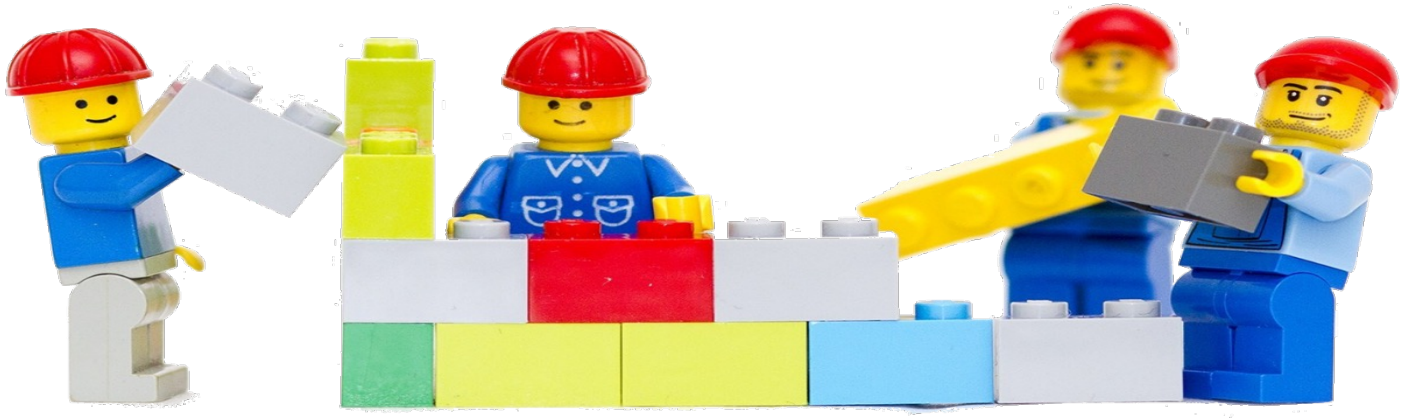


Topic 9 Methods

What is a method?

A method is a block of code that performs an operation. The objective of a method is to create a reusable block of code.

For example:

- `println()` in class `System.out`
- `nextLine()` in the `Scanner` class

What are methods used for?

1. Code reutilization.
2. Cleaner code that is easier to read and maintain.
3. Groups code into blocks of code that make sense together.

```java
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();
```

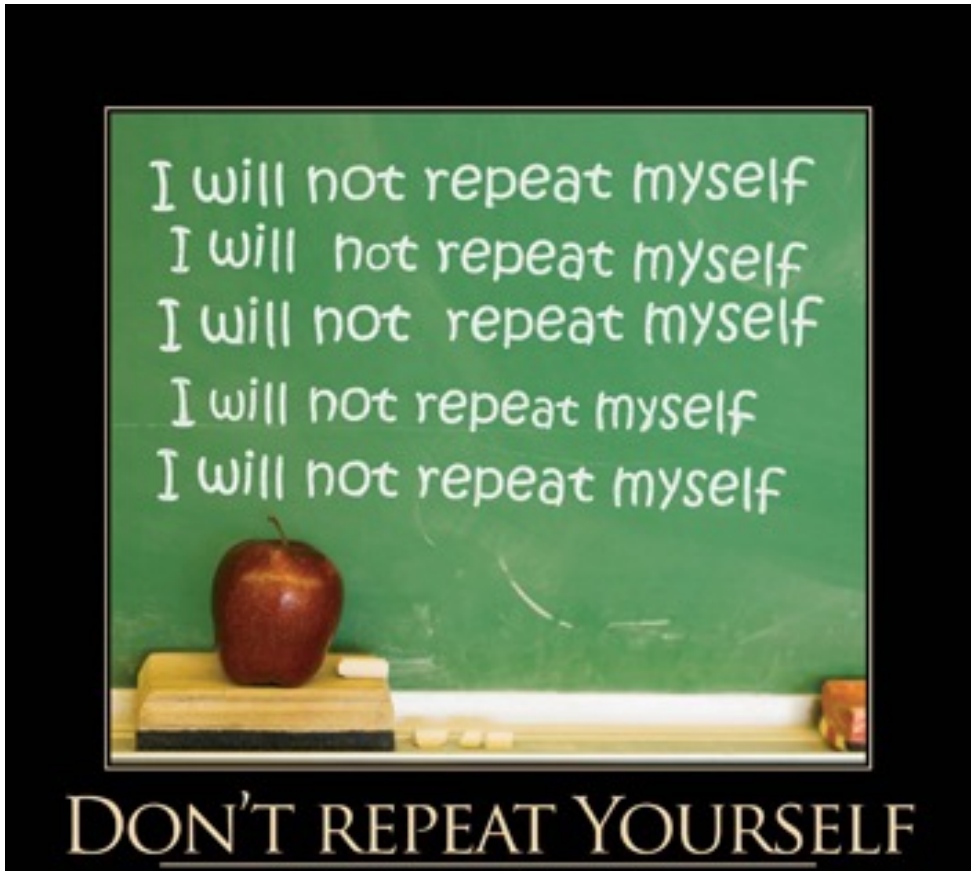
Output

```
ace
fhj
kmo
```

!

Do you think this is efficient?

DRY (Don't repeat youself)



Repetition is the root of all software evil

```java
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);
}
```

Syntax

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

`return_type:` Data type to be returned by the method. It can be an `int`, `char`, `String` or `void`.

`methodName` Name of the method. Must be a valid identifier name. It is a good practice to make method names verbs or actions

`type parameter#` Data type received as a parameter (`int`, `char`, `String`) and their identifier

Input parameters

A method receives inputs from where it is being called. In methods, all inputs are also called parameters.

Every parameter needs to be typed (int, char, double) and named (using a valid identifier).

When a method receives multiple parameters, they need to be separated by commas.

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

doSomething has three inputs: p1, p2 and p3

Calling a method

In the following example, printFormattedDouble() is invoked inside of the main() method.

```
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);
}
```

Calling a method

```
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);
    d1 = 0; //no afecta a doubleTest
}
```

When calling `printFormattedDouble()`, the contents of `doubleTest` are copied to varialbe `d1`.

But they are different variables!

If `d1` is modified inside the method, `doubleTest` is not affected.

Methods with return values

When methods are typed (int, String, double), all paths inside of a method needs to finish with a `return` statement. The code will return the results of the variable.

```
public static int addOne(int num){
    return num + 1;
}

public static String concatenateTwoStrings(String s1, String s2) {
    return s1 + s2;
}
```

Methods with return values

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

The code above would give a syntax error, because if the if() condition is NOT met, no return statement would be executed.

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