Tutorial 5
CSC 201

Java Programming Concepts

مبادئ البرمجة باستخدام الجافا

Chapter 6: Methods

1. What is a Method?

2. Why do we need methods?

Modularity Code reuse Abstraction

3. Methods Input / Output

4. Methods in Java

Types of methods
Method Declaration
Building block of a method
The ‘main’ method

5. Difference between method declaration and method invocation

What happens when we call a method
Examples
Method Calling
Who can call who?
Order of Calls

What is a Method?

- A method is like a mini-program that can be run over and over.
- It is considered a smaller block of code designated to do something.
- It usually takes input and gives output, but sometimes it doesn't need one or the other (or either).
- Methods are sometimes called "subroutines" or "functions."

Why do we need methods?

1. Modularity

As a program becomes more complex, it is easier to divide it into sub-programs, each doing a certain task.

2. Code Re-use

Write a small piece of code that can be used (called) multiple times.

3. Abstraction

Provides operations to be performed on objects.

This point will be discussed in detail after we introduce Objects and Classes.

Methods Input/Output:

It is logical that each method should have a set of input values, and a value that it returns back to its caller.

Methods in Java

In Java, there are two types of methods:

1.Methods with return values:

These methods are those that "return", "evaluate to", "result in" some value.
A method can have multiple arguments (parameters), but only one result.

Why do we need value-returning methods?

- Save the value for future calculation
- Use the value in another calculation
- Print the value

2. Methods with no return values:

Such a method performs an operation but returns no value:

Why do we need no-return methods?

- Sometimes we need a method to simply perform a job without returning anything.
- For example, a method that is supposed to simply print a message does not need to return any value.

Java Syntax

Method Declaration

type method_name ([parameter_list])
{
//method body
}

Building blocks of a method:

1. Method Header

type method_name ([parameter_list])
Example:
double calculateAverage(double x, double y)

2. Parameter List

type method_name (data_type param1, data_type param2,..., data_type paramN)
double x, double y
which indicates that the caller of this method should send two parameters each of type double and they will be referred to as x and y throughout the method body.
Parameters provide a means of getting (passing) values to a method

3.Return Type

Indicates the type of value that the method evaluates to (or, sends back to the calling location).

double *calculateAverage(double x, double y)*

- The return type of our method is double, thus the average we return must be of type double.

void is a reserved word indicating that nothing is returned. It is used when a method is meant to perform a certain task, and does not need to return a value.

Example:

```
void printValue(int x)
{
   System.out.println("The value of x is : "+x);
}
```

- In the above example, the method printValue() prints the value of the integer x sent to it, but does not return a value to its caller.

4. The return Statement

Tells the program to "return" back to the method caller. Not only this, but also specifies the value that should be returned.

Syntax :

1) Format 1:

return expression;

The data type of expression must match the method's return type.

Ex: return x; return (x*y);

```
double calculateAverage(double x, double y)
{
  double average = (x + y) /2;
  return average;
}
```

2) Format 2:

return value;

```
  return 3.2;
```

Thus, we can directly return values as long as they are the same data type as the return type of the method.

What about void methods ?

Methods with return types of void usually don't have a return statement. They can have a return statement which basically makes the method return to its caller at this point .

Thus, the above example can be re-written as

```
void printValue(int x)
{
   System.out.println("The value of x is : "+x);
   return;
}
```

It can be used to enforce returning back to the caller upon certain conditions :

```
public static void checkIfNegative(int x)
{
   if(x<0)
     return;
   else  System.out.println("The number "+x+" is positive");
}
```

5. Method Body

This is the part where the actual code is written. It follows the method header.

```
double calculateAverage(double x, double y)
{
  double average = (x +y) /2;
  return average;
}
```

What about the main method?
The main method is the first method that runs as soon as we run our program. Unlike any other method, the main method does not need anyone to call it. It just runs as soon as the program runs.
Since it is a method, it has the same syntax that we explained for any method.

main method declaration:
public static void main (String[] args)

- **void** is the return type of the method, which indicates that this method will return nothing.
- **main** is the name of the method.
- **String[] args** is the parameter list, the method takes one parameter of type String[]
- **public:** A Java keyword, which basically specifies who can call this method.
- static: A Java keyword, which indicates that it does not need to be invoked by an object.

Difference between Method Declaration and Method Invocation
The declaration is where we write the code that the method should do, but the method will not run unless someone invokes/calls it!

What happens when you call a method ?
The answer is simple. From wherever part you call your method, the java compiler goes and searches for the method with this name and passes to it the parameters you sent. After that, the method executes all its method body, finally returning to its caller.

Examples

Example 1

In this example, we would like to print a message to the user.

```
public class Print
{
    //method declaration
    public static void printMessage(String message)
    {
        System.out.println(message);
    }

    public static void main(String[] args)
    {
        //calling the method
        printMessage("Hello World!");
    }
}
```

Program Output:

Hello World!

Example2

In this example, we would like to return the larger of two given numbers.

```
public class Larger
{
    public static double larger(double x,double y)
    {
      if (x >= y)
        return x;
      else
        return y;
    }
    public static void main(String[] args)
    {
      double num1, num2;
      num1 = 250.57;      num2 = 112.0;
     System.out.println("Larger: "+larger(num1,num2));
   }
}
```

```
double larger(double x,double y)
{
      double largerN;
      if (x >= y)
        largerN = x;
      else
        laeregrN = y;
      return largerN;
 }
```

Run: Larger: 250.57

Example3

We would like to implement three mathematical functions of a calculator.

```
public class Calculator
{
   public static int square(int num)
   {    return num * num ;    }
   public static int cube(int num)
   {  int result= num * num * num;
       return result;     }
   public static long add(long num1,int num2)
  {   long result = num1 + num2;
       return result;     }
  public static void main(String[] args)
  {
    int sq,cb,number,number1,number2;
    number = 5;     number1 = 3;       number2 = 4;
    sq = square(number);
    cb = cube(number);
    System.out.println("The square of "+ number +" is "+sq);
    System.out.println("The cube of “+number+" is "+cb);
    System.out.println(number1+“+"+number2+" = "+add(number1,number2) ) ;
   }
}
```

Program Output:

The square of 5 is 25

The cube of 5 is 125

3 + 4 = 7

Method Calling

Who can call whom?

In fact, any method can call any other method declared in the same program. (order?)
We can replace the above method **cube()** with the following method, it will still do the same thing but by calling method**square()**

```
public static int cube(int num)
{
    int result= square(num)*num;
    return result;
}
```

Example4

```
// function for computing number of digits in n
public static int Num_of_Digits(int n)
{
  int digits = 0;
  while(n>0)
  {
    digits = digits + 1;
    n = n/10;
  }
  return digits;
}
```

Example5

We would like to check if a number is even.

```
class check_even
{
   public static boolean Is_even(int k)
   {
     if (k%2==0) return true;
     else return false;
   }
  public static void main(String args[])
  {
    boolean flag;
    int i=11;
    int j=21;
    flag = Is_even(i*j);
    System.out.println(‘‘IS i*j EVEN : ’’+flag);
   }
}
```