

OBJECT-ORIENTED PROGRAMMING

LAB 1: JAVA, HOW TO PROGRAM

I. Objective

After completing this first lab tutorial, you can:

- Practice with conditional statements, loop statements, methods, input in Java.

II. Overview of Java

Java is a general-purpose computer programming language that is concurrent, class-based, object-oriented, and specifically designed to have as few implementation dependencies as possible. It is intended to let application developers "write once, run anywhere" (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation. Java applications are typically compiled to bytecode that can run on any Java virtual machine (JVM) regardless of computer architecture.

As of 2016, Java is one of the most popular programming languages in use, particularly for client-server web applications, with a reported 9 million developers. The language derives much of its syntax from C and C++, but it has fewer low-level facilities than either of them.

III. Environment Setup

This section will help you to set up the environment for the Java programming language. Following the steps below:

- Download and install the [Java SE Development Kit](#). If you are using Microsoft Windows 64-bit, we recommend that you should install both 32-bit and 64-bit of Java SE Development Kit (JDK).
- Setting up the Path for Windows:
 - Right-click on My Computer and select Properties.
 - Click Advanced system settings.
 - Click the Path to include the directory path of installed JDK (as in FIG). Assuming you have installed in *C:\Program Files\Java\jdk1.8.0_121\bin*, then add it to the Path. Note that you can add only the 64-bit version of JDK.
 - Then, open command prompt (CMD) window and type **java -version**. If you get the message that provides the information about the installed Java version, that means you have successfully set up the JDK and Java Path.

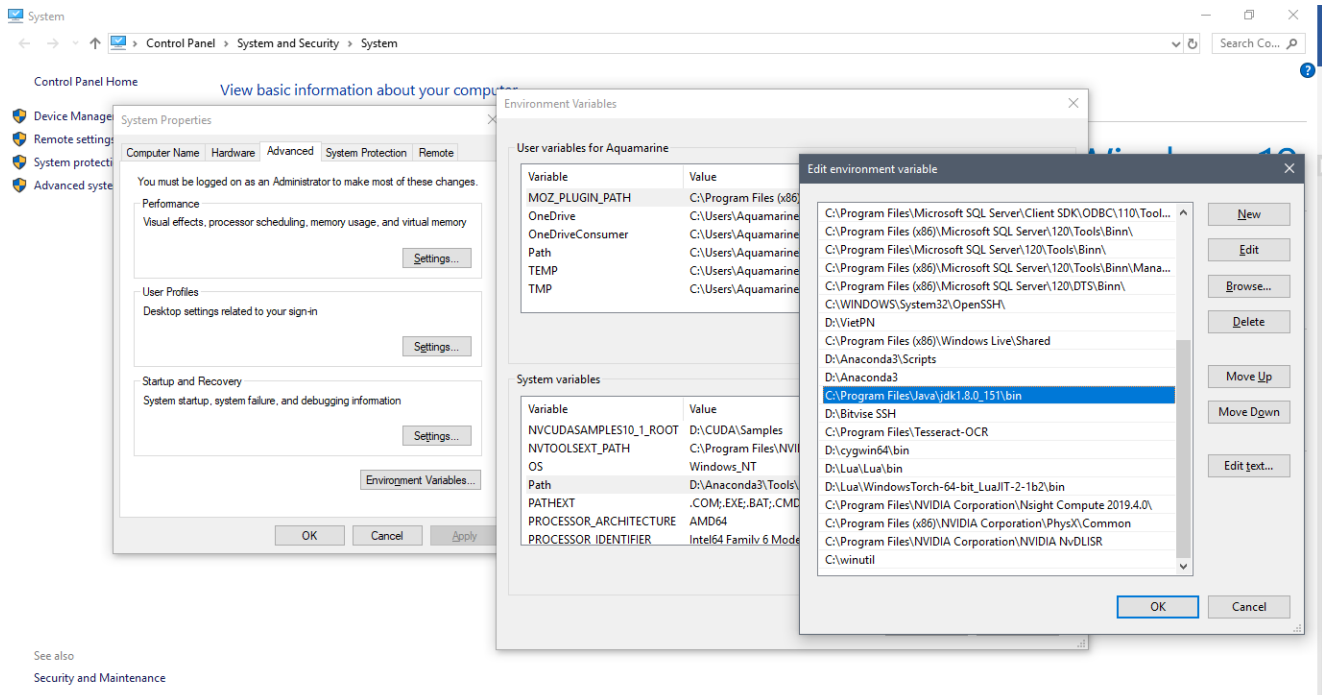


Figure 1. Environment Variables window

- For Java editor, we strongly recommend [Notepad++](#) for your very beginning course of Java. After completing this course, you can use other Java editors, e.g., Netbeans, Eclipse, IntelliJ IDEA.

IV. First Java Program

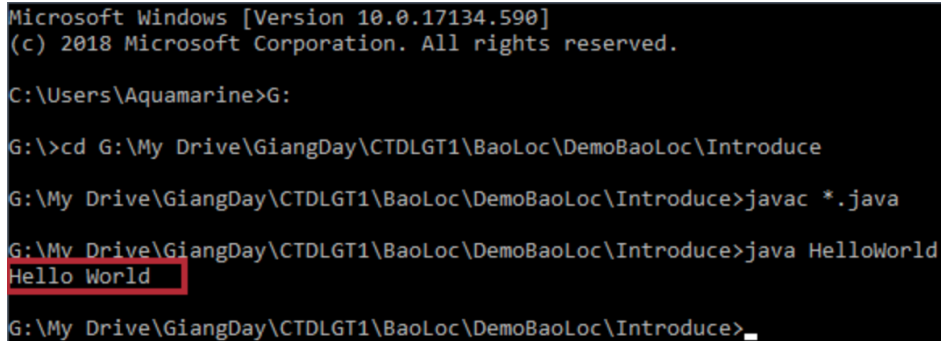
Let's look at and run your first Java program, which will print the "Hello World!" string on the screen.

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

Do the following steps to save the file, compile and run the program (as in Figure 2):

- Open the text editor program (Notepad++) and type the above code.
- Save as *MyFirstProgram.java*.
- Open the command prompt window (CMD) and navigate to the directory where you save the file.
- Type **javac MyFirstProgram.java** and press Enter to compile the source code.

- Type `java MyFirstProgram` to run the program.
- Now, you will see the "Hello World!" string on the screen.



```
Microsoft Windows [Version 10.0.17134.590]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Aquamarine>G:

G:\>cd G:\My Drive\GiangDay\CTDLGT1\BaoLoc\DemoBaoLoc\Introduce

G:\My Drive\GiangDay\CTDLGT1\BaoLoc\DemoBaoLoc\Introduce>javac *.java

G:\My Drive\GiangDay\CTDLGT1\BaoLoc\DemoBaoLoc\Introduce>java HelloWorld
Hello World

G:\My Drive\GiangDay\CTDLGT1\BaoLoc\DemoBaoLoc\Introduce>_
```

Figure 2. Compile and run Java program

1. Basic syntax

In java, you should keep in mind the following points:

- **Case Sensitivity:** Java is case sensitive, which means identifier Hello and hello would have a different meaning in Java.
- **Class Names:** For all class names, the first letter should be in Upper-Case. If several words are used to form the name of the class, each inner word's first letters should be in Upper-Case. Example: class `MyFirstProgram`
- **Method Names:** All method names should start with a Lower-Case letter. If several words are used to form the names of the method, then each inner word's first letter should be in Upper-Case. Example: `public void methodName()`
- **Program File Name:** Name of the program file has to exactly match the class name. Example: class `MyFirstProgram` -> `MyFirstProgram.java`
- `public static void main(String args[])`: Java program processing starts from the `main()` method which is a mandatory part of every Java program.

2. Java Identifiers

The identifier is the name used for classes, variables, and methods. To name an identifier in Java, you should remember the following points:

- All identifiers should begin with a letter (A to Z or a to z), currency character (\$) or an underscore (_)

- After the first character, identifiers can have any combination of characters.
- A [keyword](#) cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.
- Examples of legal identifiers: `age`, `$salary`, `_value`, `__1_values`.
- Examples of illegal identifiers: `123abc`, `-salary`.

3. Java Modifiers

There are two categories of modifiers in Java:

- Access Modifier: `default`, `public`, `protected`, `private`.
- Non-access Modifiers: `final`, `abstract`, `strictfp`.

4. Comments in Java

Java supports both single-line and multiple-line comments, and they are similar to C and C++. All characters available inside any comment are ignored by the Java compiler.

```
// This is single-line comment
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        System.out.println("Hello World!");
        /*
        This is multiple-line comment
        */
    }
}
```

V. Data Types

In this first lab tutorial, we only focus on primitive data types. For the user-defined data types, we will discuss later in this course.

Type	Description	Default	Size
boolean	true or false	false	1 bit
byte	two complement in integer	0	8 bits
char	Unicode character	\u0000	16 bits
short	two complement in integer	0	16 bits
int	two complement in integer	0	32 bits
long	two complement in integer	0	64 bits
float	IEEE 754 floating point	0.0	32 bits
double	IEEE 754 floating point	0.0	64 bits

Let's look example below:

```
// MyFirstProgram.java
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int a = 5, b = 10;
        int sum = a + b;
        System.out.println("a + b = " + sum);
    }
}
```

VI. Basic Operators

Java provides a rich set of operators to manipulate variables, including:

- Arithmetic operators
- Relational operators
- Bitwise operators
- Logical operators
- Assignments operators
- Misc. operators

In this lab tutorial, we discuss arithmetic operators, relational operators, logical operators, and assignment operators. For the others, you can read in the textbook.

1. Arithmetic operators

Arithmetic operators are used in the mathematical expression in the same way that they are used in algebra.

Operators	Example
+ (addition)	a + b will give 30
- (subtraction)	a – b will give
* (multiplication)	a * b will give 200
/ (division)	a / b will give 2
% (modulus)	a % b will give 0
++ (increment)	b++ will give 21
-- (decrement)	b-- will give 19

2. Relational operators

The followings are the relational operations supports by Java language.

Operators	Example
== (equal to)	(a == b) is false
!= (not equal to)	(a != b) is true
> (greater than)	(a > b) is false
< (less than)	(a < b) is true
>= (greater than or equal to)	(a >= b) is false
<= (less than or equal to)	(a <= b) is true

3. Logical operators

The followings are the logical operators supported by Java language.

Operators	Example
&& (logical and)	(A && B) is false
(logical or)	(A B) is true
! (logical not)	!(A && B) is true

4. Assignment operators

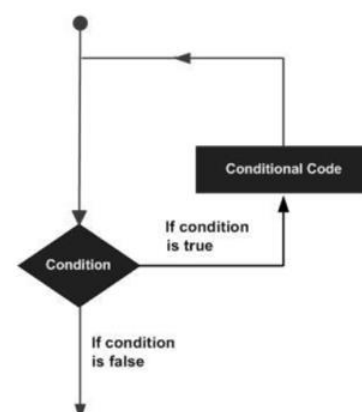
Operators	Example
=	C = A + B will assign value of A + B into C
+=	C += A is equivalent to C = C + A
-=	C -= A is equivalent to C = C - A
*=	C *= A is equivalent to C = C * A
/=	C /= A is equivalent to C = C / A
%=	C %= A is equivalent to C = C % A

VII. Loop Control

A loop statement allows us to execute a statement or group of statements multiple times (as in FIG).

Java programming language provides the following types of loop to handle looping requirements:

- **while** loop
- **for** loop, and enhanced for loop
- **do ... while** loop



1. *while* loop

A *while* loop statement in Java programming language repeatedly executes the target statement as long as a given condition is true.

Example:

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int i = 1, sum = 0;
        {
            while (i <= 0 )
            {
                sum = sum + i;
                int++;
            }
            System.out.println("sum = " + sum);
        }
    }
}
```

2. *for* loop

A *for* loop is a repetition control structure that allows you to efficiently write a loop that needs to be executed a specific number of times. A *for* loop useful when you know how many times a task is to be repeated.

Example:

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int sum = 0;
        {
            for (int i = 1; i <= 10; i++)
            {
                sum += i;
            }
            System.out.println("sum = " + sum);
        }
    }
}
```


3. *do ... while* loop

A *do ... while* loop is similar to while loop, except that, *do ... while* loop is guaranteed to execute at least one time.

Example:

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int i = 0, sum = 0;
        {
            do {
                sum = sum + i;
                i++;
            } while (i <= 10)
            System.out.println("sum = " + sum);
        }
    }
}
```

4. Enhanced *for* loop

The *enhanced for* loop is mainly used to traverse a collection of elements including arrays. The syntax is as follows.

Example:

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int[] a = {1, 3, 5, 7, 9};
        for (int x : a)
        {
            System.out.println(x);
        }
    }
}
```

VIII. Conditional Statements

Decision making structures have one or more conditions to be evaluated or tested by the program, along with a statement or statements that are to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Java provides two main types of conditional statements:

- **if ... else** statement
- **switch ... case** statement
- **? ... : ...** operator

Example 1:

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int x = 11;
        if (x % 2 == 0)
        {
            System.out.println("x is even");
        }
        else
        {
            System.out.println("x is odd");
        }
    }
}
```

Example 2:

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int x = 11, y = 12;
        if (x < y && x + y >= 10)
        {
            System.out.println("True");
        }
        else
        {
            System.out.println("False");
        }
    }
}
```

IX. Array

Java provides a data structure, the array, which stores a fixed-size sequential collection of elements of the same type. To declare an array, we have 4 possible ways:

```
dataType[] arrayName;  
dataType arrayName[];  
dataType[] arrayName = new dataType[arraySize];  
dataType[] arrayName = {value0, value1, ..., valueK};
```

Example:

```
public class MyFirstProgram  
{  
    public static void main(String[] args)  
    {  
        int[] a = {1, 2, 3, 4, 5};  
        int sum1 = 0, sum2 = 0;  
        for (int i = 0; i < a.length; i++)  
        {  
            sum1 = sum1 + a[i];  
        }  
        System.out.println("sum1 = " + sum1);  
        for (int x : a)  
        {  
            sum2 = sum2 + x;  
        }  
        System.out.println("sum2 = " + sum2);  
    }  
}
```

X. Methods

A Java method is a collection of statements that are grouped together to perform an operation. The syntax is as follows:

```
modifier returnType nameOfMethod (Parameter List)
{
    // statements
}
```

Example:

```
public class MyFirstProgram
{
    public static int findMax(int a, int b)
    {
        return (a > b)? a: b;
    }

    public static void main(String[] args)
    {
        int x = 5, y = 6;
        System.out.println("Max is " + findMax(x, y));
    }
}
```

XI. Text Input

Java provides a **Scanner** class, it is used to get user input, and it is found in the **java.util** package.

To use the Scanner class, create an object of the class and use any of the available methods found in the documentation.

Method	Description
nextBoolean()	Reads a boolean value from user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Read a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Example:

```
import java.util.Scanner;

class MyFirstProgram
{
    public static void main(String[] args)
    {
        Scanner sc = new Scanner(System.in);

        String name = sc.nextLine();
        int age = sc.nextInt();

        System.out.println("Name: " + name);
        System.out.println("Age: " + age);
    }
}
```

XII. Text Output

Here is a list of the various print functions that we use to output statements:

- **print()**: This method in Java is used to display a text on the console. This text is passed as the parameter to this method in the form of String. This method prints the text on the console and the cursor remains at the end of the text at the console. The next printing takes place from just here.

Example:

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

Output:

Hello! Hello! Hello!

- **println()**: This method in Java is also used to display a text on the console. It prints the text on the console and the cursor moves to the start of the next line at the console. The next printing takes place from the next line.

Example:

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

Output:

```
Hello!  
Hello!  
Hello!
```

- **printf()**: This is the easiest of all methods as this is similar to printf in C. Note that System.out.print() and System.out.println() take a single argument, but printf() may take multiple arguments. This is used to format the output in Java.

Example:

```
class Demo {  
    public static void main(String args[])  
    {  
        int x = 100;  
        System.out.printf("Printing simple" + " integer: x = %d\n",x);  
  
        System.out.printf("Formatted with" + " precision: PI = %.2f\n", Math.PI);  
  
        float n = 5.2f;  
  
        System.out.printf("Formatted to " + "specific width: n = %.4f\n", n);  
  
        n = 2324435.3f;  
  
        System.out.printf("Formatted to " + "right margin: n = %20.4f\n", n);  
    }  
}
```

Output:

```
Printing simple integer: x = 100  
Formatted with precision: PI = 3.14  
Formatted to specific width: n = 5.2000  
Formatted to right margin: n =                2324435.2500
```

XIII. Exercises

1. Write a program to print your name, date of birth, and student ID.
2. Write a program to compute the area of a rectangle with a height and width provided by user.
3. Write a program to convert the temperature from Fahrenheit to Celsius.
4. Write a program to check whether a year is a leap year or not.
5. Write a program to find minimum between three numbers.
6. Write a program to check whether a number is even or odd.
7. Write a program to input a character and check whether it is alphanumeric or not.
8. Write a program to input edges of a triangle and check whether triangle is valid or not.
9. Write a function to find the first digit and a function to find last digit of a number.
10. Write a function to calculate sum of digits and a function to calculate product of digits of a number.
11. Write a function to return the remainder of a division.
12. Write a function to count number of digits in a number.
13. Write a function to reverse the input integer number.
14. Write a function to check whether a number is palindrome or not.