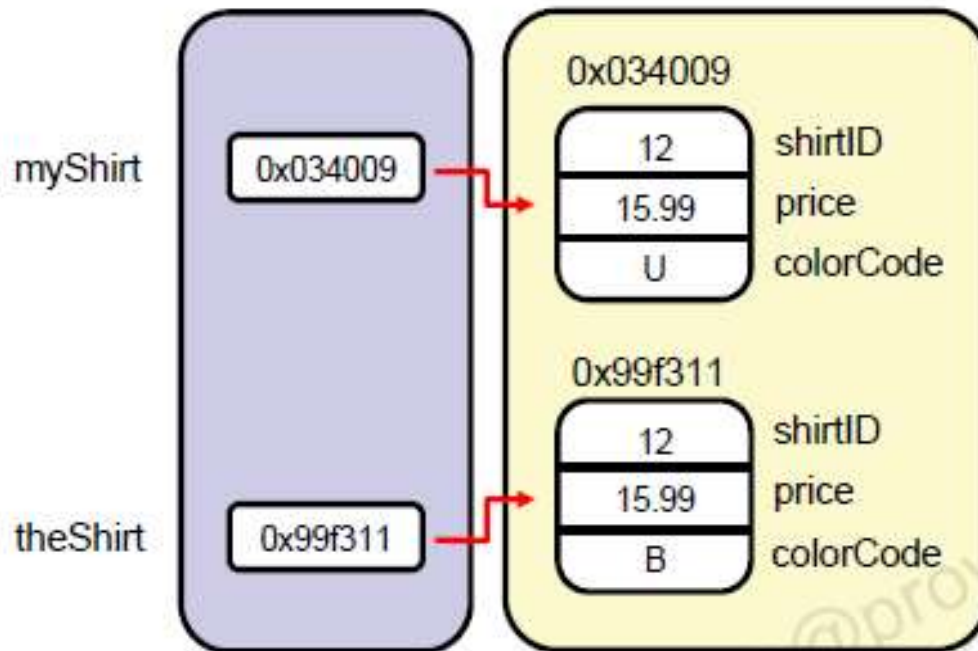
```
public class ShirtTest {  
    public static void main (String[] args) {  
        Shirt myShirt = new Shirt();  
        System.out.println("Shirt color: " + myShirt.colorCode);  
        changeShirtColor(myShirt, 'B');  
        System.out.println("Shirt color: " + myShirt.colorCode);  
    }  
    public static void changeShirtColor(Shirt theShirt, char color) {  
        theShirt = new Shirt();  
        theShirt.colorCode = color;  
    }  
}
```

Salida:

```
Shirt color: U  
Shirt color: U
```

Transferencia por valor

```
Shirt myShirt = new Shirt();  
changeShirtColor(myShirt, 'B');
```



Ventajas del uso de métodos

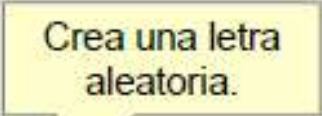
Los métodos:

- Facilitan la lectura y el mantenimiento de los programas
- Aceleran el desarrollo y el mantenimiento
- Son básicos para el software reutilizable
- Permiten la comunicación entre objetos independientes y la distribución del trabajo realizado por el programa

Utilidades matemáticas

```
String name = "Lenny";
String guess = "";
int numTries = 0;

while (!guess.equals(name.toLowerCase())) {
    guess = "";
    while (guess.length() < name.length()) {
        char asciiChar = (char) Math.random() * 26 + 97;
        guess = guess + asciiChar;
    }
    numTries++;
}
System.out.println(name + " found after " + numTries + " tries!");
```



Crea una letra aleatoria.

Métodos estáticos de Math

Observe que el tipo es double y que es estático.

Éste es el método aleatorio.

static double	<code>pow(double a, double b)</code> Returns the value of the first argument raised to the power of the second argument.
static double	<code>random()</code> Returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.
static double	<code>rint(double a)</code> Returns the double value that is closest in value to the argument and is equal to a mathematical integer.
static long	<code>round(double a)</code> Returns the closest long to the argument, with ties rounding up.

Creación de métodos y variables `static`

Los métodos y las variables no locales pueden ser estáticos.

- Pertenecen a la clase, no al objeto.
- Se declaran con la palabra clave `static`:

```
static Properties getProperties()
```

- Para llamar a los métodos `static`:

```
Classname.method();
```

- Para acceder a las variables `static` en otra clase:

```
Classname.attribute_name;
```

- Para acceder a las variables `static` en la misma clase:

```
attribute_name;
```

Creación de métodos y variables static

```
public static char convertShirtSize(int numericalSize) {  
    if (numericalSize < 10) {  
        return 'S';  
    }  
    else if (numericalSize < 14) {  
        return 'M';  
    }  
  
    else if (numericalSize < 18) {  
        return 'L';  
    }  
  
    else {  
        return 'X';  
    }  
}
```

Variables

static

- Declaración de variables `static`:

```
static double salesTAX = 8.25;
```

- Acceso a variables `static`:

```
Classname.variable;
```

- Ejemplo:

```
double myPI;  
myPI = Math.PI;
```

Métodos estáticos y variables estáticas en la API de Java

Ejemplos

- Algunas de las funcionalidades de la clase `Math` son:
 - Exponencial
 - Logarítmica
 - Trigonométrica
 - Aleatoria
 - Acceso a las constantes matemáticas comunes, como el valor pi (`Math.PI`)
- Algunas de las funcionalidades de la clase `System` son:
 - Recuperación de variables de entorno
 - Acceso a los flujos de entrada y salida estándar
 - Salida del programa actual (`System.exit()`)

Métodos estáticos y variables estáticas en la API de Java

Para declarar una variable o método `static` se debe tener en cuenta que:

- La realización de la operación en un objeto individual o la asociación de la variable a un tipo de objeto específico no es importante.
- El acceso a la variable o el método antes de instanciar un objeto sí es importante.
- El método o la variable no pertenece lógicamente a un objeto, pero posiblemente pertenezca a una clase de utilidad, como la clase `Math`, incluida en la API de Java.

Firma de método

Tipo de
método

Firma de
método

```
public int getYearsToDouble(int initialSum, int interest) {  
    int interest = 7;           // per cent  
    int years = 0;  
    int currentSum = initialSum * 100; // Convert to pennies  
    int desiredSum = currentSum * 2;  
    while ( currentSum <= desiredSum) {  
        currentSum += currentSum * interest/100;  
        years++;  
    }  
}
```

Sobrecarga de métodos

Métodos sobrecargados:

- Tienen el mismo nombre.
- Tienen firmas diferentes.
 - Número, tipo u orden de los parámetros diferente.
- Pueden tener funcionalidades diferentes o similares.
- Se utilizan ampliamente en las clases base.

Uso de la sobrecarga de métodos

```
public final class Calculator {  
  
    public static int sum(int numberOne, int numberTwo){  
        System.out.println("Method One");  
        return numberOne + numberTwo;  
    }  
  
    public static float sum(float numberOne, float numberTwo) {  
        System.out.println("Method Two");  
        return numberOne + numberTwo;  
    }  
  
    public static float sum(int numberOne, float numberTwo) {  
        System.out.println("Method Three");  
        return numberOne + numberTwo;  
    }  
}
```

Uso de la sobrecarga de métodos

```
public class CalculatorTest {  
  
    public static void main(String [] args) {  
  
        int totalOne = Calculator.sum(2,3);  
        System.out.println("The total is " + totalOne);  
  
        float totalTwo = Calculator.sum(15.99F, 12.85F);  
        System.out.println(totalTwo);  
  
        float totalThree = Calculator.sum(2, 12.85F);  
        System.out.println(totalThree);  
    }  
}
```

Sobrecarga de métodos y la API de Java

Método	Uso
<code>void println()</code>	Termina la línea actual escribiendo la cadena de separador de líneas.
<code>void println(boolean x)</code>	Imprime un valor booleano y, a continuación, termina la línea.
<code>void println(char x)</code>	Imprime un carácter y, a continuación, termina la línea.
<code>void println(char[] x)</code>	Imprime una matriz de caracteres y, a continuación, termina la línea.
<code>void println(double x)</code>	Imprime un valor <code>double</code> y, a continuación, termina la línea.
<code>void println(float x)</code>	Imprime un valor <code>float</code> y, a continuación, termina la línea.
<code>void println(int x)</code>	Imprime un valor <code>int</code> y, a continuación, termina la línea.
<code>void println(long x)</code>	Imprime un valor <code>long</code> y, a continuación, termina la línea.
<code>void println(Object x)</code>	Imprime un objeto y, a continuación, termina la línea.
<code>void println(String x)</code>	Imprime una cadena y, a continuación, termina la línea.