



Language Basics

Agenda

1

Language Basics

Language Basics



A Simple Java Program – The Set Up

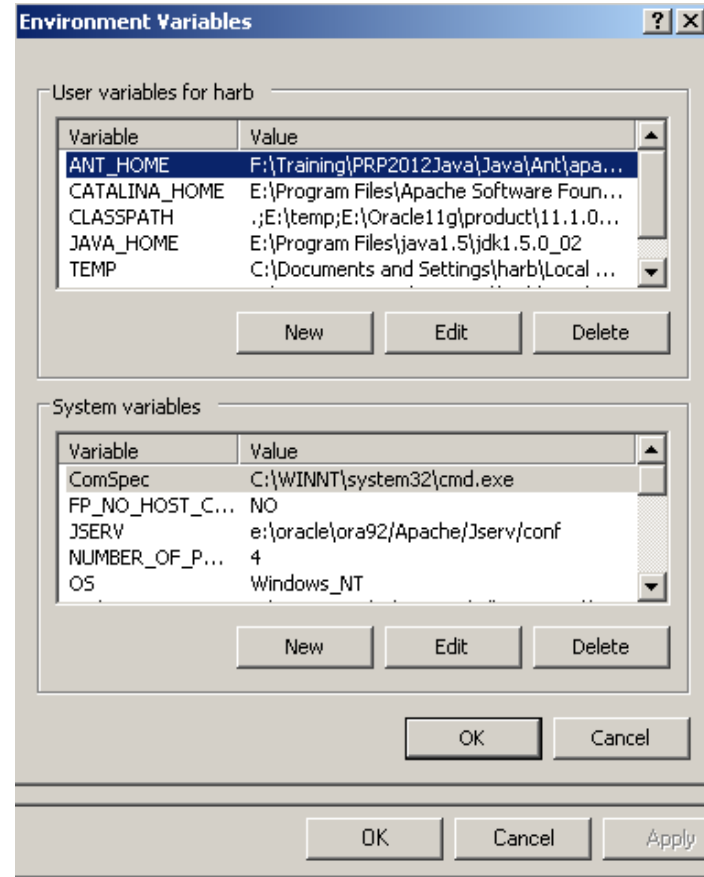
- Before we learn about writing a simple java program, let us understand what preparations are needed for running a java program from console
- We have to define PATH and CLASSPATH parameters and create necessary directories(folders) where we will be storing all our program files

PATH

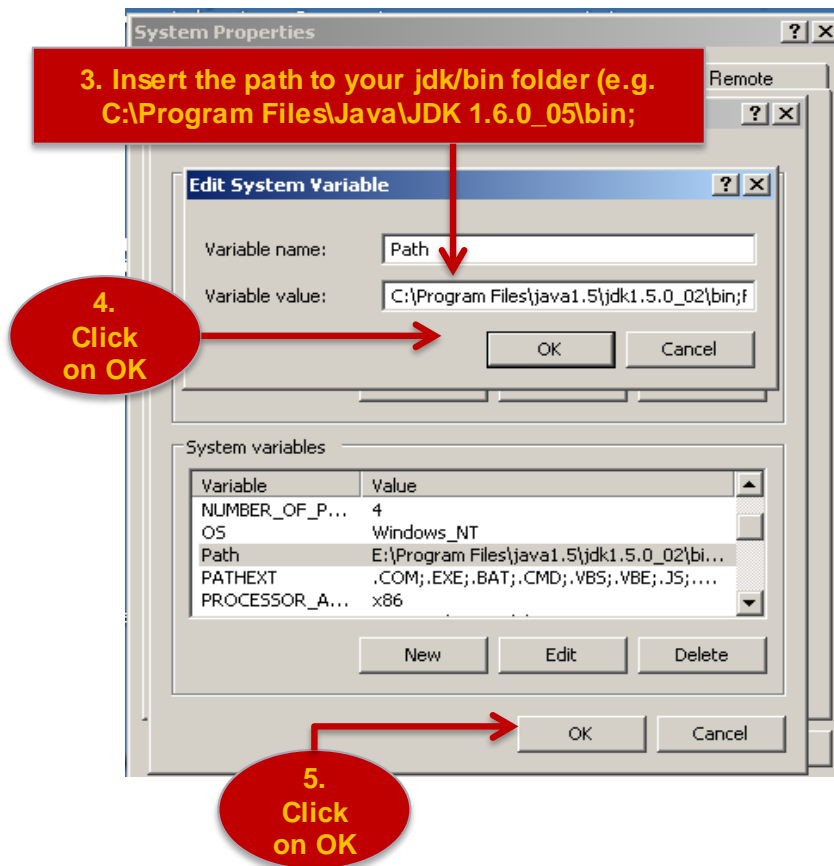
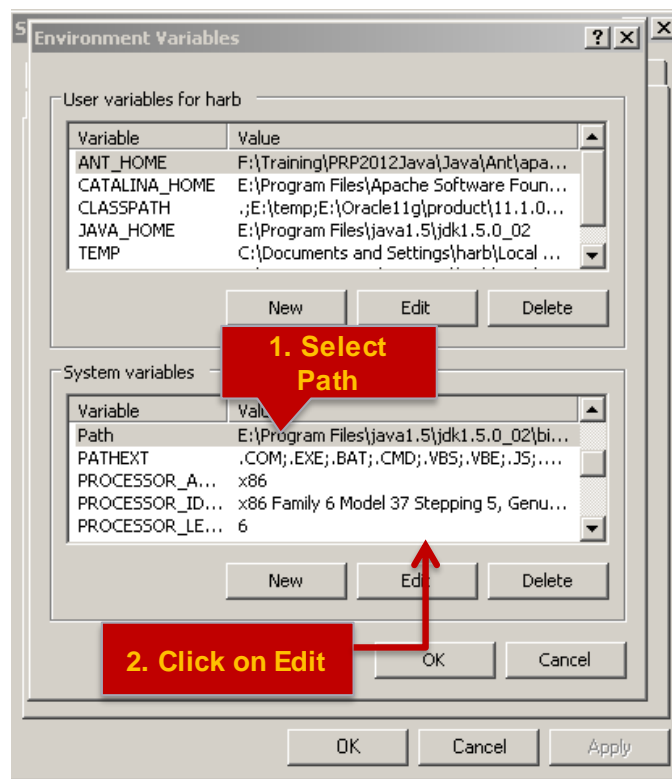
- PATH is an *environmental variable* in DOS(Disk Operating System), Windows and other operating systems like Unix.
- PATH tells the operating system which directories(folders) to search for executable files, in response to commands issued by a user .
- It is a convenient way of executing files without bothering about providing the absolute path to the folder, where the file is located.

How to set PATH ?

1. Right Click My Computer
2. Select Properties
3. You will get to see the Properties Page of My Computer
4. Select Advanced Tab
5. Select Environment Variables
6. You will see Environment Variables Page as displayed here



How to set PATH ? (Contd.).

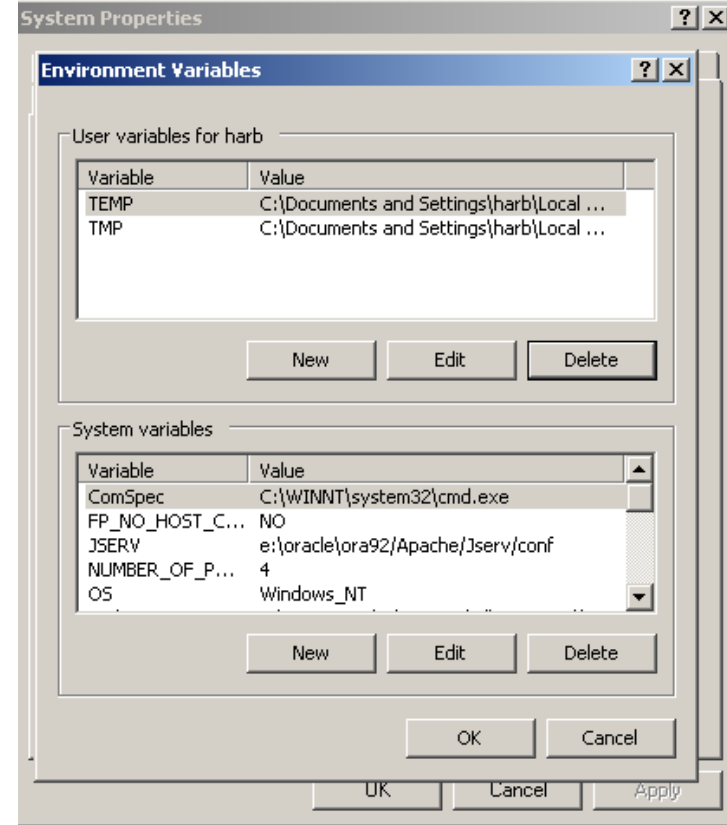


CLASSPATH

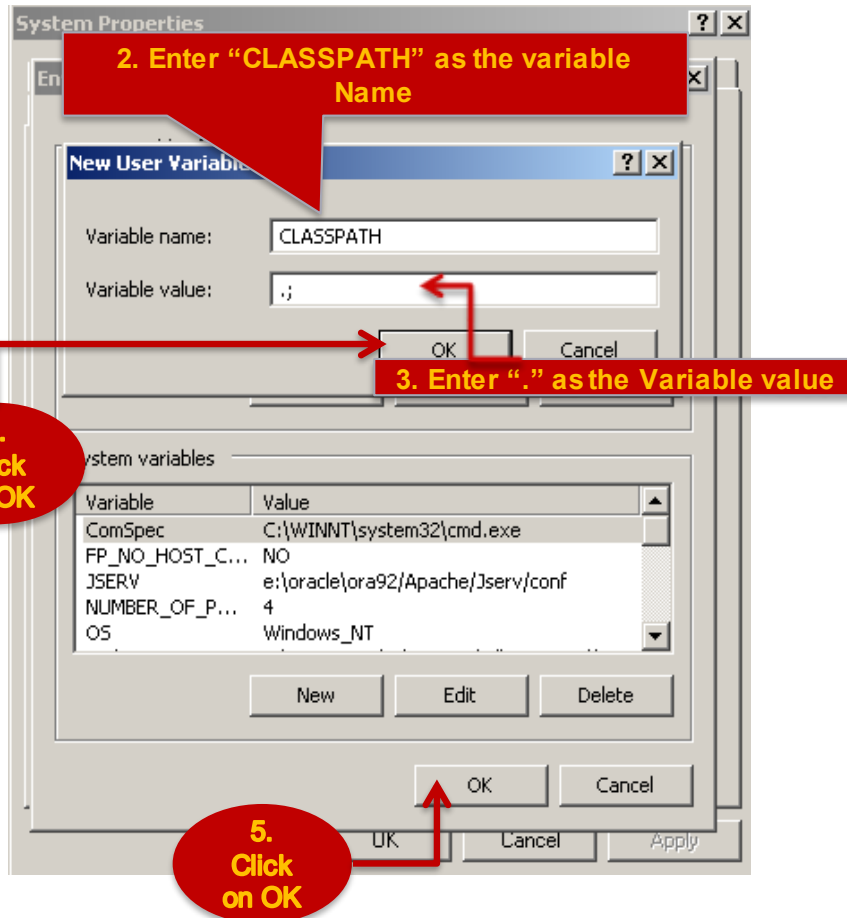
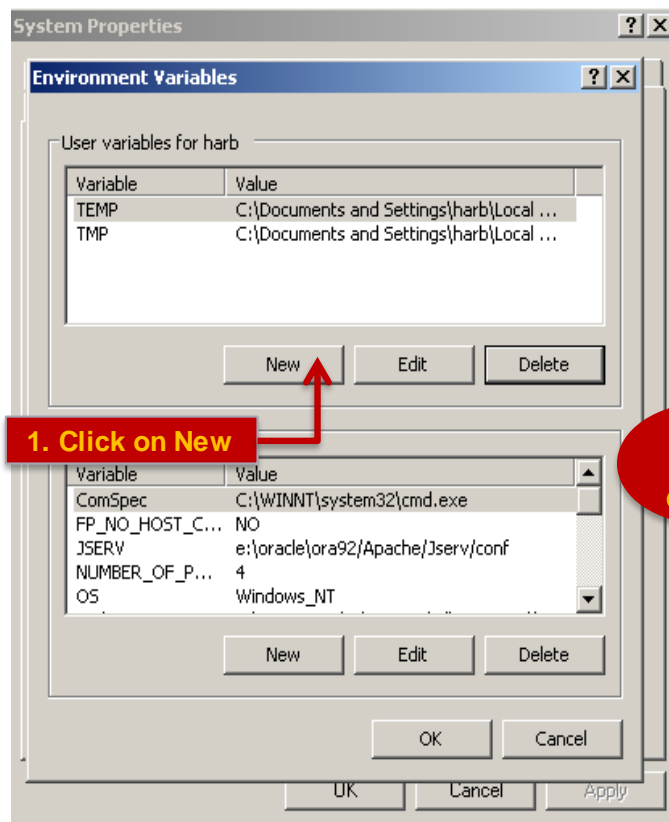
- CLASSPATH is a parameter which tells the JVM or the Compiler, where to locate classes that are not part of Java Development ToolKit(JDK).
- CLASSPATH is set either on command-line or through environment variable.
- CLASSPATH set on command-line is temporary in nature, while the environment variable is permanent.

How to set CLASSPATH ?

1. Right Click My Computer
2. Select Properties
3. You will get to see the Properties Page of My Computer
4. Select Advanced Tab
5. Select Environment Variables
6. You will see Environment Variables Page as displayed here



How to set CLASSPATH ? (Contd.).



A Simple Java Program

Our first Java Program:

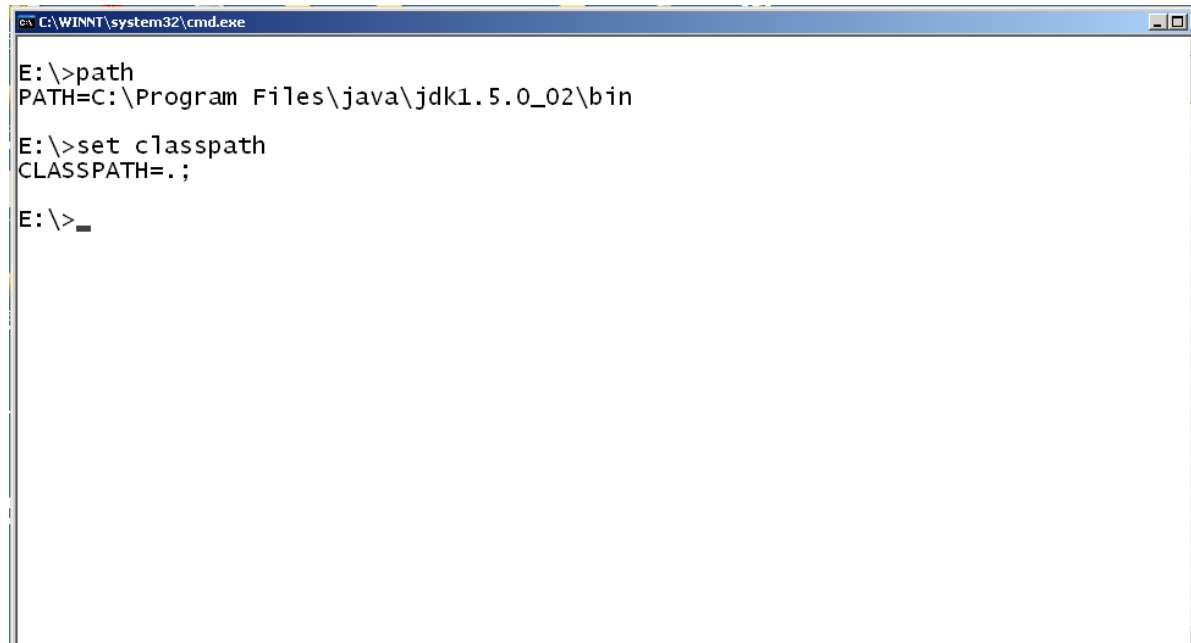
```
public class Welcome {  
    public static void main(String args[]) {  
        System.out.println("Welcome..!");  
    }  
}
```

**This program displays the output "Welcome..!"
on the console**

Create source file : Welcome.java
Compile : javac Welcome.java
Execute : java Welcome

Executing your first Java Program

- Before executing the program, just check whether the PATH and the CLASSPATH parameters are properly set, by typing in the commands as shown in the screen below:



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

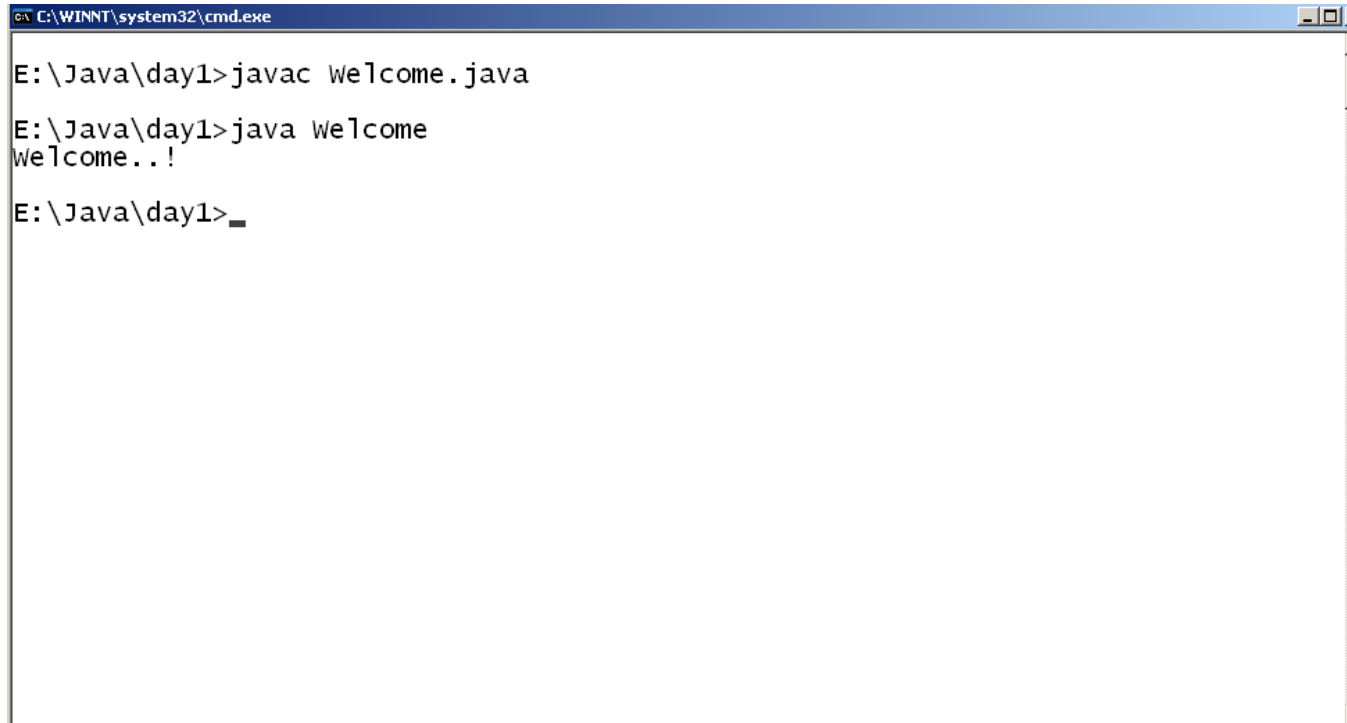
E:\>path
PATH=C:\Program Files\java\jdk1.5.0_02\bin

E:\>set classpath
CLASSPATH=.;

E:\>_
```

Executing your first Java Program (Contd.).

- Now compile and execute your program as given below :



```

C:\WINNT\system32\cmd.exe

E:\Java\day1>javac Welcome.java
E:\Java\day1>java Welcome
Welcome..!
E:\Java\day1>_

```

The screenshot shows a Windows command prompt window titled "C:\WINNT\system32\cmd.exe". The prompt is at "E:\Java\day1>". The user has entered "javac Welcome.java" and pressed Enter. The prompt has moved to the next line. The user has entered "java Welcome" and pressed Enter. The output "Welcome..!" is displayed on the next line. The prompt has moved to the next line, and the user has entered an underscore "_" and pressed Enter.

Quiz

Sample.java file contains class A, B and C. How many .class files will be created after compiling Sample.java? What is your observation?

Sample.java

```
class A {  
    void m1() { }  
}  
  
class B {  
    void m2() { }  
}  
  
class C {  
    void m3() { }  
}
```

Quiz(Contd.).

What will be the result if you try to compile and execute the following program ?

Reason out :

Sample.java

```
class Sample {  
    public static void main() {  
        System.out.println("Welcome");  
    }  
}
```

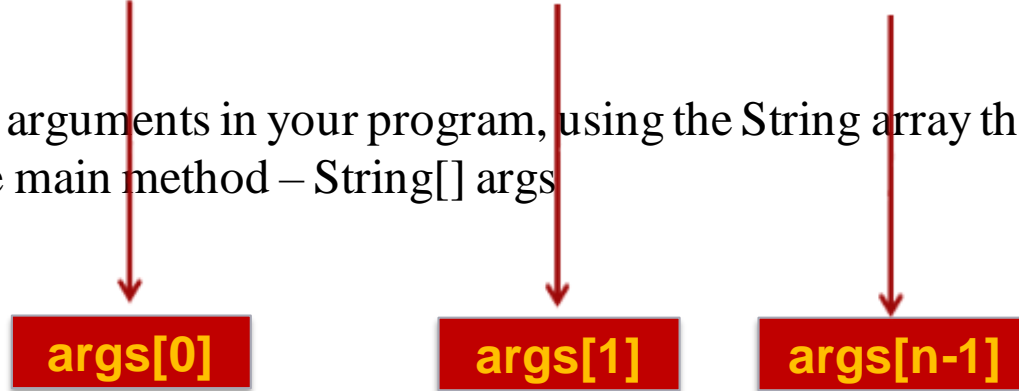
- a. Compilation Error
- b. Runtime Error
- c. The program compiles and executes successfully but prints nothing.
- d. It will print "Welcome"

Accessing command line arguments

When you execute a java program, you can pass command line arguments in the following way :

```
java Simple <argument1> <argument2>...<argument-n>
```

You can access these arguments in your program, using the String array that you have passed as an argument to the main method – String[] args



Passing command line arguments

```
class Simple {  
    static public void main(String[] args) {  
        System.out.println(args[0]);  
    }  
}
```

When we compile the above code successfully and execute it as `Java Simple Wipro`, the output will be

The Wipro logo is displayed within a red rectangular box. The word "Wipro" is written in a bold, yellow, sans-serif font.

Accessing numeric command line arguments

```
class Simple {  
    static public void main(String[] args) {  
        int i1 = Integer.parseInt(args[0]);  
        int i2 = Integer.parseInt(args[1]);  
        System.out.println(i1+i2);  
    }  
}
```

When we compile the above code successfully and execute it as `Java Simple 10 20`, the output will be



30

Finding length of an Array

- How to find the number of command line arguments that a user may pass while executing a java program?
- The answer to the above question lies in `args.length`, where `args` is the String array that we pass to the main method and `length` is the property of the Array Object

Example on Finding length of an Array

```
class FindNumberOfArguments {  
    public static void main(String[ ] args) {  
        int len = args.length;  
        System.out.println(len);  
    }  
}
```

If you compile the above code successfully and execute it as `java FindLength A B C D E F`, the result will be

6

And if you execute it as `java FindLength Tom John Lee`, the result will be

3

Quiz

What will be the result if you try to compile and execute the following code without passing any command line argument?

```
class Sample {  
    public static void main(String [ ] args) {  
        int len = args.length;  
        System.out.println(len);  
    }  
}
```

- a. Compilation Error
- b. Runtime Error
- c. The program compiles and executes successfully but prints nothing.
- d. The program compiles and executes successfully & prints 0.

Good Programming Practices

Naming Conventions

Class Names

- Class names should be **nouns**, in mixed case with the first letter of each internal word capitalized
- Class names should be simple and descriptive
- Eg: class **Student**, class **TestStudent**

Variable Names

- The variables are in mixed case with a lowercase first letter
- Variable names should not start with underscore _ or dollar sign \$ characters, even though both are allowed
- Variable names should be small yet meaningful
- One-character variable names should be avoided except for temporary “throwaway” variables
- Eg: `int y,myWidth;`

Good Programming Practices(Contd.).

Naming Conventions

Method Names

- Methods should be **verbs**, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized
- Eg: void run(), void getColor()

Comments

Block Comments

- Block comments are used to provide descriptions of files, methods, data structures and algorithms
- Block comments may be used at the beginning of each file and before each method

/*

Here is a block comment

*/

Good Programming Practices(Contd.).

Comments

Single line Comment

- Single line comments can be written using // Single line

Number per Line

- One declaration per line is recommended

```
int height;
```

```
int width;
```

is preferred over

```
int height,width;
```

Do not put different types on the same line

```
int height,width[]; // Not recommended
```


The Java API

- An application programming interface(API), in the framework of java, is a collection of prewritten packages, classes, and interfaces with their respective methods, fields and constructors
- The Java API, included with the JDK, describes the function of each of its components
- In Java programming, many of these components are pre-created and commonly used

The Java Buzzwords

- Simple
- Object-Oriented
 - Supports encapsulation, inheritance, abstraction, and polymorphism
- Distributed
 - Libraries for network programming
 - Remote Method Invocation
- Architecture neutral
 - Java Bytecodes are interpreted by the JVM

The Java Buzzwords (Contd.).

- Secure
 - Difficult to break Java security mechanisms
 - Java Bytecode verification
 - Signed Applets
- Portable
 - Primitive data type sizes and their arithmetic behavior specified by the language
 - Libraries define portable interfaces
- Multithreaded
 - Threads are easy to create and use

Language Basics

- Keywords
- Data Types
- Variables
- Operators
- Conditional Statements
- Loops

Java Keywords

abstract	continue	for	new	switch
assert	default	goto	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

Quiz

What will be the result, if we try to compile and execute the following code?

```
class Test {  
    public static void main(String [ ] ar) {  
        int for=2;  
        System.out.println(for);  
    }  
}
```

Primitive Data Types

Data Type	Size (in bits)	Minimum Range	Maximum Range	Default Value (for fields)
byte	8	-128	+127	0
short	16	-32768	+32767	0
int	32	-2147483648	+2147483647	0
long	64	-9223372036854775808	+9223372036854775807	0L
float	32	1.40E-45	3.40282346638528860e+38	0.0f
double	64	4.94065645841246544e-324d	1.79769313486231570e+308d	0.0d
char	16		0 to 65,535	'\u0000'
boolean	1	NA	NA	false

Quiz

What will be the result, if we try to compile and execute the following code?

```
class Test {  
    public static void main(String [ ] ar) {  
        byte b=128;  
        System.out.println(b) ;  
    }  
}
```


Quiz(Contd.).

What will be the result, if we try to compile and execute the following code?

```
class Test {  
    public static void main(String ar[]) {  
        float f=1.2;  
        boolean b=1;  
        System.out.println(f);  
        System.out.println(b);  
    }  
}
```

Quiz(Contd.).

What will be the result, if we try to compile and execute the following code?

```
class Test {  
    public static void main(String ar[]) {  
        double d=1.2D;  
        System.out.println(d) ;  
    }  
}
```

Quiz(Contd.).

What will be the result, if we try to compile and execute the following code?

```
class Test {  
    public static void main(String [ ] ar) {  
        int a=10,b=017,c=0X3A;  
        System.out.println(a+", "+b+", "+c) ;  
    }  
}
```

Quiz(Contd.).

What will be the result, if we try to compile and execute the following code?

```
class Test {  
    public static void main(String [ ] args) {  
        int 9A=10;  
        System.out.println(9A);  
    }  
}
```

Quiz(Contd.).

What will be the result, if we try to compile and execute the following code?

```
class Test {  
    public static void main(String [ ] args) {  
        int x;  
        System.out.println(x) ;  
    }  
}
```

Quiz

1. Match the following table:

DATA TYPES	SIZE(bytes)
char	4
byte	2
int	1
double	8

Types of Variables

The Java programming language defines the following kinds of Variables:

- Local Variables
 - Tied to a method
 - Scope of a local variable is within the method
- Instance Variables (Non-static)
 - Tied to an object
 - Scope of an instance variable is the whole class
- Static Variables
 - Tied to a class
 - Shared by all instances of a class

Operators

- Java provides a set of operators to manipulate operations.
- Types of operators in java are,
 - Arithmetic Operators
 - Unary Operator
 - Relational Operators
 - Logical Operators
 - Simple Assignment Operator
 - Bitwise Operators

Arithmetic Operators

The following table lists the arithmetic operators

Operator	Description	Example
+	Addition	$A + B$
-	Subtraction	$A - B$
*	Multiplication	$A * B$
/	Division	A/B
%	Modulus	$A\%B$

Arithmetic Operators - Example

/* Example to understand Arithmetic operator */

```
class Sample{  
    public static void main(String[ ] args){  
        int a = 10;  
        int b = 3;  
        System.out.println("a + b = " + (a + b) );  
        System.out.println("a - b = " + (a - b) );  
        System.out.println("a * b = " + (a * b) );  
        System.out.println("a / b = " + (a / b) );  
        System.out.println("a % b = " + (a % b) );  
    }  
}
```

Output:

```
a + b = 13  
a - b = 7  
a * b = 30  
a / b = 3  
a % b = 1
```

Unary Operators

The following table lists the unary operators

Operator	Description	Example
+	Unary plus operator	+A
-	Unary minus operator	-A
++	Increment operator	++A or A++
--	Decrement operator	--A or A--

Unary Operator - Example

/ Example for Unary Operators*/*

```
class Sample{  
    public static void main(String args[]) {  
        int a = 10;  
        int b = 20;  
  
        System.out.println("++a    = " + (++a) );  
        System.out.println("--b    = " + (--b) );  
    }  
}
```

Output:

++a = 11
--b = 19

Quiz

What will be the result, if we try to compile and execute the following code?

```
class Test {  
    public static void main(String [ ] args) {  
        int x=10;  
        int y=5;  
        System.out.println(++x+(++y) );  
    }  
}
```

Relational Operators

The following table lists the relational operators

Operator	Description	Example
==	Two values are checked, and if equal, then the condition becomes true	(A == B)
!=	Two values are checked to determine whether they are equal or not, and if not equal, then the condition becomes true	(A != B)
>	Two values are checked and if the value on the left is greater than the value on the right, then the condition becomes true.	(A > B)
<	Two values are checked and if the value on the left is less than the value on the right, then the condition becomes true	(A < B)
>=	Two values are checked and if the value on the left is greater than equal to the value on the right, then the condition becomes true	(A >= B)
<=	Two values are checked and if the value on the left is less than equal to the value on the right, then the condition becomes true	(A <= B)

Comparing Strings

- For comparing Strings instead of using == operator, use equals method

```
String s1="hello";
```

```
String s2="Wipro";
```

```
System.out.println(s1.equals(s2)); ➔ false
```

Relational Operators - Example

/* Example to understand Relational operator */

```
class Sample{  
    public static void main(String[] args){  
        int a = 10;  
        int b = 20;  
  
        System.out.println("a == b = " + (a == b) );  
        System.out.println("a != b = " + (a != b) );  
        System.out.println("a > b = " + (a > b) );  
        System.out.println("a < b = " + (a < b) );  
        System.out.println("b >= a = " + (b >= a) );  
        System.out.println("b <= a = " + (b <= a) );  
    }  
}
```

Output:

a == b = false
a != b = true
a > b = false
a < b = true
b >= a = true
b <= a = false

Logical Operators

The following table lists the logical operators

Operator	Description	Example
&&	This is known as Logical AND & it combines two variables or expressions and if and only if both the operands are true, then it will return true	(A && B) is false
	This is known as Logical OR & it combines two variables or expressions and if either one is true or both the operands are true, then it will return true	(A B) is true
!	Called Logical NOT Operator. It reverses the value of a Boolean expression	!(A && B) is true

Logical Operators - Example

/* Example to understand logical operator */

```
class Sample{  
    public static void main(String[] args) {  
        boolean a = true;  
        boolean b = false;  
        System.out.println("a && b = " + (a&&b) );  
        System.out.println("a || b = " + (a||b) );  
        System.out.println("!(a && b) = " + !(a && b) );  
    }  
}
```

Output:

a && b =
false
a || b = true
!(a && b) =
true

Simple Assignment Operator

= Simple assignment operator

Which assigns right hand side value to left hand side variable

Ex:

```
int a;  
a = 10;
```

Shift Operators << and >>

- The shift operators(<< and >>) shift the bits of a number to the left or right, resulting in a new number.
- They are used only on integral numbers(and not on floating point numbers, i.e. decimals).
- The right shift operator(>>) is used to divide a number in the multiples of 2, while the left shift operator(<<) is used to multiply a number in the multiples of 2.

Right Shift Operator >>

- Let us understand the use of right shift operator with the following example :

```
int x = 16;
```

```
x = x >> 3;
```

- When we apply the right shift operator >>, the value gets divided by 2 to the power of number specified after the operator. In this case, we have 3 as the value after the right shift operator. So, 16 will be divided by the value 2 to the power of 3, which is 8.
- The result is 2.

Right Shift Operator >> (Contd.).

- When we represent 16 in binary form, we will get the following binary value :

0 1 0 0 0 0

- When we apply >> which is the right shift operator, the bit represented by 1 moves by 3 positions to the right(represented by the number after the right shift operator).
- After shifting the binary digit 1, we will get :

0 1 0

↑ ↑ ↑ ↑

- $x = 2$

Right Shift Operator >> Demo

```
class ShiftExample1 {  
    public static void main(String[] args) {  
        int x = 16;  
        System.out.println("The original value of x is "+x);  
        x = x >> 3;  
        System.out.println("After using >> 3, the new value is  
"+x);  
    }  
}
```

**The original value of x is 16
After using >> 3, the new
value is 2**

Left Shift Operator <<

- Let us understand the use of left shift operator with the following example :

```
int x = 8;
```

```
x = x << 4;
```

- When we apply the left shift operator <<, the value gets multiplied by 2 to the power of number specified after the operator. In this case, we have 4 as the value after the left shift operator. So, 8 will be multiplied by the value 2 to the power of 4, which is 16.
- The result is 128.

Left Shift Operator << (Contd.).

- When we represent 8 in binary form, we will get the following binary value :

0 1 0 0 0

- When we apply << which is the left shift operator, the bit represented by 1 moves by 4 positions to the left (represented by the number after the right shift operator).
- After shifting the binary digit 1, we will get :

0 1 0 0 0 0 0 0 0

↑ ↑ ↑ ↑ ↑

- X=128

Left Shift Operator << Demo

```
class ShiftExample2 {  
    public static void main(String[] args) {  
        int x =8;  
        System.out.println("The original value of x is "+x);  
        x = x << 4;  
        System.out.println("After using << 4, the new value is "+x);  
    }  
}
```

**The original value of x is 8
After using << 4, the new value
is 128**

Bitwise Operators

The bitwise operators take two bit numbers, use OR/AND to determine the result on a bit by bit basis.

The 3 bitwise operators are :

- $\&$ (which is the bitwise AND)
- $|$ (which is the bitwise inclusive OR)
- \wedge (which is the bitwise exclusive OR)

Bitwise & (AND) operator

The & operator compares corresponding bits between two numbers and if both the bits are 1, only then the resultant bit is 1. If either one of the bits is 0, then the resultant bit is 0.

Example :

```
int x = 7;
```

```
int y = 9;
```

What will be x & y ?

7 -> 0 1 1 1

9 -> 1 0 0 1

0 0 0 1

x & y results in 1.

Bitwise & Operator Demo

```
class BitwiseExample1 {  
    public static void main(String[] args) {  
        int x = 7;  
        int y = 9;  
        int z = x & y;  
        System.out.println("z = "+z);  
    }  
}
```

Output :
z = 1

Bitwise | (inclusive OR) operator

The | operator will set the resulting bit to 1 if *either* one of them is 1. It will return 0 only if both the bits are 0.

Example :

```
int x = 5;
```

```
int y = 9;
```

What will be $x | y$?

```
5 -> 0 1 0 1
```

```
9 -> 1 0 0 1
```

```
-----
```

```
1 1 0 1
```

```
-----
```

$x | y$ results in 13.

Bitwise | Operator Demo

```
class BitwiseExample2 {  
    public static void main(String[] args) {  
        int x = 5;  
        int y = 9;  
        int z = x | y;  
        System.out.println("z = "+z);  
    }  
}
```

Output :
z = 13

Bitwise ^ (exclusive OR) operator

The ^ operator compares two bits to check whether these bits are different. If they are different, the result is 1. Otherwise, the result is 0. This operator is also known as XOR operator.

Example :

```
int x = 5;
```

```
int y = 9;
```

What will be $x \wedge y$?

```
5 -> 0 1 0 1
```

```
9 -> 1 0 0 1
```

```
-----
```

```
    1 1 0 0
```

```
-----
```

$x \wedge y$ results in 12.

Bitwise ^ Operator Demo

```
class BitwiseExample3 {  
    public static void main(String[] args) {  
        int x = 5;  
        int y = 9;  
        int z = x ^ y;  
        System.out.println("z = "+z);  
    }  
}
```

Output :
z = 12

Quiz

What will be the output for the below code ?

```
public class Sample {  
    public static void main() {  
        int i_val = 10, j_val = 20;  
        boolean chk;  
        chk = i_val < j_val;  
        System.out.println("chk value: "+chk);  
    }  
}
```

Summary

In this session, you were able to :

- Learnt about how to write a Java Program
- How to access command line arguments
- Good Programming Practices
- Various type of Operators in Java and their Usage





Thank You