



Módulo 7

Métodos y subrutinas

¿Qué es un método?

Un método es un conjunto de instrucciones agrupadas para realizar una operación. Estas instrucciones se codifican fuera del `main()`.

El objetivo de un método es agrupar acciones de tal manera que en un futuro puedan ser reutilizables.

Por ejemplo:

- `println()` de la clase `System.out`
- `nextLine()` de la clase `Scanner`

¿Para que nos sirven los métodos?

1. Permiten reutilizar código en distintas partes.
2. Permiten generar código más limpio y entendible.
3. Nos ayudan a segmentar acciones de un programa en secciones lógicas.
4. Facilitan el mantenimiento.

```

String s1 = "abcde";
for(int i = 0; i<s1.length(); i++){
    if (i%2 == 0){
        System.out.print(s1.charAt(i));
    }
}
System.out.println();

s1 = "fghij";
for(int i = 0; i<s1.length(); i++){
    if (i%2 == 0){
        System.out.print(s1.charAt(i));
    }
}
System.out.println();

s1 = "klmno";
for(int i = 0; i<s1.length(); i++){
    if (i%2 == 0){
        System.out.print(s1.charAt(i));
    }
}
System.out.println();

```

Output

```

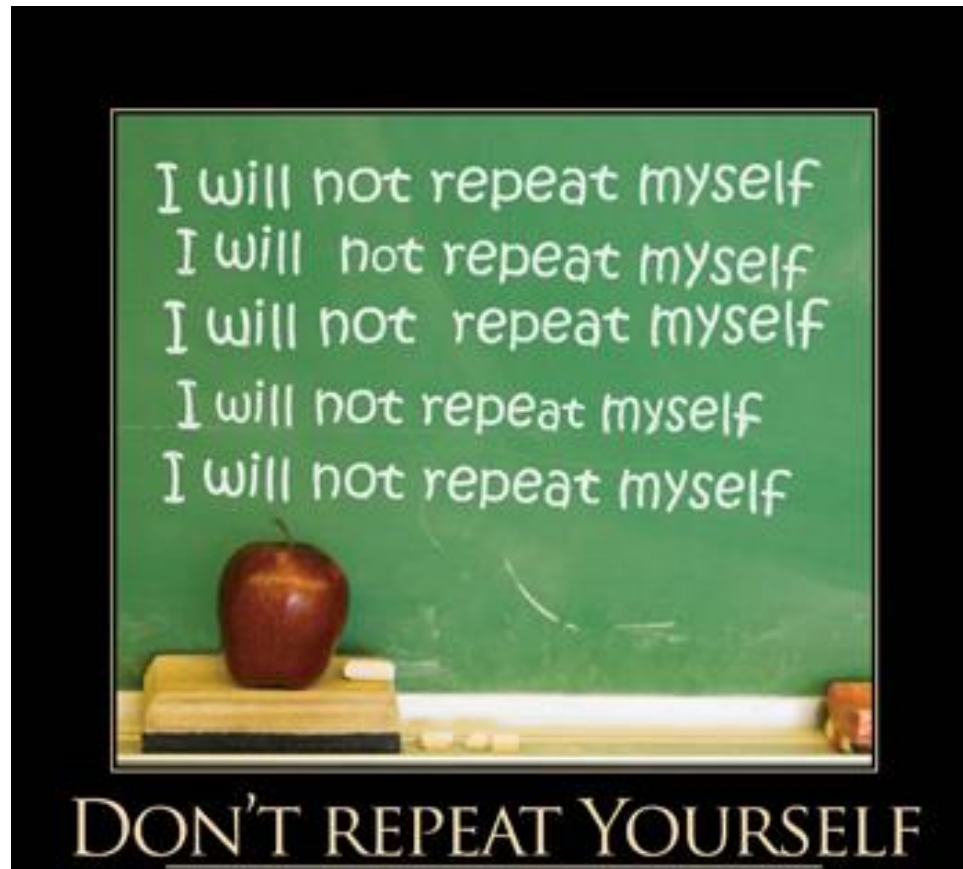
ace
fhj
kmo

```

!

**¿Es eficiente
repetir 3 veces el
mismo loop?**

DRY (Don't repeat yourself)



Repetition is the root of all software evil

```
public static void printEvenChars(String s1){  
    for(int i = 0; i<s1.length(); i++){  
        if (i%2 == 0){  
            System.out.print(s1.charAt(i));  
        }  
    }  
    System.out.println();  
}
```

```
public static void main(String[] args){  
    String s1 = "abcde";  
    printEvenChars(s1);  
  
    s1 = "fghij";  
    printEvenChars(s1);  
  
    s1 = "klmno";  
    printEvenChars(s1);  
}
```

Sintaxis

```
public static return_type methodName(type parameter1, type parameter2, ...)
{
    //
    // CODE TO BE EXECUTED
    //
}
```

return_type: Tipo de dato que va a retornar el método. Puede ser un **int**, **char**, **String** o **void**.

methodName Nombre identificador del método. Debe cumplir con todas las reglas de los nombres válidos de identificadores.

type parameter# Tipo de dato recibido como parámetro **int**, **char**, **String** y su nombre identificador.

Parámetros de entrada

Un método puede recibir información desde donde está siendo invocado. La información que es enviada a un método se denomina parámetro.

Cada parámetro debe ir acompañado de su tipo de dato y un identificador. Cuando incluimos más de 1 parámetro, los separamos por comas.

```
public static void doSomething(int p1, int p2, int p3) {  
    System.out.println(p1+p2+p3);  
}
```

Dentro del bloque que abarca el método doSomething, estamos declarando 3 parámetros: p1, p2 y p3.

Invocar un método

Para hacer uso de un método, debemos invocarlo. Invocar un método es la acción de llamar un método para ejecutar el código que se encuentra dentro.

```
public static void main(String[] args){
    double doubleTest = Math.PI;
    printFormattedDouble(doubleTest);
}

public static void printFormattedDouble(double d1) {
    String strDouble = String.format("%.2f", d1);
    System.out.println(strDouble);
}
```


En el main(), podemos observar que se invoca el método printFormattedDouble(), enviando como parámetro el valor de Math.PI

Invocar un método

```
public static void main(String[] args){
    double doubleTest = Math.PI;
    printFormattedDouble(doubleTest);
}

public static void printFormattedDouble(double d1) {
    String strDouble = String.format("%.2f", d1);
    System.out.println(strDouble); //no afecta a doubleTest

    d1 = 0;
}
```



Al invocar el método **printFormattedDouble()**, el contenido de la variable **doubleTest** es copiado a **d1**.
¡Pero son variables distintas!

Si **d1** es modificada, **doubleTest** NO es afectada.

Métodos con valor de retorno

Todos los caminos de estos métodos deben terminar con la instrucción **return**, en donde se regresará el contenido de alguna variable.

```
public static int addOne(int num){  
    return num + 1;  
}  
  
public static String concatenateTwoStrings(String s1, String s2) {  
    return s1 + s2;  
}
```

Métodos con valor de retorno

```
public static boolean checkValidYear(int year) {  
    if (year >= 0 && year <= 9999 ){  
        return true;  
    }  
}
```



¡Error de sintaxis!

Si la condición booleana no se cumple, el método no tiene un return!

```
public static boolean checkValidYear(int year) {  
    if (year >= 0 && year <= 9999 ){  
        return true;  
    }  
  
    return false;  
}
```



Todos los caminos cuentan con una salida adecuada.

Métodos void

Estos métodos no tienen un valor de retorno.

- Se especifica a través de la palabra **void**

La instrucción **return** es opcional, pero se puede utilizar para terminar anticipadamente un método.

```
public static void printTenNumbers(int i){  
    int j = i + 10;  
    while(i<j){  
        System.out.println(i++);  
    }  
}
```

Métodos void

Estos métodos no tienen un valor de retorno, y lo especificando con la palabra reservada **void**.

```
public static void printTenNumbers(int i){  
    int j = i + 10;  
    while(i<j){  
        System.out.println(i++);  
    }  
}
```

Métodos void

Estos métodos no tienen un valor de retorno, y lo especificando con la palabra reservada **void**.

```
public static void printTenNumbers(int i){  
    int j = i + 10;  
    while(i<j){  
        //never print bad luck numbers!  
        if (i == 13)  
            return;  
        System.out.println(i++);  
    }  
}
```

Variables locales

Las variables declaradas dentro de un método son variables *locales*.

Si declaramos dos variables con el mismo nombre en distintos métodos, son variables con contenido distinto.

```
public static void main(String[] args){  
    int i = 10;  
}  
  
public static void printNumber(){  
    int i;  
    System.out.print(i);  
}
```


Ejemplo

Diseña un método sin valor de retorno que reciba dos números enteros **n1** y **n2**, los compare e imprima en consola el número mayor.

Diseña un método con un valor de retorno entero que reciba dos números enteros **n1** y **n2** devuelva la multiplicación de ambos