Computer Programming Using JAVA
COP 2800 - Fall 2016

Lecture 6: Java Methods

By Ahmed E. Khaled
Recap to previous lecture!

• What are the different control statements?
• How to get input from a user?
Lecture 6 : Java Methods

• What is a Java Methods.
• How to invoke Method.
• Nested Methods.
One way to break up a complex program into manageable pieces is to use subroutines (methods).

A subroutine consists of the instructions for carrying out a certain task, grouped together and given a name.

Elsewhere in the program, that name can be used as a stand-in for the whole set of instructions.
As a computer executes a program, whenever it encounters a method name, it executes all the instructions necessary to carry out the task associated with that method.

Methods can be used over and over, at different places in the program, and even inside another method.

A subroutine is sometimes said to be a “black box” as you don’t want to see inside it and focus on your program.
Java Method

• A method is a set of code which is referred to by *name* and can be called (invoked) at any point in a program simply by utilizing the method's name.

• Think of a method as a subprogram that acts on data and often returns a value.
• When the name of a method is encountered in a program, the execution of the program branches to the body of that method.

• When the method is finished, execution returns to the area of the program code from which it was called, and the program continues on to the next line of code.
Public Main()
{
    Statement;
    Statement;
    Method1();
    Statement;
    Statement;
    Method2();
    Statement;
    Statement;
}

Method1()
{
    Statement;
    Statement;
    Statement;
}

Method2()
{
    Statement;
    Statement;
    Statement;
}
Why we need Methods?

• The use of methods will be our first step in the direction of modular programming which allows several programmers to work independently then assemble the work.

• Methods are time savers, in that they allow for the repetition of sections of code without retyping the code.

• In addition, methods can be saved and utilized again and again in newly developed programs.
Imagine your program is doing range summation in some parts, the first one is 1 – 10, second is 12 – 15 and third is 40 – 50.

Wouldn’t it be nice if we could write the *common code* once and *reuse* it? Just give it the start and end of the required range, and get back the summation.
Methods’ Types

There are two basic types of methods:

1. Built-in: part of the compiler package, such as `System.out.println()` and `System.exit(0)`.

2. User-defined: methods created by you, take names that you assigned and perform tasks that you create.
Method Structure

A method structure consists of its name, parameters, return value type, and body. The syntax is as follows:

```
modifier returnType methodName (list of parameters) {
    // Method body
    int result = num1 + num2;
    return result;
}
```
• Function is defined once, while invoked as required.
• A method *may* return a value with data type, If method is not returning a value, we use keyword *void* in function definition.

```java
public class WelcomeJava {

    public static void printHello() {
        System.out.println("Print Hello from function");
    }

    public static void main(String[] args) {
        printHello();
        printHello();
    }
}
```
A method *may* return a value with data type. If method is returning a value, we use its data type.

```java
public static int returnHundred() {
    return 100;
}

public static void main(String[] args) {
    int num1 = returnHundred();
    System.out.println("Value of Num " + num1);
}
```
The variables defined in the method header are known as parameters, the placeholders for values you pass to the method when the method is invoked, *optional*.

```java
public static int addNum( int num1, int num2 ){
    int result = num1 + num2;
    return result;
}

public static void printHello(){
    System.out.println("Print Hello from function");
}
```
Simple example, define a method to find the summation of two integers, this method has:

1. name sumBoth.
2. Takes two int parameters, num1 and num2.
3. Returns the summation of them.

```java
public static int addNum( int num1 , int num2 ){
    int result = num1 + num2;
    return result;
}
```
The method body of the `addNum` method defines new variable to hold the summation of the 2 passed values.

In order for a value-returning method to return a result, a return statement using the keyword `return` is required.
public static int sumBoth ( int num1 , int num2 ){
    int result = num1 + num2;
    return result;
}

public static void main(String[] args) {
    int x = 100;
    int y = 200;
    int z = sumBoth ( x , y );
    System.out.println("result = " + z );

    int m = 300;
    int n = 500;
    System.out.println("result = " + sumBoth ( m , n ));
}
Simple example, define a method to find the larger between two integers.

```java
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```
public static void printHello()
{
    System.out.println("Print Hello from function");
}

public static void printValue( int num )
{
    System.out.println("Print Value from function " + num);
}

public static void main(String[] args) {
    printHello();
    
    printValue(100);
    
    int x = 200;
    printValue(x);
}
Each time a method is invoked, the system creates an activation record that stores parameters and variables for the method and places the activation record in an area of memory known as a call stack.

When a method calls another method, the caller’s activation record is kept intact, and a new activation record is created for the new method called.

When a method finishes its work and returns to its caller, its activation record is removed from the call stack.
Suppose method m1 calls method m2, and m2 calls method m3, what the stack status in each case?

1. The runtime system pushes m1 ’s activation record into the stack, then m2 ’s, and then m3 ’s.
2. After m3 is finished, its activation record is removed.
3. After m2 is finished, its activation record is removed.
4. After m1 is finished, its activation record is removed.
When calling a method, you need to provide arguments, which must be given *in the same order* as their respective parameters in the method signature.

For example, the following method prints a message n times:

```java
public static void nPrintln(String message, int n)
```

- `nPrintln("Hello", 3)` prints Hello three times.
- However `nPrintln(3, "Hello")` would be *wrong*. 
Nested Functions

```java
public static void secondFunction()
{
    System.out.println("Here is the second function");
}

public static void firstFunction()
{
    System.out.println("Here is the first function");
    secondFunction();
}

public static void main(String[] args) {
    firstFunction();
}
```
public static void displaySortedNumbers(double num1, double num2, double num3)

public static double celsiusToFahrenheit(double celsius)
public static double fahrenheitToCelsius(double fahrenheit)

• The formula for the conversion is:
  
  fahrenheit = (9.0 / 5) * celsius + 32
  
  celsius = (5.0 / 9) * (fahrenheit − 32)
• isValid method for triangle, that returns true if the sum of any two sides is greater than the third side.
  
  public static boolean isValid( double side1, double side2, double side3)

• area of the rectangle
  
  public static double area(double length, double width)
• Write method that returns the minimum value of three double numbers.

• Write method that returns the maximum value of five int numbers.

• Define method that prints and returns A/B/C/D according to grade you give.