

Data Structures and Algorithms

Lab 1: Java, How to Program

I. Objective

After completing this first lab tutorial, you can:

- Practice with conditional statements, loop statements, methods and array 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 developers¹. 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 setup the environment for Java programming language. Following the steps below:

- Download and install Java SE Development Kit². If you're 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 *Environment Variables* button under *Advanced* tab;
 - + Alter 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 provide the information the installed Java version, that means, you have successful set up the JDK and Java Path.

¹ [https://en.wikipedia.org/wiki/Java_\(programming_language\)](https://en.wikipedia.org/wiki/Java_(programming_language))

² <http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html>

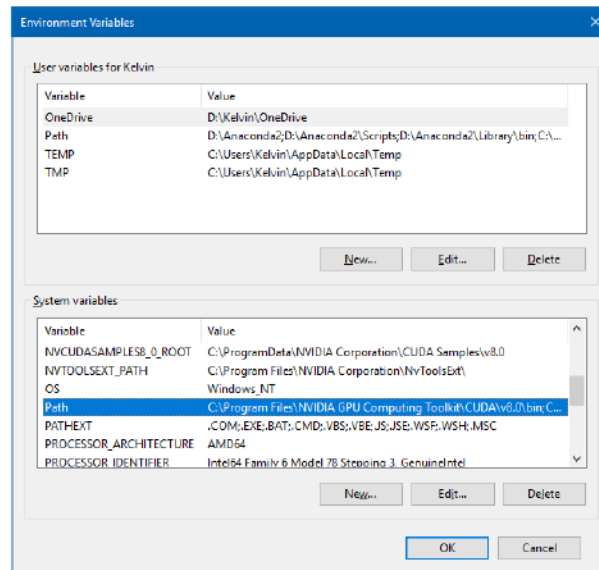


Figure 1 Environment Variables window

- For Java editor, we strongly recommend *Notepad++*³ 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

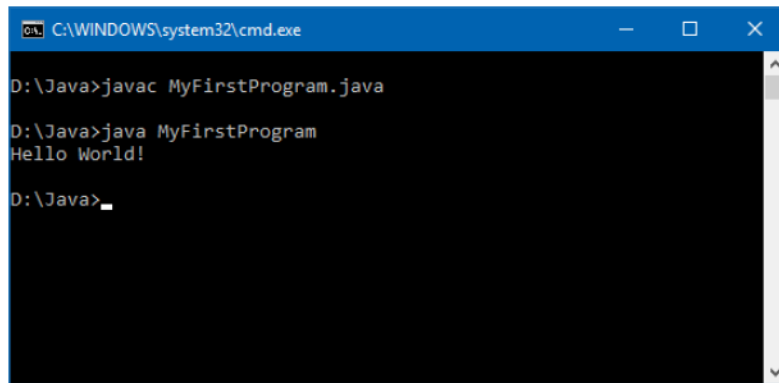
Let's look at and run your first Java program, which will print the *"Hello World!"* string on screen.

```
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 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 screen.

³ <https://notepad-plus-plus.org/download/>



```
C:\WINDOWS\system32\cmd.exe

D:\Java>javac MyFirstProgram.java

D:\Java>java MyFirstProgram
Hello World!

D:\Java>
```

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 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 a name of the class, each inner word's first letter 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 name 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

Identifier is 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 keyword⁴ cannot be used as an identifier.
- Most importantly, identifiers are case sensitive.
- Examples of **legal** identifiers: age, \$salary, _value, __1_value.

⁴ Full list of Java keyword - https://docs.oracle.com/javase/tutorial/java/nutsandbolts/_keywords.html

- Examples of **illegal** identifiers: 123abc, -salary.

3. Java modifiers

There are two categories of modifiers in Java:

- Access Modifiers – default, public, protected, private.
- Non-access Modifiers – final, abstract, strictfp.

4. Comments in Java

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

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

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	twos complement integer	0	8 bits
char	Unicode character	\u0000	16 bits
short	twos complement integer	0	16 bits
int	twos complement integer	0	32 bits
long	twos complement 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 at the 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
- Assignment 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 textbook.

1. Arithmetic operators

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

Operator	Description	Example
+ (Addition)	Adds values on either side of the operator.	a + b will give 30
- (Subtraction)	Subtracts right-hand operand from left-hand operand.	a - b will give -10
* (Multiplication)	Multiplies values on either side of the operator.	a * b will give 200
/ (Division)	Divides left-hand operand by right-hand operand.	b / a will give 2
% (Modulus)	Divides left-hand operand by right-hand operand and returns remainder.	b % a will give 0
++ (Increment)	Increases the value of operand by 1.	b++ gives 21
-- (Decrement)	Decreases the value of operand by 1.	b-- gives 19

2. Relational operators

Followings are the relational operators supported by Java language.

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(a == b) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(a != b) is true.

> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(a > b) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(a < b) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(a >= b) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(a <= b) is true.

3. Logical operators

Followings are the logical operators supported by Java language.

Operator	Description	Example	Operator
&& (logical and)	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false	&& (logical and)
(logical or)	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A B) is true	(logical or)
! (logical not)	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	!(A && B) is true	! (logical not)

4. Assignment operators

Operator	Description	Example
=	Simple assignment operator. Assigns values from right side operands to left side operand.	C = A + B will assign value of A + B into C
+=	Add AND assignment operator. It adds right operand to the left operand and assign the result to left operand.	C += A is equivalent to C = C + A

- =** Subtract AND assignment operator. It subtracts right operand from the left operand and assign the result to left operand. $C -= A$ is equivalent to $C = C - A$
- *=** Multiply AND assignment operator. It multiplies right operand with the left operand and assign the result to left operand. $C *= A$ is equivalent to $C = C * A$
- /=** Divide AND assignment operator. It divides left operand with the right operand and assign the result to left operand. $C /= A$ is equivalent to $C = C / A$
- %=** Modulus AND assignment operator. It takes modulus using two operands and assign the result to left operand. $C \% = A$ 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

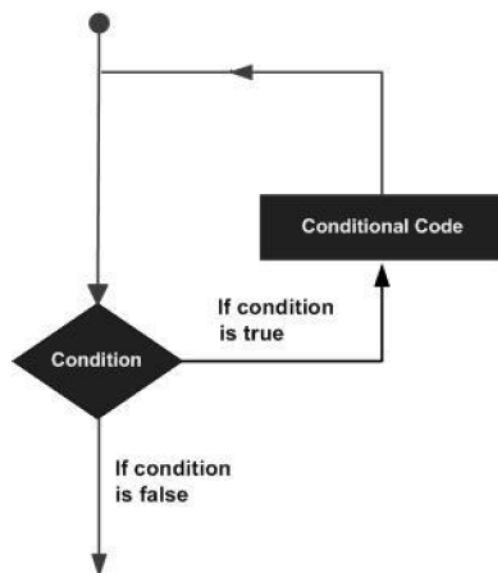


Figure 3 Loop structure

1. **while** loop

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

```
while(boolean_expression)
{
    // statements
}
```

Example:

```
public class MyFirstProgram
{
    public static void main(String[] args)
    {
        int i = 1, sum = 0;

        while (i <= 10)
        {
            sum = sum + i;
            i++;
        }

        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 is useful when you know how many times a task is to be repeated.

```
for (initialization; boolean_expression; update)
{
    // statements
}
```

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

```
do
{
    // statements
} while(bool_expression);
```


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 collection of elements including arrays. Syntax is as follows:

```
for (declaration : expression)
{
    // statements
}
```

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. Exercises

1. Write a program to print your name, date of birth, and mobile number.
2. Write a program prompting user to input two integer numbers⁵, then compute and print the results of addition, subtraction, multiplication, division, and remainder.
3. Write a program to compute the perimeter and area of a rectangle with a height provided by user.
4. Write a program to convert specified days into years, weeks and days. (Note: ignore leap year).
5. Write a program to convert the temperature from Celsius to Fahrenheit.
6. Write a program to return an absolute value of a number.
7. Write a program to check whether a year is a leap year or not.

⁵ Sample program - <http://it.tdt.edu.vn/~dhphuc/teaching/cs501043/lab/scanner-sample.zip>

8. Write a program to find maximum between two numbers.
9. Write a program to find maximum between three numbers.
10. Write a program to check whether a number is even or odd.
11. Write a program to input a character and check whether it is alphanumeric or not.
12. Write a program to input angles of a triangle and check whether triangle is valid or not.
13. Write a program to input marks of five subjects Physics, Chemistry, Biology, Mathematics and Computer. Calculate percentage and grade according to following:

Percentage > 90%: Grade A

Percentage > 80%: Grade B

Percentage > 70%: Grade C

Percentage > 60%: Grade D

Percentage > 40%: Grade E

Percentage < 40%: Grade F
14. Write a program/function to find first and last digits of any number.
15. Write a program/function to calculate sum of digits of any number.
16. Write a program/function to calculate product of digits of any number.
17. Write a program/function to count number of digits in any number.
18. Write a program to swap first and last digits of any number.
19. Write a program/function to reverse the input integer number.
20. Write a program/function to enter any number and check whether the number is palindrome or not.
21. Write a program/function to check whether a number is Prime number or not. Validating the input, in case the input isn't correct, prompt user to enter it again.
22. Write a program/function to check whether a number is Armstrong number or not.
23. Write a program/function to check whether a number is Perfect number or not.
24. Write a program/function to print all Prime numbers between 1 to n by using three loop structures. Validating the input, in case the input isn't correct, prompt user to enter it again.

25. Write a program/function to print all Armstrong numbers between 1 to n by using three loop structures. Validating the input, in case the input isn't correct, prompt user to enter it again.
26. Write a program/function to print all Perfect numbers between 1 to n by using three loop structures. Validating the input, in case the input isn't correct, prompt user to enter it again.
27. Write function to find the maximum number of an integer array.
28. Write function to find the minimum number of an integer array.
29. Write function to sum all numbers of an integer array.
30. Write function to sum all non-positive numbers of an integer array.
31. Write function to sum all even numbers of an integer array.

XII. Homework

Read **Chapter 16**⁶, in *Java: How to Program, 9th Edition (Deitel)*, to get used to **Java's String** data type.

⁶ <http://it.tdt.edu.vn/~dhphuc/teaching/cs501043/lab/java-string.pdf>