CHAPTER 12: METHODS

12.1. What Are Methods?

Methods are blocks of reusable codes designed to perform specific tasks. These can be called from anywhere in the code and does not require to be written multiple times. It is also known as functions in some other programming languages.

12.2. Types of Methods

- **Pre-Defined/Built-In methods**: These methods are already defined in java libraries e.g: .length(), .equals(), .compareTo(), .toLowerCase() etc. We only need to call these methods and these do the work for us.
- **User-Defined Methods**: These are the methods written by the programmer. Not only we have to call these methods but also design them. In this lesson, we will learn how to write user-defined methods.

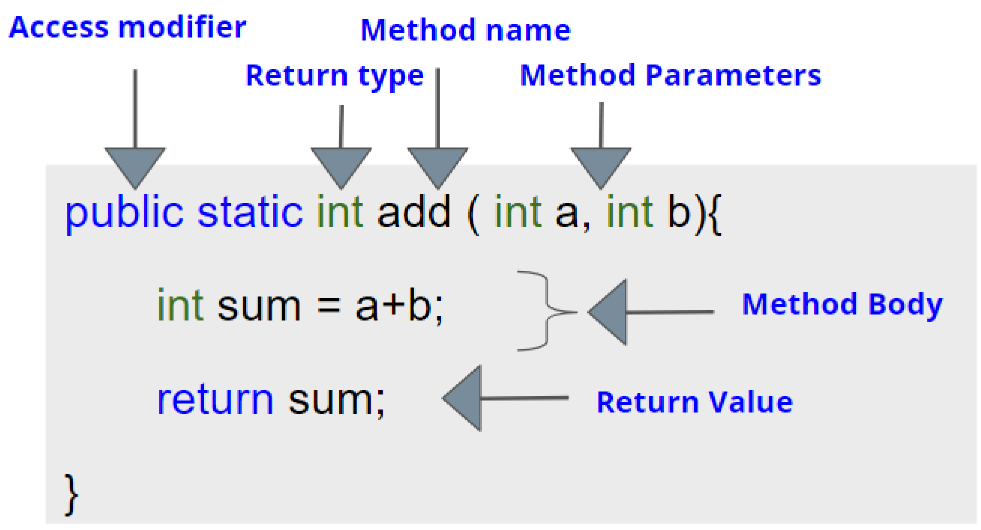
12.3. Basic Structure of a User Defined Method

A method consists of the following:

- **Access Modifier (public/private/protected/default):** Used to set access permission of the method. For the time being, we shall only focus on the public methods. A public method can be accessed from anywhere inside or outside the class.
- **static Keyword:** the keyword static indicates that a method is static. A static method belongs to the class and not to the instance of a class. As it belongs to the class, it can be called without creating any object. We use the keyword "static" to define a method as static. We can call a static method from inside any other static method.
- **Return Type:** Indicates the data type of the returned entity. If the method contains a return statement, the return type must match the data type of the returned entity. If a method does not have a return statement, or if we do not explicitly return any data or variable from the method it returns Null and the return type is written "**void**".
- **Method Name:** Every method has a name that should signify its purpose.
- **Parameter Type:** In order to achieve its purpose, a method may take none, one or more parameters. Datatype of every parameter must be specified.
- **Parameter Name:** Every parameter should be given a variable name, and the scope of these variables is within the method. Which means, these variables are not accessible outside that method.
- **Method Body:** Each method is designed for a unique purpose, and that purpose is fulfilled inside the method body. The method body varies for every method. Any variable declared inside the scope of a method is not accessible from outside including the parameter variables.
- **Return Statement(s):** A method may require one or more return statements depending on its method call. Return statements are the final lines of a method. No other line in the method is read after reaching one of the return statements and then we go back to the point where that method was called from.

Note that all of the above are not mandatory while writing a method. Depending on the situation, you may or may not include some of these fields. For example, a method may or may not have return statements, parameters, etc.

```
access_modifier static return_type method_name (parameter_type parameter_name)
{
        method body;
        return return_value;
}
```



```
public static int add ( int a, int b){
        int sum = a+b;
        return sum;
}
```

Now let us look at some of the examples of different return types in action:

1 void return

```
static void print_sum( int a, int b) {
    int sum = a+b;
    System.out.println(sum);
 } //this method does not return any value.
```

2 int return

```
static int sum( int a, int b) {
    int sum = a+b;
    return sum;
 } //this method returns an integer type value.
```

3 boolean return

```
static boolean check( int a) {
    if(a>10){
        return true;
    }
    else{
        return false;
    }
 } //this method returns a boolean type value.
```

4 String return

```
static String okay(int a, String str1) {
    return (a+2)+str1;
 } //this method returns a String type value.
```

5 double return

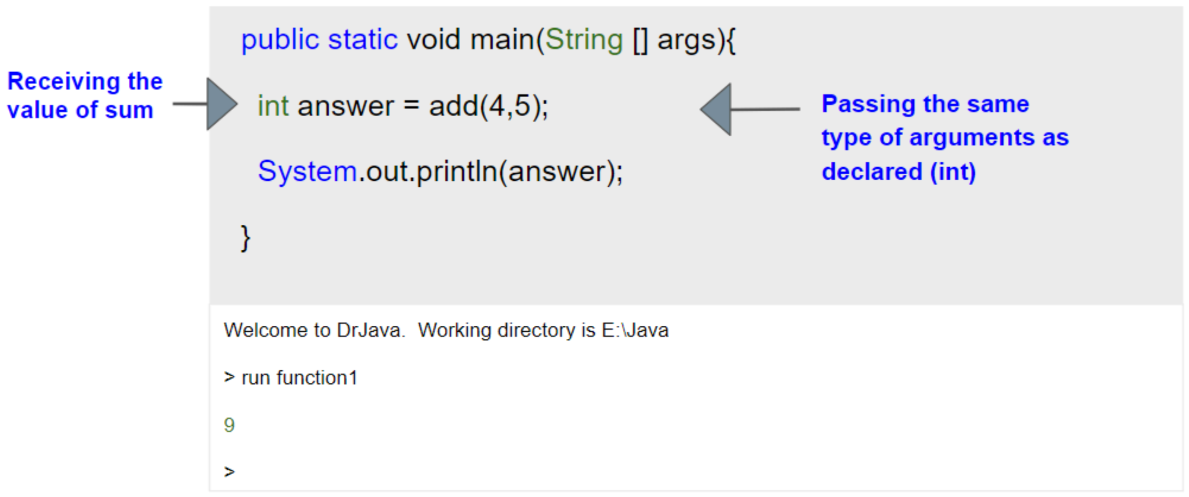
```
static double point(int a) {
        double db = a/10.0;
        return db;
 } //this method returns a double type variable.
```

12.4. Method Call

We have stated that not only do we have to design the user-defined methods but also require to call them. Your code may have hundreds of methods but will stay useless unless you call them. Here are the important points you need to know about method calls:
To run a method we MUST call the method.

- Method call is done inside the main() method.
- The number of arguments passed must be equal to parameters received.
- The sequence of the parameters passed is important.
- The type of the arguments passed must be the same as the type of parameters being received.
- After a method call, a value is returned to the line of method call.

Calling the previously written add method.

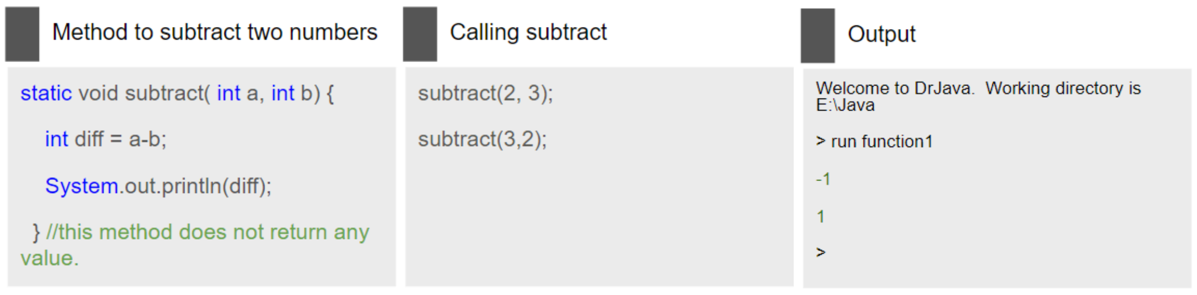

Receiving the value of sum
Passing the same type of arguments as declared (int)

```
public static void main(String [] args){
  int answer = add(4,5);
  System.out.println(answer);
}
```

```
Welcome to DrJava.  Working directory is E:\Java
> run function1
9
>
```

12.5. Method Arguments & Parameters

The sequence of the arguments passed corresponds to the variables receiving the parameters. Which means the entities we send during the method call are called arguments and the entities received by the method through its variables are called the parameters.



Method to subtract two numbers

```
static void subtract( int a, int b) {
   int diff = a-b;
   System.out.println(diff);
 } //this method does not return any value.
```

Calling subtract

```
subtract(2, 3);
subtract(3,2);
```

Output

```
Welcome to DrJava.  Working directory is E:\Java
> run function1
-1
1
>
```

Here in case 1: subtract(2,3), the values 2 and 3 will be received as int a = 2 and int b = 3. Therefore, the difference is (2-3) = -1.

In case 2: int a = 3 and int b = 2 and so the difference is (3-2) = 1.

12.6. Understanding Whether a Return Statement is Needed or Not

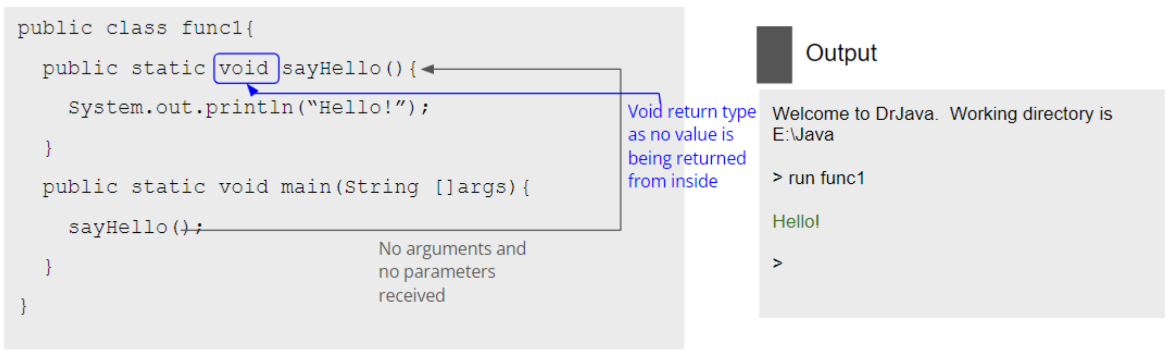
There are two cases where your method must have at least one return statements:

- If the method call is initialized in a variable
- If the method call if inside another user-defined or built-in method

In these two cases if your method does not explicitly return anything, a Null type is returned by default and the results may be catastrophic.

In the following example, the method did not require any return statement, therefore the return type was void and no return statements were written.

Writing a method that prints “Hello!”



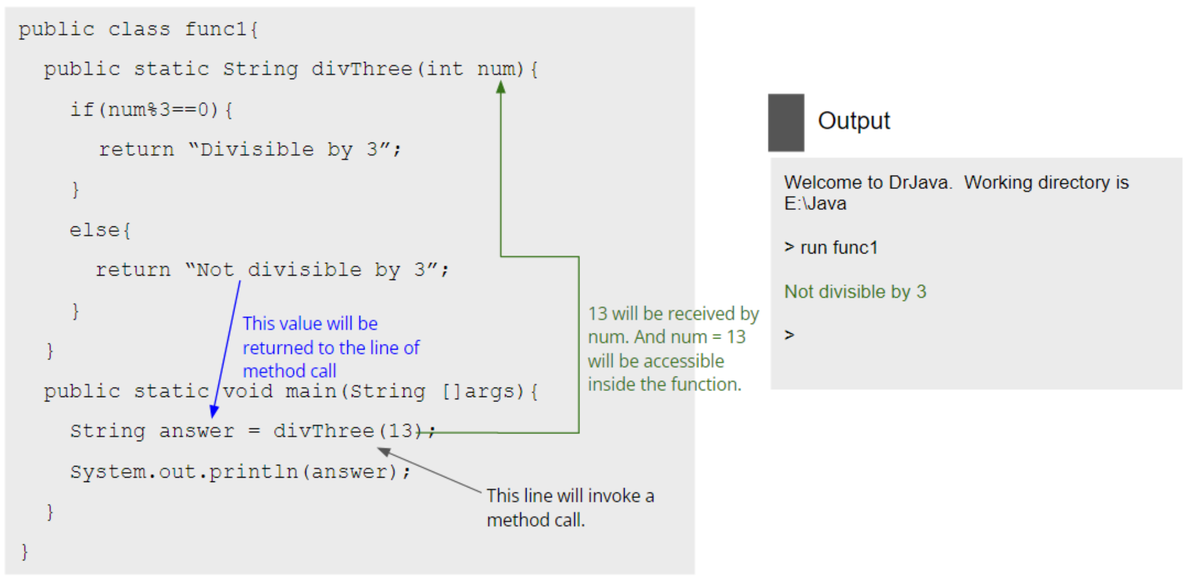
Void return type as no value is being returned from inside
No arguments and no parameters received
Output

```
public class func1{
  public static void sayHello(){
    System.out.println("Hello!");
  }
  public static void main(String []args){
    sayHello();
  }
}
```

```
Welcome to DrJava.  Working directory is
E:\Java

> run func1

Hello!

>
```

In the following example, the method call is initialized inside a variable. Therefore, return statements were required.

Writing a method that checks if a number divisible by three or not.



This value will be returned to the line of method call
13 will be received by num. And num = 13 will be accessible inside the function.
This line will invoke a method call.
Output

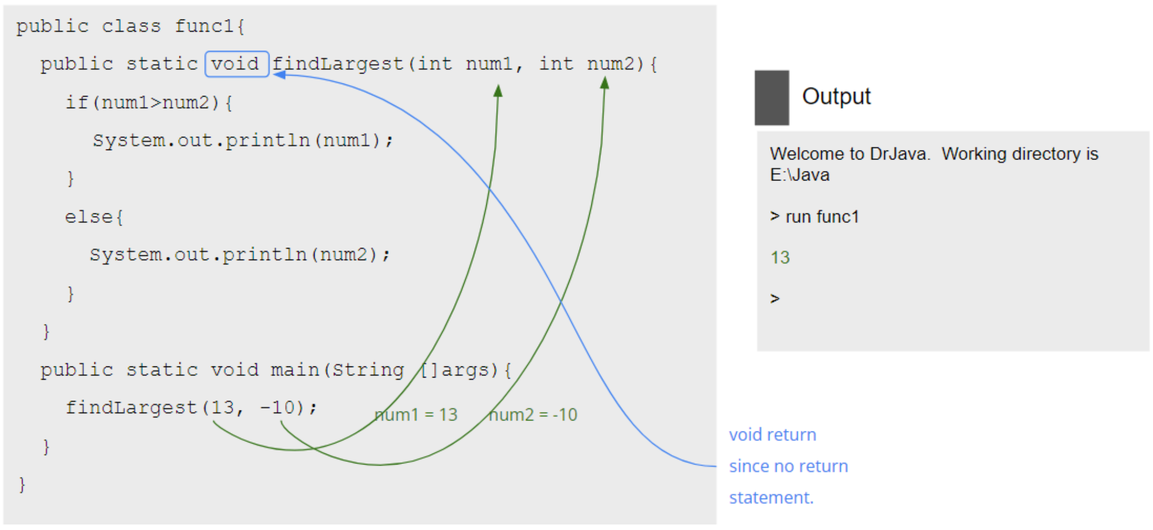
```
public class func1{
  public static String divThree(int num){
    if(num%3==0){
      return "Divisible by 3";
    }
    else{
      return "Not divisible by 3";
    }
  }
  public static void main(String []args){
    String answer = divThree(13);
    System.out.println(answer);
  }
}
```

```
Welcome to DrJava.  Working directory is
E:\Java

> run func1

Not divisible by 3

>
```

The example above also shows that your method can have more than one return statement inside the if-elif-else blocks. In this case, only one of the possibilities can happen and only one return statement will be executed at a time.

In the next example, again no return statement was required.

Writing a method that prints the largest number out of two numbers.



```
public class func1{
  public static void findLargest(int num1, int num2){
    if(num1>num2){
      System.out.println(num1);
    }
    else{
      System.out.println(num2);
    }
  }
  public static void main(String []args){
    findLargest(13, -10);
  }
}
```

```
Welcome to DrJava.  Working directory is
E:\Java

> run func1

13

>
```

12.7. Method Signature

- A method is identified based on its name, parameter list it receives and return type.
- The number of parameters passed in method call must be equal to the number of parameters received during method declaration.
- Multiple methods can have the same name but different sets of parameters.
- Multiple methods can have the same name and same set of parameters but have different return types.

The red marked lines are called method overloading and in CSE111, we shall explore this in detail.

12.8. Calling a Method from another Method

We can call a method from inside another method. In the following example:

- method1() is called from the main() method.
- method2() is called from method1().

```
public class Methods {
   public static void main(String[] args) {
       method1();
   }
   static void method1(){
       System.out.print("CSE");
       method2();
   }
   static void method2(){
       System.out.print("110");
   }
}
```

```
Welcome to DrJava. Working directory is
C:\Users\Desktop
CSE110
```

12.9. Worksheet

A. Specify when the following are needed and not needed:
   a. Parameter Type
   b. return Type
   c. Return Statements

B. Write a method named evenPositive to find even and positive numbers from an array of numbers.

C. Now, can you print a triangular palindrome like the following using a method called show_triangle()? Note that you have to take the help of show_spaces() and show_palindrome() methods.

```
1
121
12321
1234321
123454321
```

show_spaces() method prints a number of spaces passed to its parameter
show_palindrome() prints a palindrome. These methods are already done for you.

```java
public class MethodCalling {
   public static void main(String[] args) {
       show_triangle(4);
       System.out.println();
       show_triangle(5);
   }
   static void show_spaces(int a){
       for (int i = 0; i<a; i++){
           System.out.print(" ");
       }
   }
   static void show_palindrome(int b) {
       for (int i = 1; i <= b; i++) {
           System.out.print(i);
       }
       for (int i = b - 1; i >= 1; i--) {
           System.out.print(i);
       }
   }
      static void show_triangle(int a){
        //To Do
   }
}
```