What is Java :

Java is a **programming language** and a **platform**. Java is a high level, robust, object-oriented and secure programming language. It is a case-sensitive language.

Java was developed by *Sun Microsystems* (which is now the subsidiary of Oracle) in the year 1995. *James Gosling* is known as the father of Java.

History of Java :

Founded by	: SUN Mc Systems (Oracle Corporation)
Author	: James Gosling
Objective	: To prepare simple electronic consumer goods.
Project	: Green
First Version	: JDK 1.0 (1996, Jan-23rd)
Strong Features	: Object-oriented, Platform Independent, Robust, Portable, Dynamic, Secure.....

Features of Java :

Java language has the following features :

1) Simple:

Java is simple programming language, because,

a) Java applications will take less memory and less execution time.

b) Java has removed all most all the confusion oriented features like pointers, multiple inheritance,.....

c) Java is using all the simplified syntaxes from C and C++.

2) Object Oriented:

Java is an object oriented programming language, because, JAVA is able to store data in the form of Objects only.

3) Platform Independent:

Java is platform independent programming Language, because, Java allows its applications to compile on one operating system and to execute on another operating system.

Java is platform independent because of its byte code (.class file)

4) Architecture Neutral:

It means that even if you write and compile a program on one hardware configuration and try to run on some other hardware configuration still it will execute.

5) Portable:

Java is a portable programming language, because, JAVA is able to run its applications under all the operating systems and under all the hardware Systems.

6) Robust:

Java is Robust programming language, because,

1) Java is having very good memory management system in the form of heap memory Management System, it is a dynamic memory management system, it allocates and deallocates memory for the objects at runtime.

2) JAVA is having very good Exception Handling mechanisms, because, Java has provided very good predefined library to represent and handle almost all the frequently generated exceptions in java applications.

7) Secure:

Java is very good Secure programming language, because,

1) JAVA has provided an implicit component inside JVM in the form of "Security Manager" to provide implicit security.

2) This makes Java popular for banking applications.

8) Dynamic:

If any programming language allows memory allocation for primitive data types at RUNTIME then that programming language is called as Dynamic Programming Language.

JAVA is a dynamic programming language, because, JAVA allows memory allocation for primitive data types at RUNTIME.

9) Distributed:

By using JAVA we are able to prepare two types of applications

1) Standalone Applications

2) Distributed Applications

1) Standalone Applications:

If we design any java application without using client-Server arch then that java application is called as Standalone application.

2) Distributed Applications:

If we design any java application on the basis of client-server arch then that java application is called as Distributed application.

To prepare Distributed applications, JAVA has provided a seperate module that is "J2EE/JAVA EE".

10) Multi Threaded:

Thread is a flow of execution to perform a particular task.

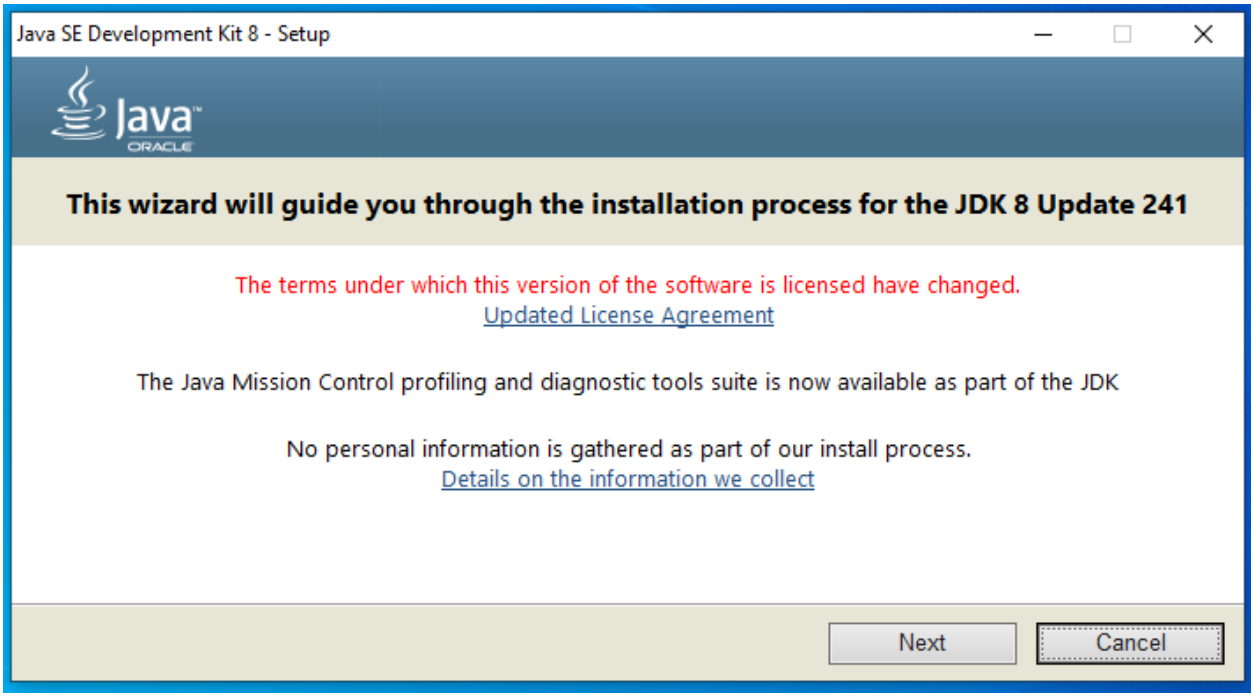
Multithreaded means ability to allow more than one thread to execute application, It follows parallel execution, it will reduce execution time, it will improve application performance.

JAVA is following Multi Thread Model, JAVA is able to provide very good environment to create and execute more than one thread at a time, due to this reason, JAVA is Multi threaded Programming Language.

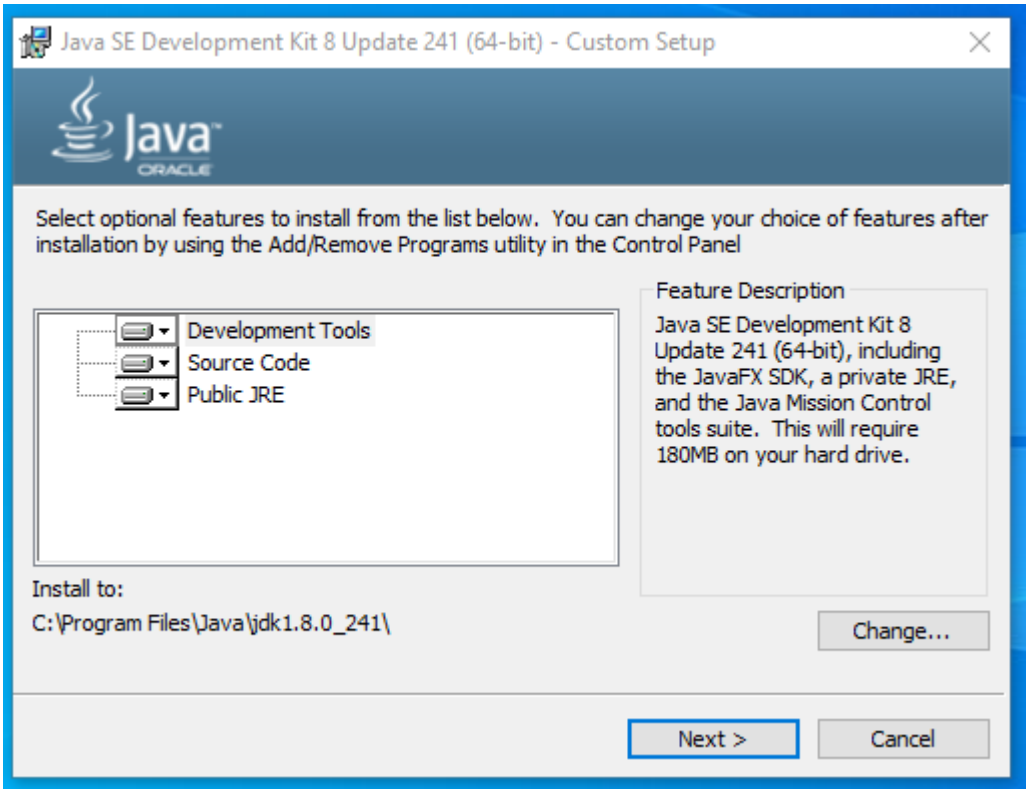
11) High Performance:

JAVA is high performance programming language due to its rich set of features like Platform independent, Arch Neutral, Portable, Robust, Dynamic,.....

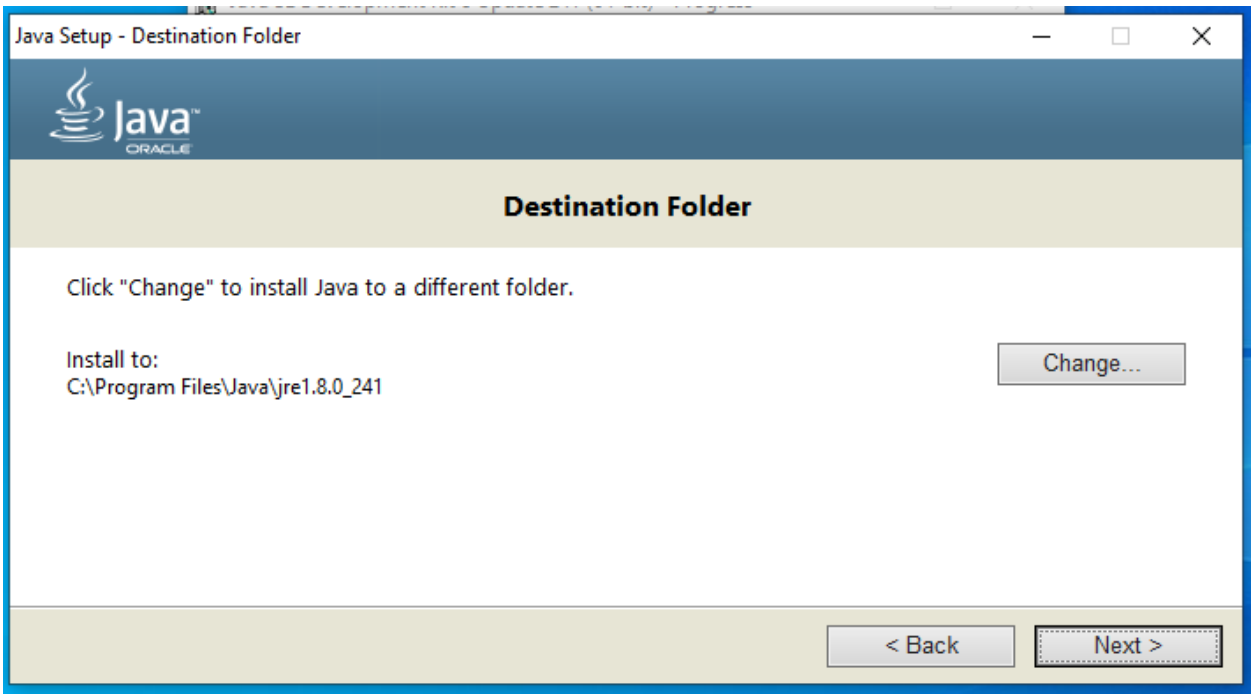
How to install java and set path?



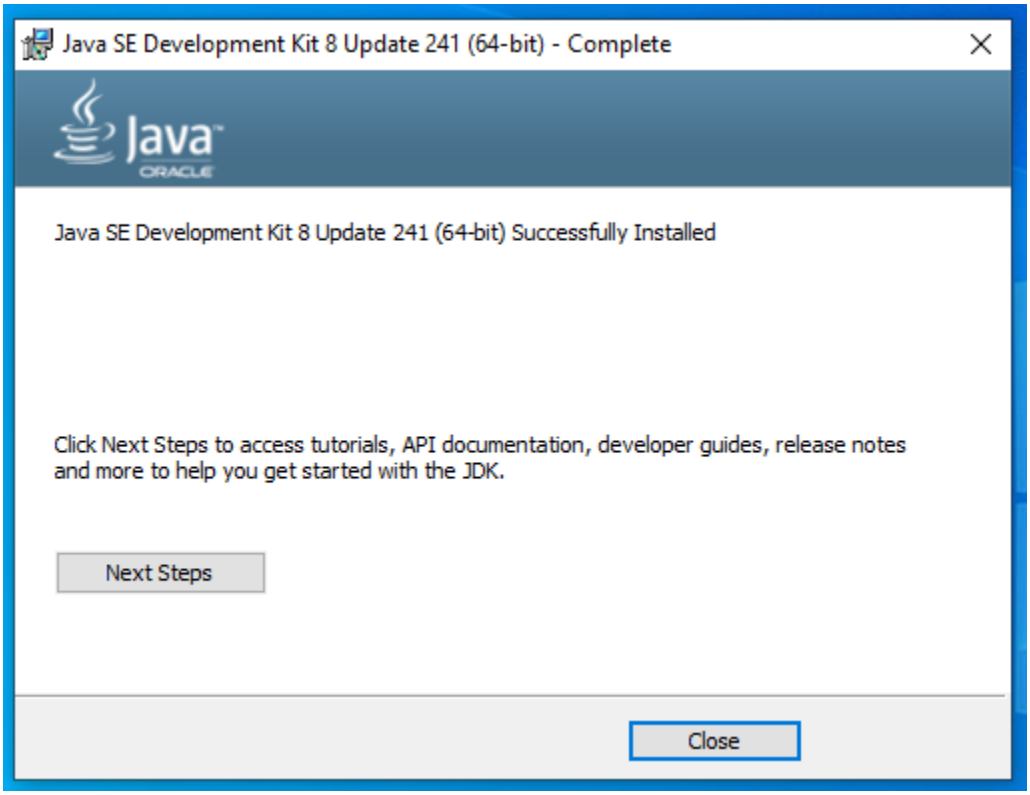
Click on Next button.



Click on Next button



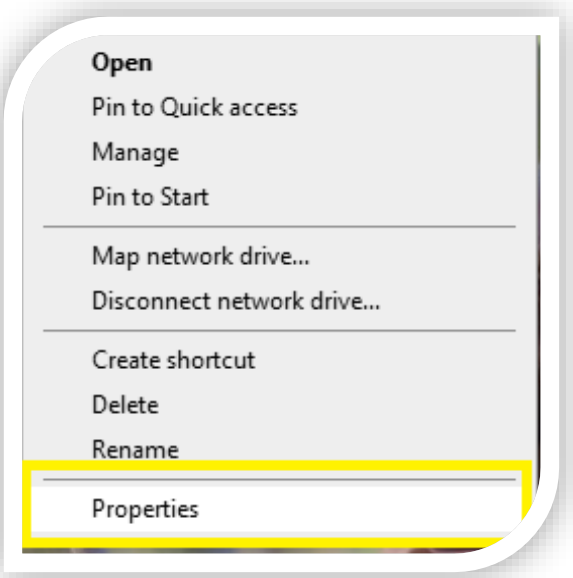
Click on next button



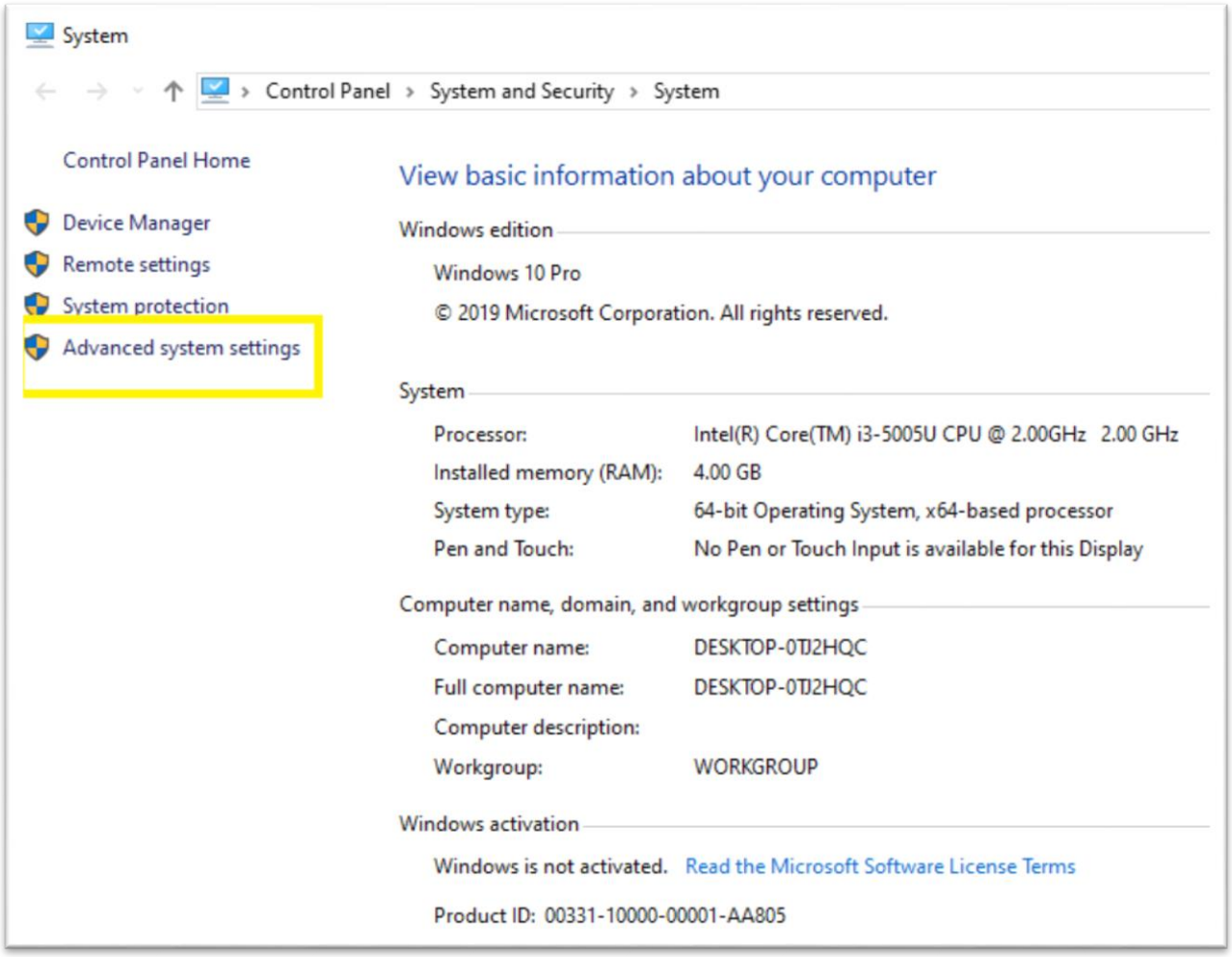
Click on close button

How to set class path in java.

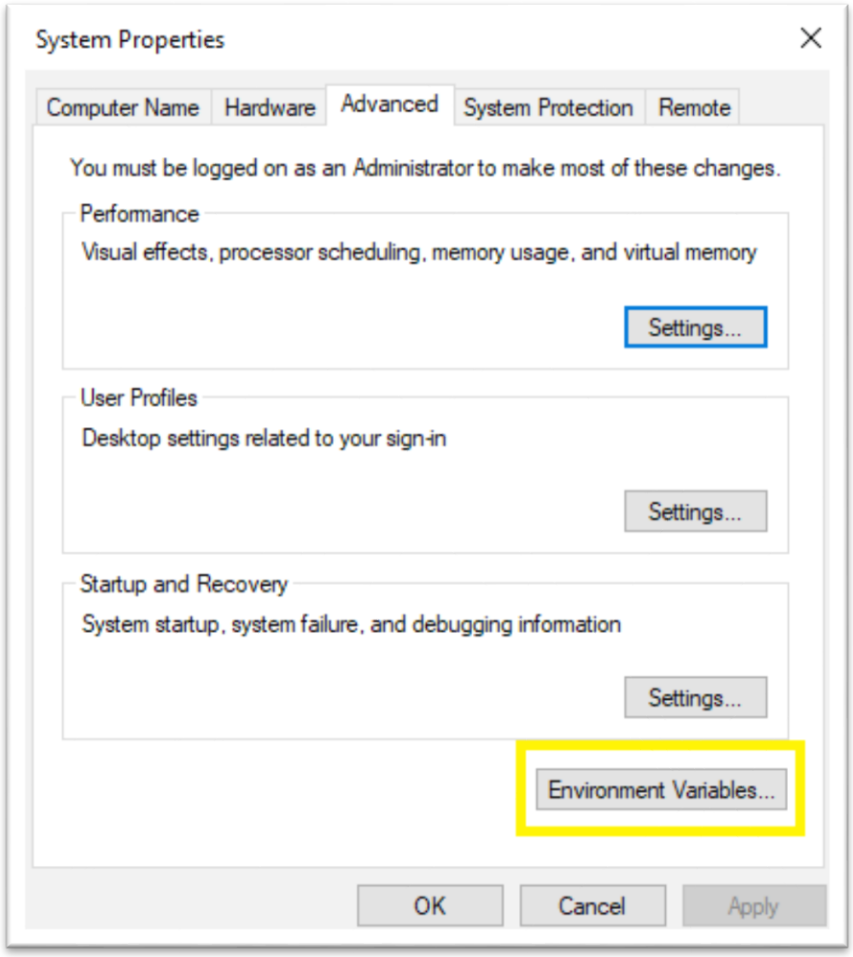
Windows 10 Operating system, click on My Computer then Right click->select properties.



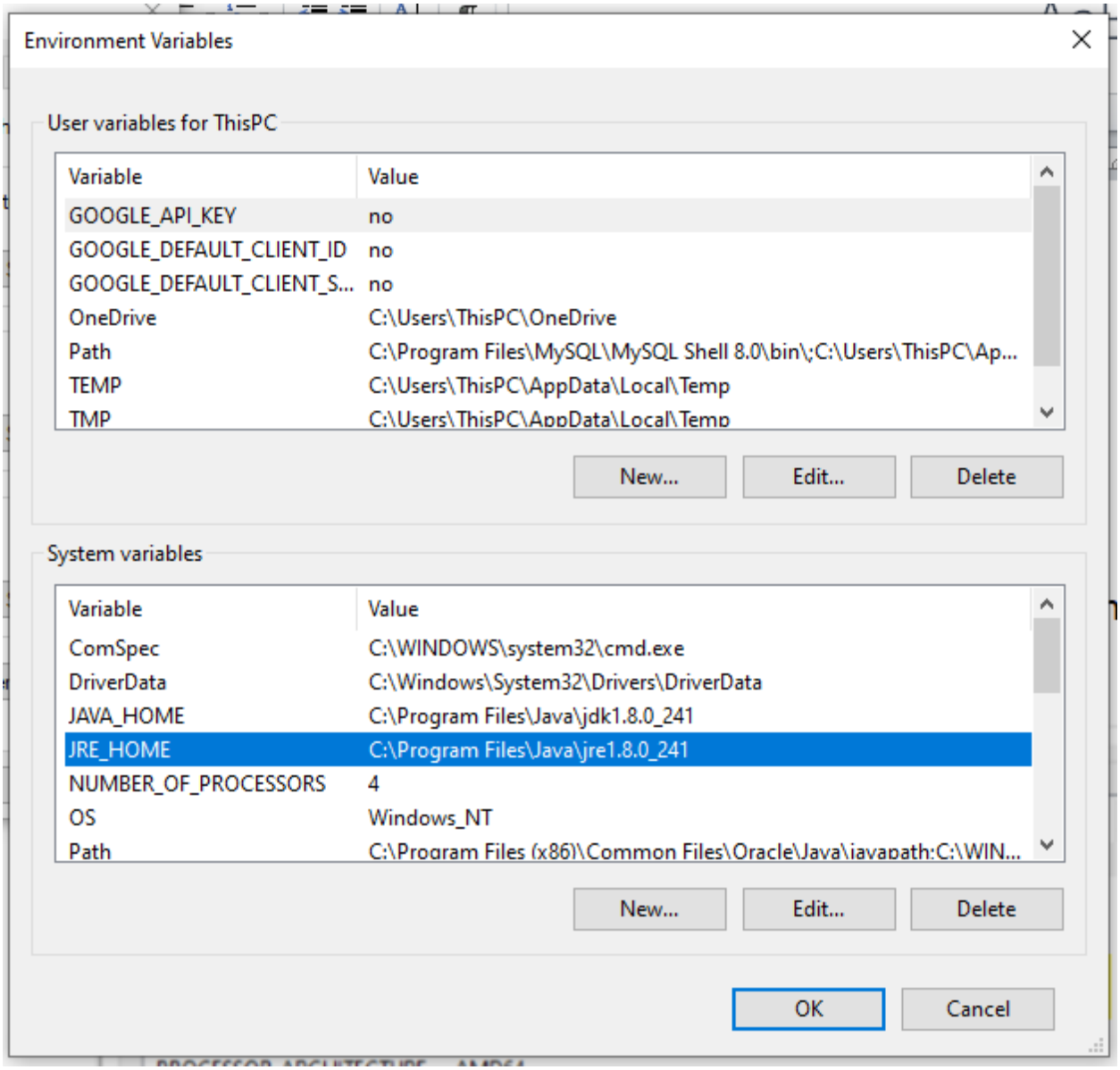
On left hand side menu-> Select Advanced System Setting option.



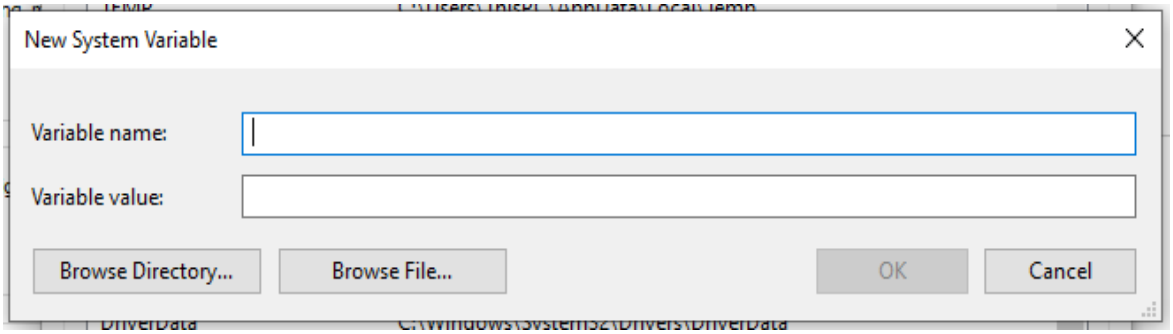
Click on Environment variable button at displayed on bottom off screen.



Go to System variable section, set JAVA_HOME and JRE_HOME



Click on New button



Go to c drive and copy path

C:\Program Files\Java\jdk1.8.0_241

Type

Variable name as JAVA_HOME

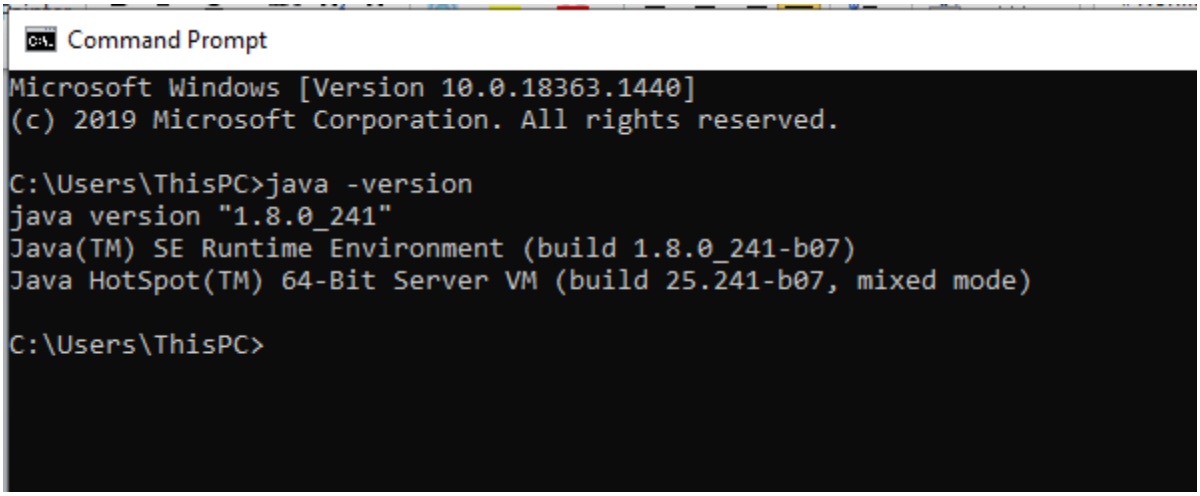
Variable value as C:\Program Files\Java\jdk1.8.0_241

Same apply for JRE also

Variable name as JRE_HOME

Variable value as C:\Program Files\Java\jre1.8.0_241

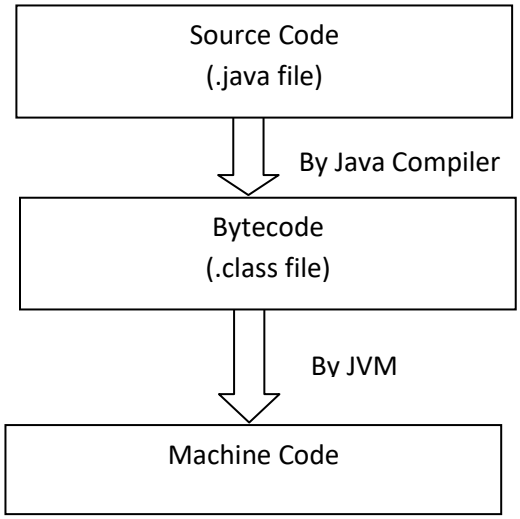
Then go to command prompt and type



Java Program Execution:

Any java program execution has the following set of steps:

1) Whatever code we write using IDE, Notepad, etc, it is called as the source code(.java file).
2) This source code is converted to bytecode(.class file) using compiler. This bytecode is machine-independent and hence makes java language as platform independent.
3) Further this bytecode is converted to machine code by the JVM which is custom-built for every operating system. . This machine code is machine/OS specific.



4) Due to the two-step execution process described above, a java program is independent of the target operating system. However, because of the same, the execution time is way more than a similar program written in a compiled platform-dependent program.

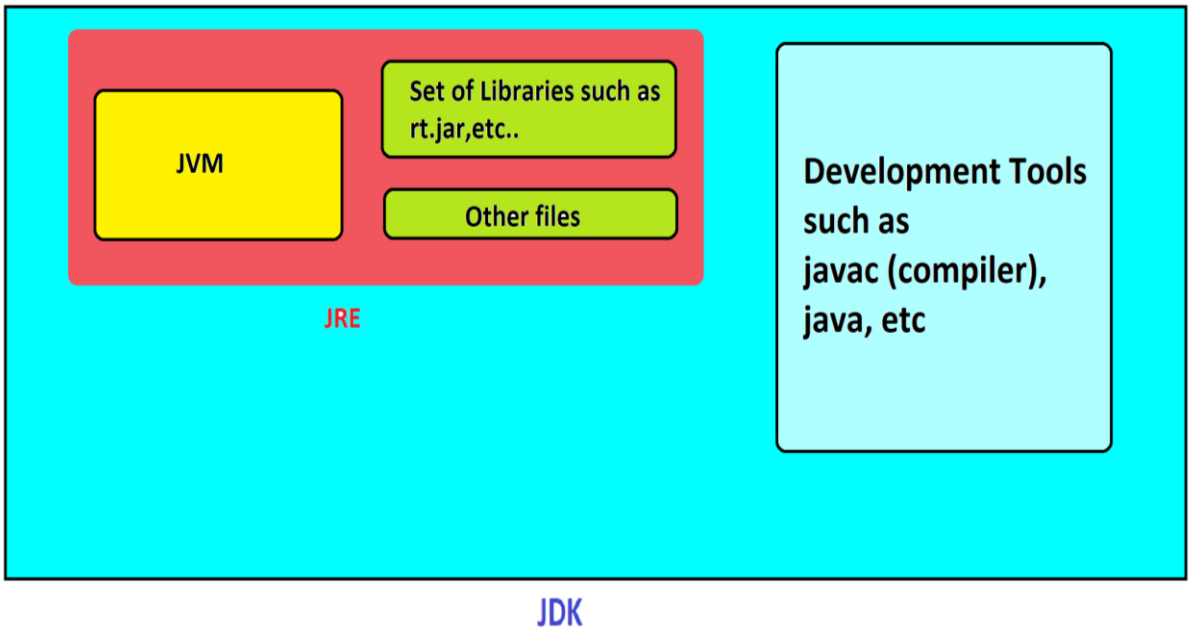
JVM JRE and JDK :

JDK:

Java Development Kit (JDK) is a software development environment used for developing Java applications. It includes the Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), and other tools needed in Java development.

JRE:

JRE stands for **Java Runtime Environment**. The Java Runtime Environment provides the minimum requirements for executing a Java application; it consists of the Java Virtual Machine (JVM), core classes, and supporting files like rt.jar files.



JVM:

JVM stands for **Java Virtual Machine**. It is a part of JRE. JVM is responsible to load and run the java program.

JVM Runs Java Byte Code by creating 5 Identical Runtime Areas to execute Class Members.

- Class Loader Sub System
- Memory Management System

• PC-Registers
• Execution Engine
• Native Methods Stack

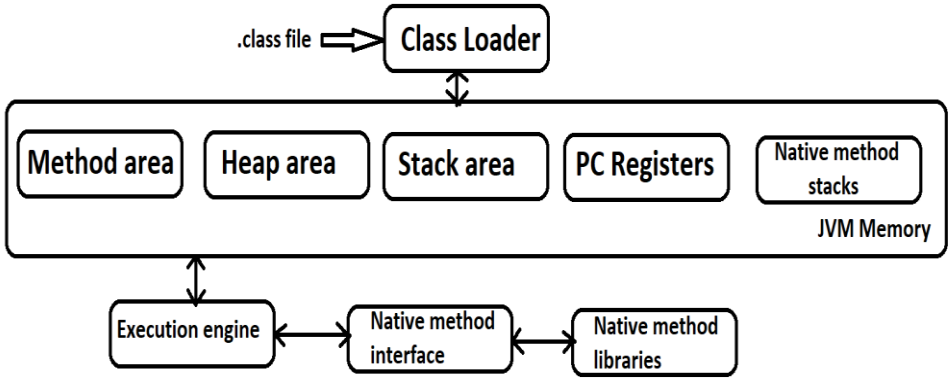


Fig. JVM Arch

a) Class Loader:

Class Loader is used to load the .class file in the memory.

b) Memory Management System:

While Loading and Running a Java Program JVM required Memory to Store Several Things Like Byte Code, Objects, Variables, Etc.

Total JVM Memory organized in the following 5 Categories:

- Method Area
- Heap Area OR Heap Memory
- Java Stack Area
- PC Registers Area
- Native Method Stacks Area

Method Area:

- Total Class Level Binary Information including Static Variables Stored in Method Area.

Heap Area:

- All Objects and corresponding Instance Variables will be stored in the Heap Area.

Java Stack Area:

• All Method Calls and corresponding Local Variables, Intermediate Results will be stored in the Stack.

c) PC Registers Area

• For Every Thread a Separate PC Register will be created at the Time of Thread Creation.
• PC Registers contains Address of Current executing Instruction.
• Once Instruction Execution Completes Automatically PC Register will be incremented to Hold Address of Next Instruction.

d) Execution Engine

This is the Central Component of JVM.
• Execution Engine is Responsible to Execute Java Class Files.
• It contain information about JIT (Just in time) compiler and interpreter

e) Native Methods Stack

• For Every Thread JVM will Create a Separate Native Method Stack.
• All Native Method Calls invoked by the Thread will be stored in the corresponding Native Method Stack

f) Java Native Interface (JNI):

JNI Acts as Bridge (Mediator) between Java Method Calls and corresponding Native Libraries.

g) Java Native Library:

Java Native Library is the collection of Native methods which are required in java.
Native method is a method declared in java, but, implemented in non java programming languages like C , C++,...

Identifiers:

A name in java program is called identifier. It may be class name, method name, Variable name and label name.

Rules to define java identifiers:

Rule 1: The only allowed characters in java identifiers are:

1) a to z
2) A to Z
3) 0 to 9
4) _ (underscore)
5) $

Rule 2: If we are using any other character we will get compile time error.

Example: Total#----------invalid

Rule 3: identifiers are not allowed to starts with digit.

Example: 123ABC---------invalid

Rule 4: java identifiers are case sensitive of course java language itself treated as case sensitive language.

Rule 5: There is no length limit for java identifiers but it is not recommended to take more than 15 lengths.

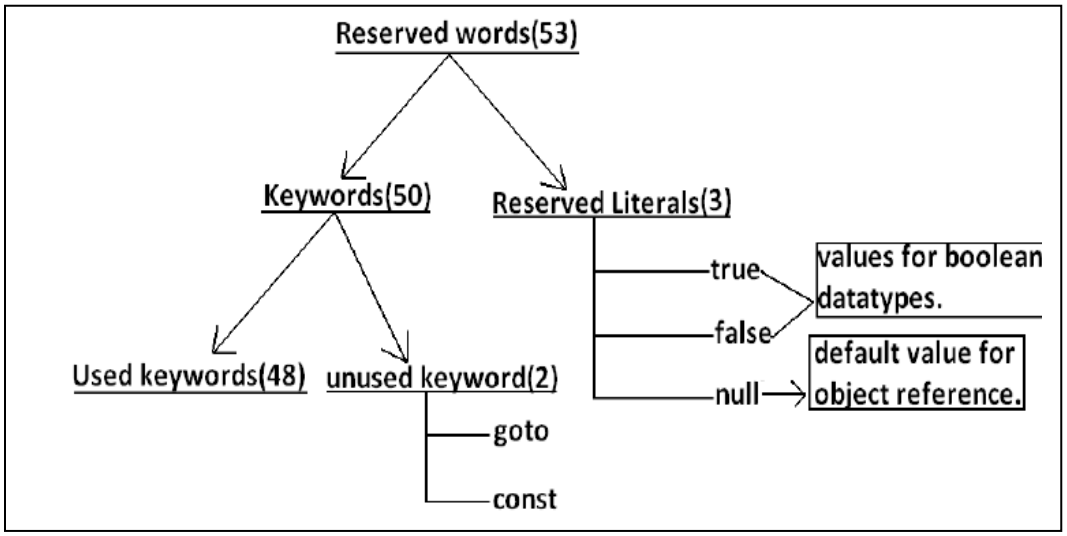
Rule 6: We can't use reserved words as identifiers.

Example: int if=10; ----------invalid

Rule 7: All predefined java class names and interface names we use as identifiers. Even though it is legal to use class names and interface names as identifiers but it is not a good programming practice.

Reserved word and keywords:

In java some identifiers are reserved to associate some functionality or meaning such type of reserved identifiers are called reserved words.



Reserved words for data types: (8)

1) byte
2) short
3) int
4) long
5) float
6) double
7) char
8) Boolean

Reserved words for flow control:(11)

1) if
2) else
3) switch
4) case
5) default
6) for
7) do
8) while
9) break
10) continue
11) return

Keywords for modifiers:(11)

1) public
2) private
3) protected
4) static
5) final
6) abstract
7) synchronized
8) native
9) strictfp(1.2 version)
10) transient
11) volatile

Keywords for exception handling:(6)

1) try
2) catch
3) finally
4) throw
5) throws
6) assert(1.4 version)

Class related keywords:(6)

1) class
2) package
3) import
4) extends
5) implements
6) interface

Object related keywords:(4)

1) new
2) instanceof
3) super
4) this

Void return type keyword:

1) void

Data types:

Java every variable has a type, every expression has a type and all types are strictly define more over every assignment should be checked by the compiler by the type compatibility hence java language is considered as strongly typed programming language.

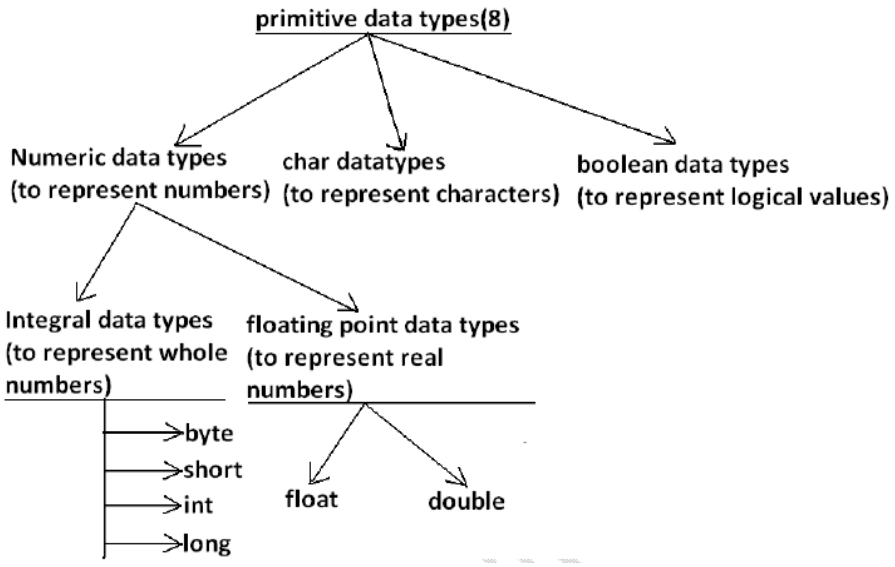
Datatypes are of two type:

1) Primative Data type (Already defined in Java)

- This are the data type which are predefined in Java
- For ex. Boolean, int, long, etc

2) Non Primative Data type

- Programmer defines it. Not already defined in Java, except String
- For ex. Student, Employee,etc.



Data type	Size	Range	Corresponding Wrapper class	Default value
byte	1 byte	-2^7 to 2^7-1 (-128 to 127)	Byte	0
short	2 bytes	-2^{15} to $2^{15}-1$ (-32768 to 32767)	Short	0
int	4 bytes	-2^{31} to $2^{31}-1$ (-2147483648 to 2147483647)	Integer	0
long	8 bytes	-2^{63} to $2^{63}-1$	Long	0
float	4 bytes	-3.4e38 to 3.4e38	Float	0.0
double	8 bytes	-1.7e308 to 1.7e308	Double	0.0
boolean	NA	Not applicable(but allowed values true|false)	Boolean	false
char	2 bytes	0 to 65535	Character	0(represents blank space)

Class and Objects:

Class:

- Class is a blueprint or template from which objects are created.
- Object is an instance (i.e. example) of class.

  Why we write a class:
  - Java is an object-oriented programming language.
  - To create an object first, we need to define the properties which that object should have.
  - These properties and behavior which we want to impart to the object are declared in class. Hence class is called as blueprint of objects.

  Coding standard for class name:
  - Usually class are nouns
  - Class name starts with uppercase letter.
  - If using multiple words, then each first letter of word should start in uppercase.

    For ex. class Student, class String, class StringBuffer, class StundentInformation etc.

  Type of classes:
  1) Build in classes
  2) User defined classes

1) Build in classes:

- Java language has provided set of predefined classes within predefined package.
- This classes are required commonly in all type of projects hence they are provided by default in java.
- Majorly used building classes are:
  1. java.lang.String
  2. java.lang.Exception
  3. java.lang.Object
  4. java.lang.Class
  5. java.util.Date
  6. java.util.Scanner
  7. java.util.HashMap
  8. java.util.ArrayList ….etc…

2) **Custom defined classes:**
   - This is user defined classes which are created by user as per their project requirement.

How to declare a class:

Syntax:

<**Access Specifier**> **class** <**Class Name**>{
    //class body here
}

For ex. public class Employee{}

Components of class:

a) Fields:
   - Fields are used to define the properties of the object of class.
   - Fields are declared within class body.
   - Fields also means variables.
     For ex.

```
public class Student{
    private double percent;
    private int rollNum;
}
```

b) Methods:
   - Method is a collection of statement which defines a behavior of class object.

c) Constructor:
   - Consturctors are special type of method which are used for object creation.

d) Blocks:
   - Whenever we want to execute some code during class loading or object creation then we use blocks.
   - There are two blocks in java. Static blocks and instance blocks.

e) Nested / Inner class:
   - A class within a class in called as Inner class.

Rules for class:

i. A java class must have a class keyword.
ii. Class name must start with uppercase and if using more than one word then each starting letter of word should be upper case.
iii. There should not be any space or special character in class name. Only allowed special character are $ and _ (underscore).
iv. Java class can only have public or default access specifier.
v. A class can extend only one parent class. By default all the class in java extends java.lang.Object class directly or indirectly.
vi. Class can implement any number of interfaces separated by commas.
vii. Class containing main method is known as main class and is the entry point of any java program.

Objects:

1) Object is a real world entity which has its own property/state and behavior.
2) So object contains of the following things:
   a) State:
      This is represented by the attribute and properties of object
   b) Behaviour:
      This is defined by the methods of the object.
      For ex.
      Mobile is an object which has :
      State/property such as colour, model number, ram,etc
      Behaviour such as calling, messaging, etc.

How to create an Object:

Syntax:

<Class Name> <object name> = new <Class Name>();

For ex.

If there is a student class then to create object of student class :

Student student = new Student();

Hello World Program:
Open IDE
Select File -> New -> Project -> Under Java select Java Project -> Under Project Name ->Enter any name you wish -> Finish

Right click on project name -> New -> Class -> Under Name enter the class name you wish -> Finish

Program:

```
public class Demo {

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

}
```

Output :
Hello World

Variable in Java:

- The variable is the basic unit of storage in a Java program.
- A variable is a name given to a memory location. It is the basic unit of storage in a program.
- Variable is a memory location name of the data.
- A variable is defined by the combination of an identifier, a type, etc.
- In addition, all variables have a scope, which defines their visibility, and a lifetime.

Declaring a variable:

- In Java, all variables must be declared before they can be used.

Syntax:

<Date-type> <variableName> = value,< variableName>= value ;

Rules / Coding standard for variables/object name:

- Usually variable names are nouns.
- Should starts with lowercase alphabet symbol and if it contains multiple words every inner word should starts with upper case character.(camel case convention)

For ex. student, studentInformation, employee, employeeAddress, etc

Types of variables:

Variables in java can be classified in two ways:

Division 1 : Based on the type of value represented by a variable all variables are divided into 2 types. They are:

1. Primitive variables:

   Primitive variables can be used to represent primitive values.

   Example: `int x=10;`

2. Reference variables

   Reference variables can be used to refer objects.

   Example: `Student student=new Student();`

Division 2 : Based on the behavior and position of declaration all variables are divided into the following 3 types.

1. Instance variables
2. Local variables
3. Static variables

Instance variables:

- If the value of a variable is varied from object to object such type of variables are called instance variables.
- For every object a separate copy of instance variables will be created.
- Instance variables will be created at the time of object creation and destroyed at the time of object destruction hence the scope of instance variables is exactly same as scope of objects.
- Instance variables will be stored on the heap as the part of object.
- Instance variables should be declared with in the class directly but outside of any method or block or constructor.

- Instance variables can be accessed directly from Instance area. But cannot be accessed directly from static area. But by using object reference we can access instance variables from static area.
- Instance variables also known as object level variables or attributes.

Example:

```
public class Demo {
	int rollNum =10;
		public static void main(String[] args){
			//System.out.println(rollNum);
			//C.E:non-static variable i cannot be referenced
			from a static context(invalid)

			Test test=new Test();
			System.out.println(test. rollNum);//10(valid)
			test.methodOne();
		}
		public void methodOne(){

			System.out.println(i);//10(valid)
		}
}
```

Note:
For the instance variables it is not required to perform initialization, JVM will always provide default values.

Example:

```
class Test{
	boolean isValid;
	public static void main(String[] args){
		Test test=new Test();
		System.out.println(test.isValid);//false
	}
}
```

Local Variables:

- Some times to meet temporary requirements of the programmer we can declare variables inside a method or block or constructors such type of variables are called local variables or automatic variables or temporary variables or stack variables.
- Local variables will be stored inside stack area.
- The local variables will be created as part of the block execution in which it is declared and destroyed once that block execution completes. Hence the scope of the local variables is exactly same as scope of the block in which we declared.
  For ex:

```
public class Demo {
	int i =100;
	public static void main(String[] args) {
		int x;
		System.out.println(x); // CE : Local variable x
		may not have been initialised
	}

}
```

Methods :

- Method is a block of statements used to perform some task.
  For ex. addition method can be used to perform addition of numbers, calculatePercentage method can be used to calculate percentage scored by student, etc

- Basically methods are used to perform some operation.

Why do we write a method:

- We write a method to avoid rewriting a block of statement which will be required frequently in our program/project.
- Suppose we have a block of statement in our program which calculates the addition of numbers.
- In some another program we want to calculate the percentage. In this case we again need to calculate the addition of numbers and divide it by number count to calculate an average.Due to this we are writing the same code in our program again and again which increase the efforts and time required.
- If we define one method and write a logic for addition of marks then instead of writing a entire code for addition of numbers again in another class, we will call this method into the class and calculate **the** average.
- This is were method comes into picture.

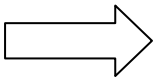
Coding Standard for Methods:

- Usually method names are either verbs or verb-noun combination.
  For ex. verb like run(), sleep(),addition(),etc
  verb-noun like getSalary(), setPercentage(),etc
- Should starts with lowercase character and if it contains multiple words every inner word should starts with upper case letter.(camel case convention)
  For ex. main(), addition(), getStudentDetails(), etc.

- Method name should be some meaningful name so that other developer can understand exactly what the purpose of the method is by reading method name itself.
- Method body is enclosed in curely braces.i.e. { }

Syntax:

```
<Access Specifier> <return type> <methodName>(arg1, arg2..)
{
        //Statement
        .
        .
        . //Statement

}
```



Space between opening and closing curly braces is method body

For ex:

1.
```
public void demo() {

        // no need to return anything

}
```

2.
```
public int test() {

        // should return integer. for ex returning 10
        return 10;

}
```

3.
```
public String demo2() {

        //should return String, for ex returning "Velocity"
        return "Velocity";

}
```

Note:

1. **Access specifier can be anything .i.e. public / private / protected / default.**
2. **Return type can be any type of datatype. i.e. primitive datatype like int, float, char, etc or non primitive data type like String, Student, etc. It can be also void.**
3. **Arguments can be of any type of data type. i.e. primitive datatype like int, float, char, etc or non primitive data type like String, Student, etc. Argument is optional hence method can be without any argument.**

Types of Methods:

- There are two types of methods:
  a) Static method
  b) Non Static method

a) Static Method

- When we declare a method with static keyword, then such method is called as static method.
- We can call / use the static method using following ways:
  - By using a class name
  - By creating a object
  - By using a method name directly into the class (within the same class only)

For ex:

```
public class Student {

        public static void test() {
                System.out.println("In test method");
        }

        public static void main(String args []) {

                //1. Calling 1st way using a class name
                Student.test();
```

```
                    //2. By 2nd way creating a Object of class
                    Student student = new Student();
                    student.test();

                    //3. By 3rd way using method name directly
                    within the class
                    test();

                }
            }
```

Output:

```
In test method
In test method
In test method
```

Description:

public : Public is a Access specifier

static : Static is a keyword which indicates that given method is static method

void : It indicates the return type of method. Void means it return nothing

test : test is the name of the method

b) Non static method:

- When we define a method without static keyword, then that method is called as non static method.
- We can call this method by using object of class.

For ex.

```
public class Employee {
    public static void main(String args[]) {
        //By creating a Object of class
        Employee employee = new Employee();
        employee.getSalary();;
    }
```

```
    public void getSalary() {
        System.out.println("Employee salary is 80000");
    }

}
```

Output:

Employee salary is 80000

Method with arguments:

- Till now in above example, we haven't passed any argument till yet.
- Following example shows how we can pass an argument in a method and how it can be used in a method.

For ex.

```
public class Employee {

    public void getEmpName(String name) {
        System.out.println("Employee name is : "+ name);
    }

    public static void main(String args[]) {

        //By creating a Object of class
        Employee employee = new Employee();
        employee.getEmpName("Prashant");
    }
}
```

Output:

Employee name is : Prashant

Operators in Java

Operators are nothing but to perform some operations on variables called as operators. There is list of types of operators in java are as follow.

- Arithmetic operators
- Logical operators
- Relational operators
- Assignment operators
- Bitwise operators
- Unary operators
- Ternary operators
- Shift operators
- instanceOf operators
- new operators
- . operators

1. Arithmetic operators-

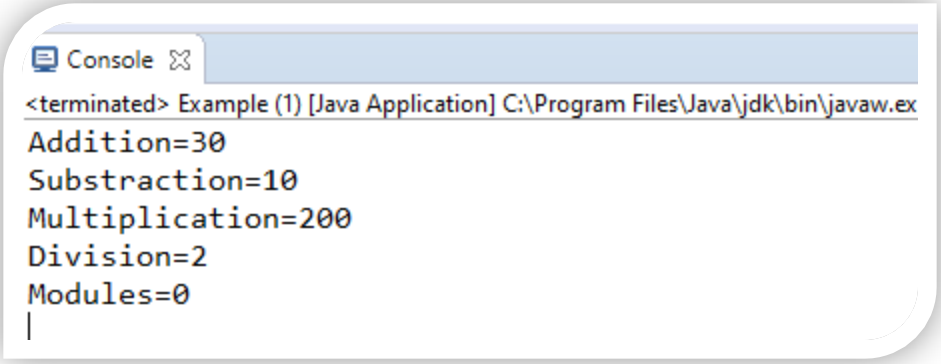
This operators are used to perform some mathematical operation such as addition (+), subtraction (-)
, Multiplication (*), Division (/) and modules (%), etc.

Example-

Example.java

```
1  package com.test;
2
3  public class Example {
4
5      public static void main(String[] args) {
6          int x = 20;
7          int y = 10;
8          System.out.println("Addition=" + (x + y));
9          System.out.println("Substraction=" + (x - y));
10         System.out.println("Multiplication=" + (x * y));
11         System.out.println("Division=" + (x / y));
12         System.out.println("Modules=" + (x % y));
13
14     }
15 }
16
```

Output-



2. Logical operators

This operators are used to perform logical AND & OR operation.

1. Logical AND (&&) operators-

Logical && operator doesn't check second condition if first condition is false. It checks second condition only if first one is true.

Expression 1	Expression 2	Results
T	T	T
T	F	F
F	T	F
F	F	F

Fig- Truth Table for Logical AND operator

Example- Scenario- 1

Example.java

```
1  package com.test;
2
3  public class Example {
4
5      public static void main(String[] args) {
6          int x = 10;
7          int y = 20;
8          int z = 30;
9          System.out.println(x < y && x < z);
10
11     }
12 }
13
```

In this example, first condition 10<20 is becomes true and second condition 10<30 is becomes true, both conditions are true, hence output is true.

Output-

Console

```
<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\javaw.exe
true
```

Example- Scenario 2

Example.java

```
1  package com.test;
2
3  public class Example {
4
5      public static void main(String[] args) {
6
7          int x=10;
8          int y=20;
9          int z=30;
10         System.out.println(x>y && x<z);
11     }
12 }
13
```

In second example, first condition 10>20 is becomes false and second condition 10<30 is becomes true, hence output is false.

Console

```
<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\javaw.
false
```

2. Logical OR (||) operators-

Logical || operator doesn't check second condition if first condition is true. It checks second condition only if first one is false.

Expression 1	Expression 2	Results
T	T	T
T	F	T
F	T	T
F	F	F

Fig- Truth table for Logical OR Operator

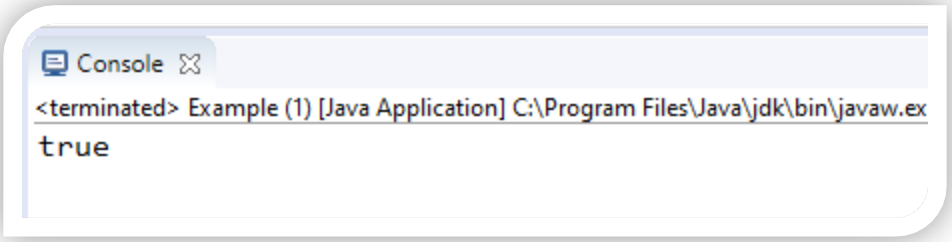
Example- Scenario-1

Example.java

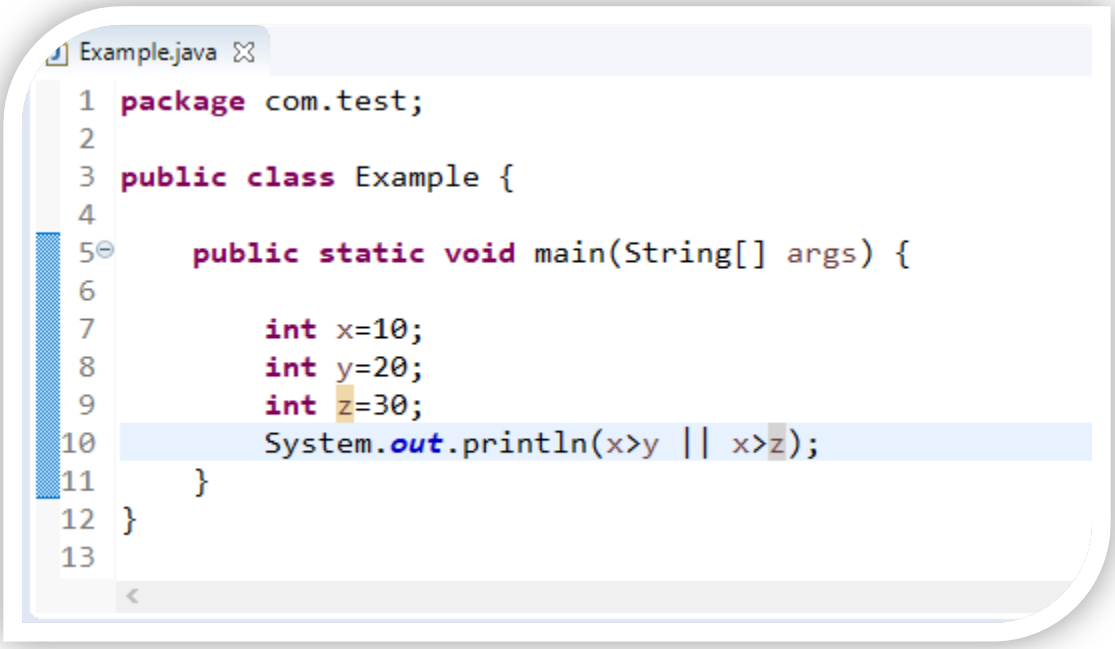
```
1  package com.test;
2
3  public class Example {
4
5      public static void main(String[] args) {
6          int x = 10;
7          int y = 20;
8          int z = 30;
9          System.out.println(x < y || x < z);
10
11     }
12 }
13
```

In this example, first condition 10<20 is becomes true and second condition 10<30 is becomes true, hence output is true.

Output-

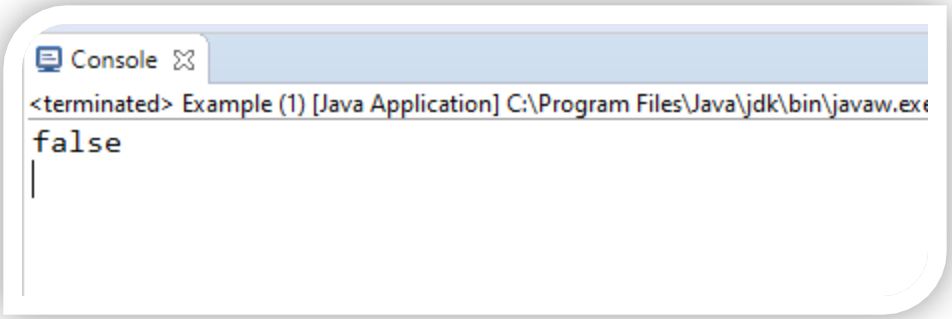


Example- Scenario-2



In this example, first condition 10>20 is becomes false and second condition 10>30 is becomes false, hence output is false.

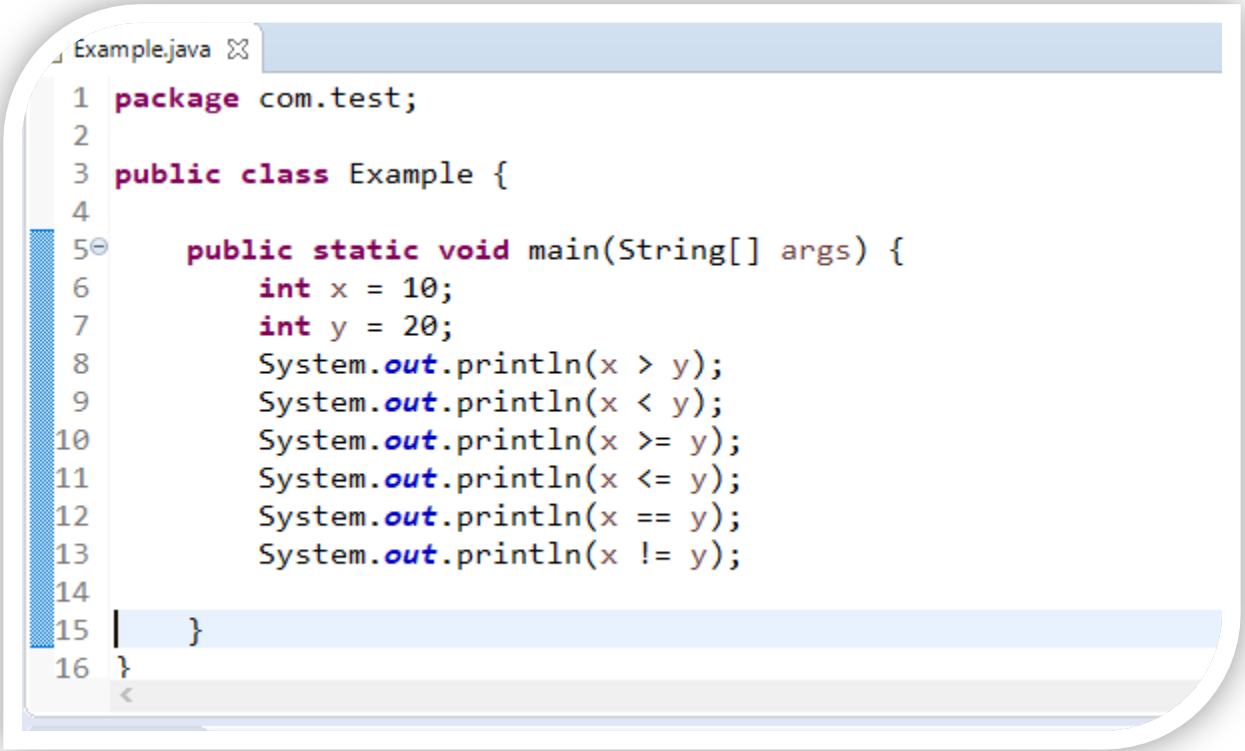
Output-



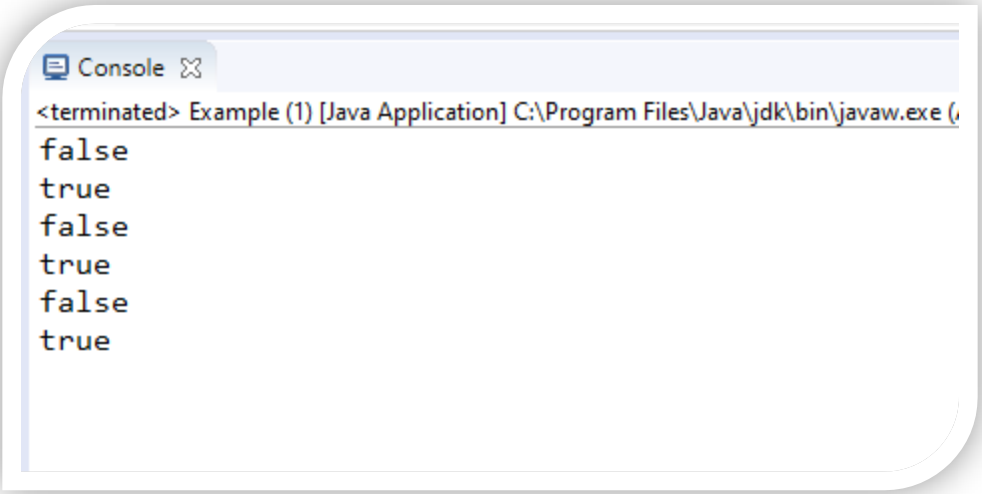
3. Relational Operators-

This operators are used to perform greater than (>), less than (<), greater than or equal to (>=), less than or equal to (<=), equal to (==), not equal to (!=), etc.

Example-



Output-

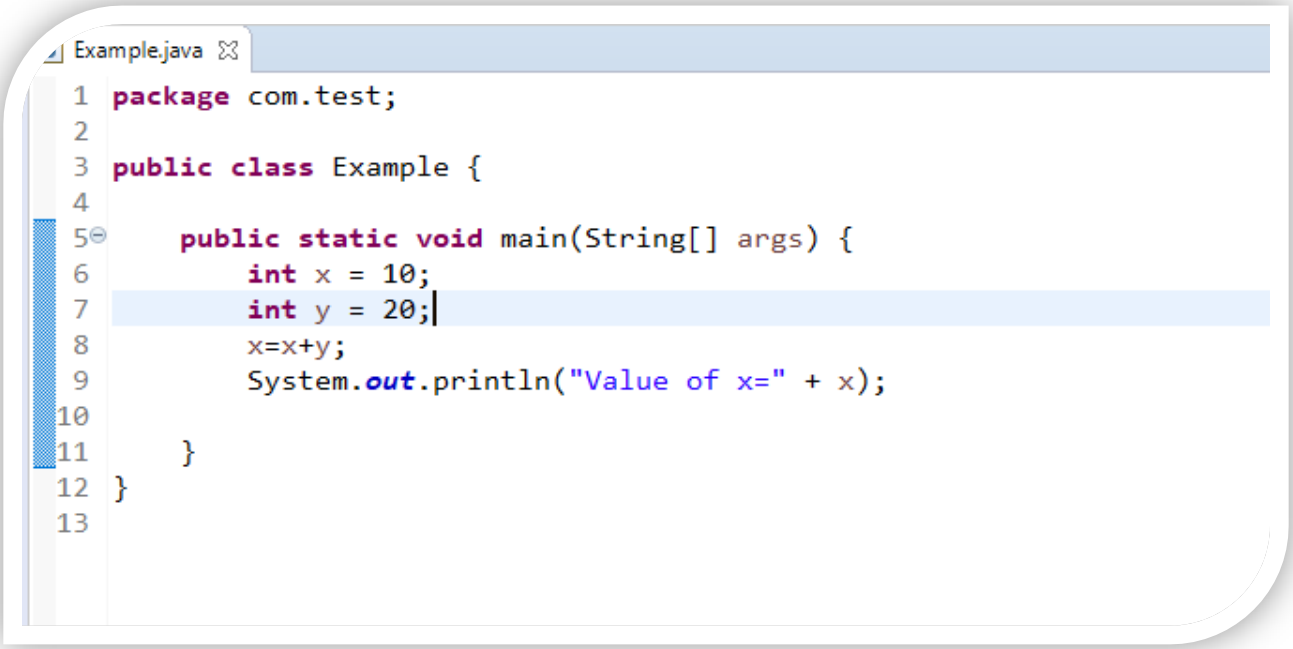

Console
<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\javaw.exe (
false
true
false
true
false
true

4. Assignment operators-

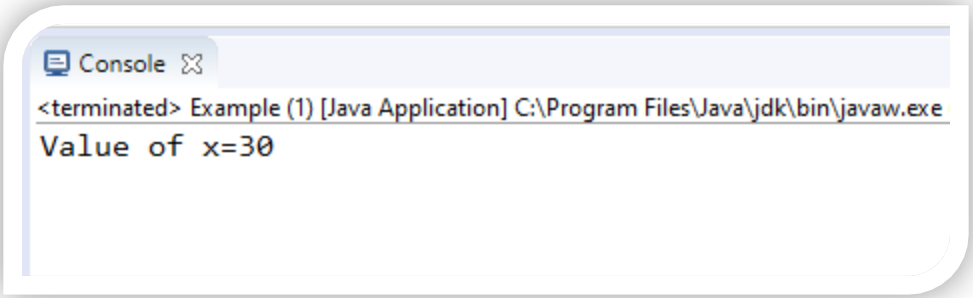
This operators is used to assign the values to variable.

Syntax- Variable =value;

Example-



Output-



5. Bitwise operators-

This operators are used to perform Bitwise AND & OR operation.

1. **Bitwise AND(&) operators-**

The bitwise & operator always checks both conditions whether first condition is true or false.

Expression 1	Expression 2	Results
T	T	T
T	F	F
F	T	F
F	F	F

Fig- Truth table for Bitwise AND operator

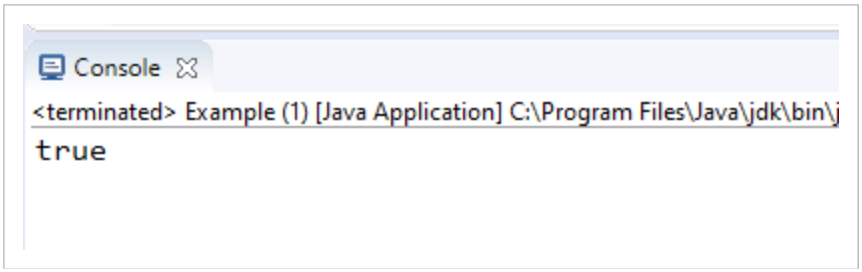
Example- Scenario-1

Example.java

```
1  package com.test;
2
3  public class Example {
4
5      public static void main(String[] args) {
6          int x = 10;
7          int y = 20;
8          int z = 30;
9          System.out.println(x < y & x < z);
10
11     }
12 }
13
```

In this example, first condition 10<20 is becomes true and second condition 10<30 is becomes true, hence output is true.

Output-



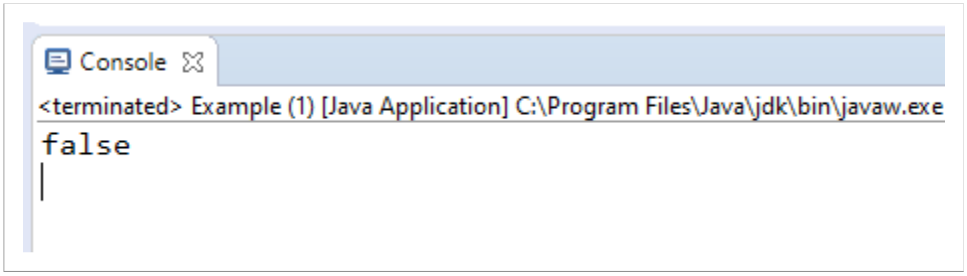
Example- Scenario- 2

Example.java

```
1  package com.test;
2
3  public class Example {
4
5      public static void main(String[] args) {
6
7          int x=10;
8          int y=20;
9          int z=30;
10         System.out.println(x>y & x<z);
11     }
12 }
13
```

In second example, first condition 10>20 is becomes false and second condition 10<30 is becomes true, hence output is false.

Output-



2. Bitwise OR(|) operators-

The bitwise (|) operator always checks both conditions whether first condition is true or false.

Expression 1	Expression 2	Results
T	T	T
T	F	T
F	T	T
F	F	F

Fig- Truth table for Bitwise OR operator

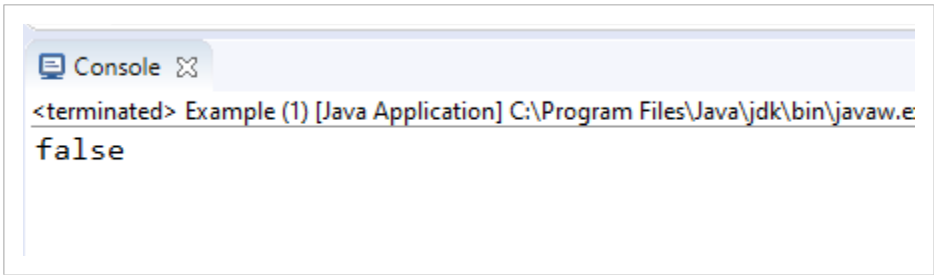
Example-Scenario-1

Example.java

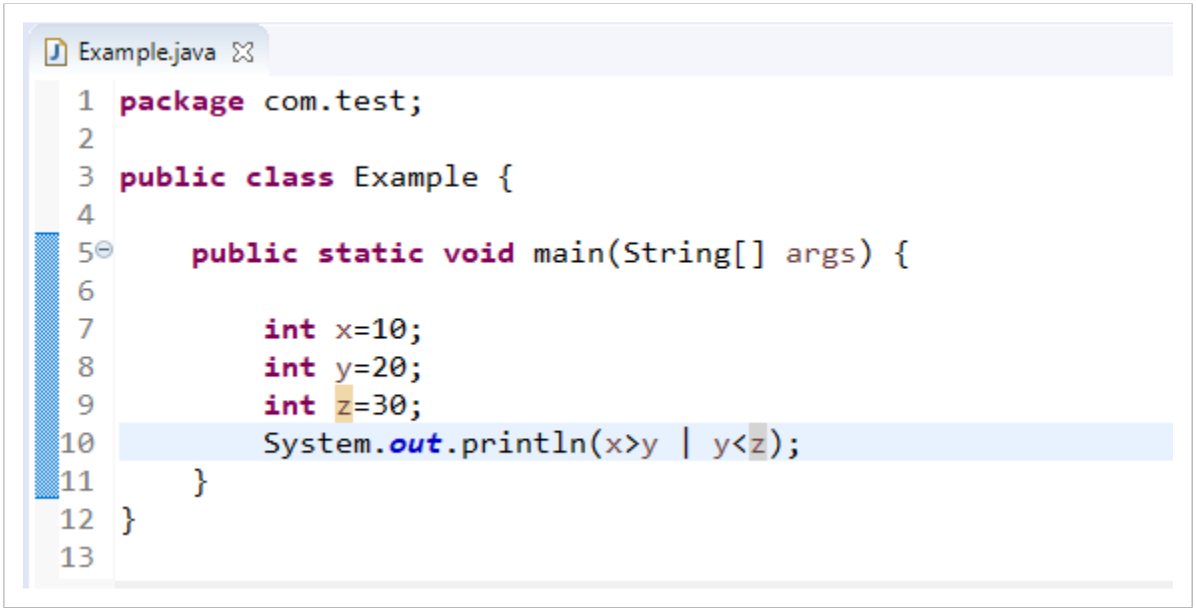
```
1  package com.test;
2
3  public class Example {
4
5      public static void main(String[] args) {
6          int x = 10;
7          int y = 20;
8          int z = 10;
9          System.out.println(x > y | x < z);
10
11     }
12 }
13
```

In this example, first condition 10>20 is becomes false and second condition 10<30 is becomes false, hence output is false.

Output-

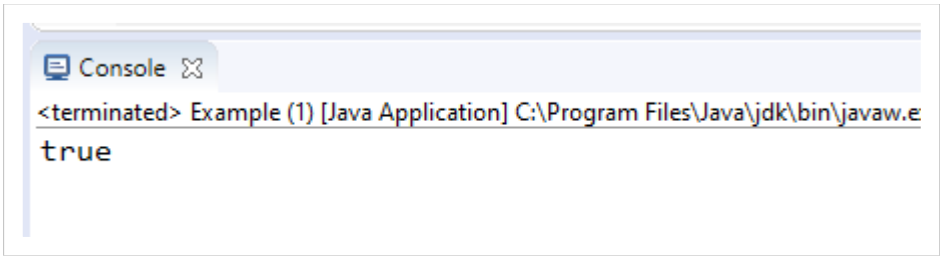


Example- Scenario- 2



In second example, first condition 10>20 is becomes false and second condition 20<30 is becomes true, hence output is true.

Output-



6. Unary operators-

This operators are used to perform an operation like increment (++) or decrement (--).

Example

Example.java

```
1  package com.test;
2
3  public class Example {
4
5      public static void main(String[] args) {
6          int x = 10;
7          System.out.println(x++);
8          System.out.println(x--);
9          System.out.println(++x);
10         System.out.println(--x);
11
12     }
13 }
14
```

Output-

Console

<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\javaw.ex

```
10
11
11
10
```

7. Ternary operators-

It includes three operands.

Why?

If else statement requires group of line code to execute the statement but by using this, we can write the code into one line only.

Example-

Example.java

```
1  package com.test;
2
3  public class Example {
4
5      public static void main(String[] args) {
6          int x = 10;
7          int y = 20;
8          int no=(x<y)?x:y;
9          System.out.println("No is="+no);
10     }
11 }
12
```

In this example, condition 10<20 becomes true, so output is 10.

Output

Console

```
<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\javaw.ex
No is=10
```

8. Shift operators (Right/Left)-

Right shift operator >> is used to move left operands value to right by the number of bits specified by the right operand.

Left shift operator << is used to shift all of the bits in a value to the left side of a specified number of times.

Example-

Example.java

```
1  package com.test;
2
3  public class Example {
4
5      public static void main(String[] args) {
6
7          int a=10;
8          System.out.println(a<<2);
9          System.out.println(a<<3);
10         System.out.println(a>>2);
11         System.out.println(a>>3);
12     }
13 }
14
```

Output-

Console

```
<terminated> Example (1) [Java Application] C:\Program Files\Java\jdk\bin\javaw.ex
40
80
2
1
```

1. On line 9, left shift operators occurs two times (<<), so we write it as 2, Right hand side we shift the position by 3 bits (i.e. numeric 3), Hence statement is 2^3.
   We will always perform the multiplication operation on left shift operators. So we are putting value of a variable is 10.

Then will calculate, 10 * 2^3 = ?
Cube of 2 is 8, so 10 *8 =80.
We will get the output as **80**

2. On line 11, right shift operators occurs two times (>>), so we write it as 2, Right hand side we shift the position by 3 bits (i.e. numeric 3), Hence statement is 2^3.
   We will always perform the division operation on right shift operators. So we are putting value of a variable is 10.
   Then will calculate, 10 / 2^3 = ?
   Cube of 2 is 8, so 10 /8 =1.25 but the rounded value is 1.
   We will get the output as **1**.
3. 10 /22 = 2
4. 10 /23 = 1

1. (.) operators

It is used to refer the member of class using class name or objects.

10. new operators

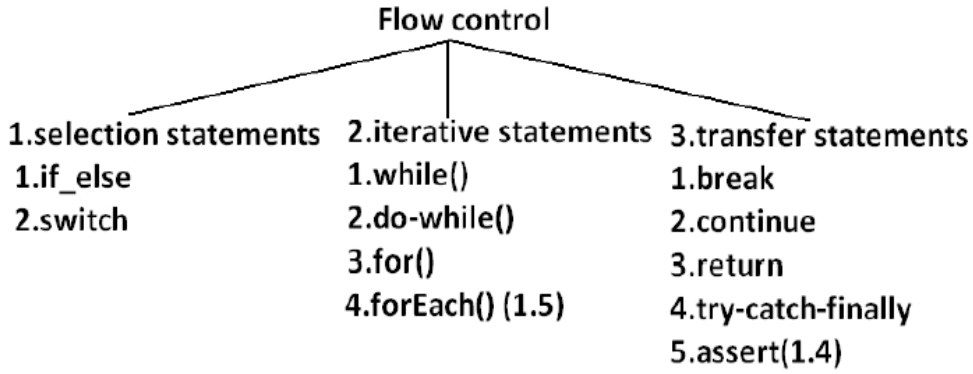
It is used to create the object of class.

Flow Control Statements :

While programming, we need to manage the flow of the program execution. To describe the order in which program statements will be executed, we have flow control statements.

Flow control statements classification:

There are various types of flow control statements as follows:



Selection Statements:

Java Selection statements are used to executive a particular statement when some condition is fulfilled or not.

Selective statement can be further classified as :

1. if – else
2. switch

1) if-else:

If-else is used to executive a statement/statements when a particular condition is true or false.

Note : Note that while providing the condition for if else, it should be Boolean type always.

It should be Boolean always, even if it's some expression still it's net result should be of Boolean type

Syntax:

```
if(condition){
//if condition is true executive if block only
}else{
//if condition is false then executive else block only
}
```

Example 1:

```
public class Demo {
	public static void main(String args[]) {
		int x = 0;
		if (x) {
			System.out.println("hello");
		} else {
			System.out.println("hi");
		}
	}
}
```

OUTPUT:

Compile time error:

Demo.java:4: incompatible types

found : int

required: Boolean if(x)

Example 2:

```
public class Demo {
	public static void main(String args[]) {
		boolean x = true;
		if (x) {
			System.out.println("hello");
		} else {
			System.out.println("hi");
		}
	}
}
```

OUTPUT:
hello

Note:

1) **Curly braces after if and else are optional. In such case, we can write only one statement after if and else , which should not be declarative statement.**

   For ex:

```
if(true)
        System.out.println("in if block");
else
        System.out.println("in else block");
```

   This is valid. Note there are no curely braces after if and else.But there must be only one statement which should not be declarative statement.

   Ex2.

```
if(x)
   int y=10;
else
   int z=100;
```

   This is invalid as the statement after if else without curly braces are declarative statements.

2) **If can exist without else. Else block is optional.**

   For ex.

```
if(2<10){
        System.out.println("in if block");
        }
```

   Output:

   in if block

3) **Nesting of if else is possible. Nesting means writing if/else blocks within other if/else block.**

For ex.

```
if(x){
        if(y){
        //execute this
        }else{
        //execute this
        }
}else{
        if(z){
        //execute this
        }
}
```

4) **If we want to check multiple conditions out of which only one conditions should be executed then we use if-else-if ladder**

For ex.

```
int x=3;
        if(x==1){
                System.out.println("Value of x is one");
        }else if(x==2){
                System.out.println("Value of x is two");
        } else if(x==3){
                System.out.println("Value of x is three");
        }
```

Output:

Value of x is three

In if-else-if ladder we can use a default condition which will execute if none of the condition is matched.

For ex.

```
int x=4;
        if(x==1){
            System.out.println("Value of x is one");
        }else if(x==2){
            System.out.println("Value of x is two");
        } else if(x==3){
            System.out.println("Value of x is three");
        }else{
            System.out.println("Value of x is not matched");
        }
```

Output:

Value of x is not matched

2) switch:

If multiple options are available, then instead of using if-else-if ladder/if-else multiple times we should go for switch statement as it improves the readability of code.

Syntax:

```
switch(x){
        case 1:
            action1
        case 2:
            action2
        ...default:
            default action
}
```

For ex.

```
int x=3;
        switch(x){
        case 1:
        System.out.println("Executing case 1");
        case 2:
        System.out.println("Executing case 2");
        case 3:
        System.out.println("Executing case 3");
        default:
        System.out.println("Executing default case ");
        }
```

Output:

```
        Executing case 3
```

Note:

1) **If no case is matched then default case will be executed.**
2) **Until 1.4 version the allow types for the switch argument are byte, short, char, int but from 1.5 version on wards the corresponding wrapper classes (Byte, Short, Character, Integer) and "enum" types also allowed.**
3) **Curly braces are mandatory.(except switch case in all remaining cases curly braces are optional)**
4) **Both case and default are optional.**
5) **Every statement inside switch must be under some case (or) default. Independent statements are not allowed.**
6) **Every case label should be within the range of switch argument type.**
7) **Duplicate case labels are not allowed.**
8) **Within the switch statement if any case is matched from that case onwards all statements will be executed until**

end of the switch (or) break. This is call "fall-through" inside the switch .

9) **Within the switch we can take the default only once**
10) **Within the switch we can take the default anywhere, but it is convention to take default as last case.**

Iteration Statements:

This type of statement is used to execute a statement or set of statements multiple times.

Iteration statement is further classified as:

1) for loop

- This is one of the most widely used loop.
- This loop is the best choice if the number of iterations is already known.

Syntax:

```
        1          2 ,5              4 ,7
for( initilization ;condition check;increment/decrement)
   body;   3 ,6
}
```

For loop have four sections:

a) Initialization Section:

- In this section, we declare the loop variable.
- This section is the starting point of for loop and is executed only once.
- We can initialize multiple variables at the same time but should be of same type.
- We can write any valid java statement here, even System.out.println()

For ex.

```
1.  for(int j=10, j=20;…;….){
    //valid
    }
2.  for(int j=10, int j=20;…;….){
    //invalid
```

```
    }
3.  for(int j=10, boolean b=false;…;….){
    //invalid
    }
4.  for(System.out.println("Hi");…;….){
    //valid
    }
```

b) Conditional Check:

- This section of for loop is executed multiple times depending upon the condition used, variable value, increment/decrement, etc.
- As we are checking a condition here so any java statement which ultimately evaluates to boolean result (true/false) can be used here.
- This section is optional and if are not using any statement here then compiler will by default keep it as true and it will be an infinite for loop.
- Till the time conditions check is true, it will execute the for loop body. Once the condition check becomes false, then loop body won't be executed and control will completely come out of for loop.

c) Increment and decrement section:

- This section is used to increase/decrease the value of loop variable.
- Although we can use any valid java statement including s.o.p also.

Note:
All the section of for loop are optional.
Curly braces are optional and without curly braces we can take only one statement which should not be declaration statement.

For ex:

1.
```
for( ; ; ){
        System.out.println("Hi");
}
```

Output: Infinite time Hi.

2.
```
int x=0;
for(System.out.println("In initialization");  x<3 ;
System.out.println("In increment/decrement")){
        System.out.println("In for loop body");
        x++;
}
```

Output:

```
In initialization
In for loop body
In increment/decrement
In for loop body
In increment/decrement
In for loop body
In increment/decrement
```

Unreachable statement in for loop:

There can be a compile time error of unreachable statement while using for loop.

It happens when for loop will be executed infinitely and the next statement after for loop won't execute ever.

For ex.

```
for(int i=0;true;i++){
        System.out.println("In for loop body");
}
System.out.println("Outside for loop");
```

Output:

Here System.out.println("Outside for loop") will give compile time error as the for loop will run indefinitely and this statement will never get a chance to execute.

1. while loop

- While loop is the best choice when the number of iterations are not known in advance.

Syntax:

```
while(condition){
    loop body;
}
```

- The Condition argument should be of Boolean type. Otherwise we will get a compile time error.
- Curly braces are optional here also and without curly braces we can write only one java statement which should not be a declaration statement.

  For ex.

  1)
```
while(rs.next()){
//body
}
```

Unreachable statement in while loop:

Like for loop, while loop can also have unreachable statement.

For ex:

```
while(true){
        System.out.println("Inside while loop");
}
System.out.println("Outside while loop");
```

Output:

Here it will be compile time error as Outside while loop statement will never get a chance to execute as while loop will execute infinitely.

do-while loop:

If we want to execute a loop body atleast once, then we use do while loop.

Syntax:

```
do{
  ------
  //loop body
  ------
}while(condition);
```
⇦ Semi colon compulsary

- *Curly braces are optional.*
- *If we use without curly braces then we should take exactly one statement which should not be declarative statement.*

For ex.

1.
```
do{
    System.out.println ("Hello");
}while(true);
```

Output:

Hello (infinite times)

2.
```
do{
    System.out.println ("Hello");
}while(false);
```

Output:

Hello

Unreachable statement in while loop:

Like for and while loop, do-while loop can also have unreachable statement.

For ex:

```
do{
        System.out.println ("Hello");
}while(true);
System.out.println("Outside do-while loop");
```

Output:

Here it will be compile time error as Outside do-while loop statement will never get a chance to execute as do-while loop will execute infinitely.

Transfer / Jump Statements:

Break statement:

Break statement is used in following cases:

1) Inside loop when you want to break the loop execution under some condition
   - If you don't want to continue the loop to execute if some particular condition is met, then we use break within the loop.

For ex.

```
for(int i=0; i<10;i++){
        if(i==4)
        break;
        System. out.println("Hello "+i);
}
        System. out.println("Outside for loop");
```

Output:
Hello 0
Hello 1
Hello 2
Hello 3
Outside for loop

2) Inside switch to stop the fall through within switch

- Within the switch statement if any case is matched from that case onwards all statements will be executed until end of the switch (or) break. This is call "fall-through" inside the switch
- Hence to avoid or stop fall-through we use break in switch.

For ex

1. Switch case without break:

```
int x=2;
switch(x){
        case 1:
            System.out.println("Case one");
        case 2:
            System.out.println("Case two");
        case 3:
            System.out.println("Case three");
        case 4:
            System.out.println("Case four");
        default:
            System.out.println("Default Case");
}
```

Output:

Case two
Case three
Case four
Default Case

2. Switch case with break:

```
int x=2;
switch(x){
    case 1:
        System.out.println("Case one");
        break;
    case 2:
        System.out.println("Case two");
        break;
    case 3:
        System.out.println("Case three");
        break;
    case 4:
        System.out.println("Case four");
        break;
    default:
        System.out.println("Default Case");
}
```

Output:

Case two

Note:

If you use break statement at any other location apart from mentioned above, then it will be a compile time error.

Continue Statement:

If you want to skip just current iteration and go to next iteration of loop instead of skipping the entire loop execution then we use continue statement.

For ex:

1) Print all even number between 1 to 10.

```
for(int i=1; i<=10;i++){
    if(i%2!=0)
    continue;
    System.out.println(i);
}
```

Output:

2
4
6
8
10

Note:
If you use continue statement at any other location apart from mentioned above, then it will be a compile time error.

Packages-

Package is nothing but collection of classes and interface that's works together called as packages.

Java.lang is default package in java. We can create our custom packages also.

Why?

Suppose imagine, if you have large number of files in your project that is deployed on server, now the code is released on production server. There are bugs in specific files then how you can reach to that file without packages is very difficult. If you have packages then it will get very easy to go specific folder and found that file. That's why packages comes into picture.

Advantages

Packages helps to resolves naming confliction

Reusability- we can placed the common code into one folder and reuse it.

Maintenance- if any new developer/tester joined your company then it will be easy to find the file which they wanted.

Syntax-

com.wipro.jpmorgan.insurance.policy.education

Here,

Package are generally starts with com folder.

wipro is your company name.

jpmorgan is your client name.

insurance is your project name.

policy is your module name.

education is your sub-module name.

Note- All alphabets are starts with small case letters only.

import-

When we use one class within another class then go for import statement.

Example- suppose we have two different classes Test & Example in different packages.

```
package com.velocity;

public class Test {

    //method or variable
    public void m1() {
        System.out.println("this is the m1 method");
    }
}
```

```
package com.wipro.jpmorgan;

public class Example {

    public static void main(String[] args) {

        Test test= new Test();
    }
}
```

In the test class, we are calling the method of test class, so we need to use the import statement here. Otherwise it will give compile time error

To resolve this issue, we need to import the highlighted line that is Import `import 'Test'(com.velocity)` by just clicking on it.

Different ways for import-

```
import com.velocity.Test; //correct
import com.velocity.*;  //correct- it will import the all the classes.

import com.velocity; //wrong
```

Scanner in java

Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings. It is the easiest way to read input in a Java program.

- To create an object of Scanner class, we usually pass the predefined object System.in.
- To read numerical values of a certain data type, the method to use is nextXYZ(). For example, to read a value of type short, we can use nextShort() and so on.
- To read strings, we use nextLine().

Program for using scanner.

Example-1

```
import java.util.Scanner;

public class Demo {

	public static void multiplication(int no) {

		for (int i = 1; i <= 10; i++) {
			int c = no * i;
			System.out.println(no + "*" + i + "=" + c);
		}
	}

	public static void main(String[] args) {

		System.out.println("Enter the number for multiplication>>");
		Scanner scanner = new Scanner(System.in);
		int x = scanner.nextInt();
```

```
            System.out.println("value>>" + x);
            multiplication(x);
      }
}
```

Example-2

```
import java.util.Scanner;

public class Test{

      public int add(int a, int b) {
            int c = a + b;
            return c;
      }

      public static void main(String[] args) {

            Scanner scanner = new Scanner(System.in);

            System.out.println("Enter the first number>>");
            int firstNumber = scanner.nextInt();
            //take the input from user use nextInt();

            System.out.println("Enter the second number>>");
            int secondNumber = scanner.nextInt();

            System.out.println("first Number>>"+firstNumber);
            System.out.println("second Number>>"+secondNumber);

            Demo demo = new Demo();
            int add=demo.add(firstNumber, secondNumber);
            System.out.println("Addition>>"+add);

      }
}
```

Access Specifiers-

Access Specifiers are the keyword which are used to control the access.

There are four types of access specifiers:

a) public b) private c) protected d) default

a) private :

1) It apply to global variables, static variables, methods, constructors and inner class.
2) Outer class cannot be private.
3) It can be accessed within class only , not outside class because scope is very limited.
4) Local variable can't be private.

b) default :

1) It apply to global variables, static variables, methods, constructors, inner class and outer class.
2) It can be accessed only within a same package.
3) Whenever access specifier is not provided it will be treated as default. It doesn't require any keyword.

c) protected :

1) It apply to global variables, static variables, methods, constructors and inner class.
2) It can't be applied to local variable.
3) It can be accessed within the same package and in different package if inheritance exists while calling.

d) public :

1) It applies to outer class, inner class, global variables, static variables, methods and constructors
2) It can be accessed from same class or different class in same package or different package.
3) Local variables can't be public because they have a limited scope, within the method/loop/block in which they are declared.

Note:

Local variable can only be final

The only modifiers to outer class are : public, default, final, abstract, strictfp. If you use any other modifier it will be an error.

The least accessible modifier is private.

The most accessible modifier is public.

Recommended modifier for variables is private and recommended modifiers for method is public.

Summary:

Visibility	**private**	**default**	**protected**	**public**
1. Within the same class	Yes	Yes	Yes	Yes
2) From child class in same package	No	Yes	Yes	Yes
3) From non child class of same package	No	Yes	Yes	Yes
4) From child class of outside package	No	No	Yes (but we should use child reference only)	Yes
5) From non child class of outside package	No	No	No	Yes

Static variables:

- If the value of a variable is not varied from object to object such type of variables is not recommended to declare as instance variables. We have to declare such type of variables at class level by using static modifier.
- In the case of instance variables for every object a separate copy will be created but in the case of static variables for entire class only one copy will be created and shared by every object of that class.
- Static variables will be created at the time of class loading and destroyed at the time of class unloading hence the scope of the static variable is exactly same as the scope of the .class file.
- Static variables will be stored in method area. Static variables should be declared with in the class directly but outside of any method or block or constructor.
- Static variables can be accessed from both instance and static areas directly.
- We can access static variables either by class name or by object reference but usage of class name is recommended.
- But within the same class it is not required to use class name we can access directly.
- Static variables also known as class level variables or fields.

Example:

```
class Test{
	int x=100;
	static int y=200;

	public static void main(String[] args){
		Test t1=new Test();
		t1.x=888;
		t1.y=999;
		Test t2=new Test();
		System.out.println(t2.x+"----"+t2.y);//100------- 999
	}
}
```

How to access static variable.

There are two ways to access the static variables.

1. By using class name
2. By using object name.

Program for static variables

```
class Statickeyword{
    public static void main(String args[]){
    Statickeyword obj= new Statickeyword();
    static int a=5;
    System.out.println(Statickeyword.a); //by using class name
    System.out.println(obj.a);  // by using object name
 }
}
```

Output-

5

5

In this example, we are calling static variable by using statickeyword class name, also calling it by using object obj as shown in above program.

Why it is called as single copy storage?

Because for multiple object only one copy of variable will be made for a given static variable.

```
package com.velocity;

public class Test10 {

	int a = 5;
	static int b = 5;

	public static void main(String args[]) {

		Test10 sd2 = new Test10();
		System.out.println("non static>>"+sd2.a++);
		System.out.println("static>>"+sd2.b++);
		Test10 sd3 = new Test10();
		System.out.println("non static>>"+sd3.a++);
		System.out.println("static>>"+sd3.b++);
		Test10 sd4 = new Test10();
		System.out.println("non static>>"+sd4.a++);
		System.out.println("static>>"+sd4.b++);
		Test10 sd5 = new Test10();
		System.out.println("non static>>"+sd5.a++);
```

```
            System.out.println("static>>"+sd5.b++);

      } }
```

Output-

```
non static>>5
static>>5
non static>>5
static>>6
non static>>5
static>>7
non static>>5
static>>8
```

Note- We cannot call non-static member from static member because static variables stored into memory before object creation and non-static variables stored into memory after object creation.

How to access static members from non-static members. Following program shows you.

```
class StaticDemo{
      void test(){
            System.out.println("This is non static method");
            StaticDemo().x1 () //calling static method from non static method.
      }

      static void x1(){
            System.out.println("This is static method");
      }

      public static void main(String args[]){
            StaticDemo s= new StaticDemo();
      s.test();
      }
```

Output-

This is non static method.

This is static method.

Static method-

If you define any method with static keyword then it is called as static method.

It belongs to class rather than object of class.

It loads into memory before object creation.

It can access only static data member only.

Note: - Main method is static method.

```
class StaticDemo{
	static void x1(){
		System.out.println("This is static method.");
	}

	public static void main(String args[]){
		StaticDemo.x1();
	}
}
```

Output-

This is static method.

Static block-

It is group of statements that are executed when class is loading into memory by Classloader.

It is widely used to create the static resource.

We cannot access non-static variable into static block.

It is always executed first.

```
package com.test;

public class Test3 {

    static {
        System.out.println("this is the 1st static
block...");
    }
```

```
    public static void main(String[] args) {

        System.out.println("this is main method..");
    }
}
```

Output-

```
this is the 1st static block...
this is main method..
```

Here, we will get first output is “This is 1st static block” because it is executed first than main method.

Note- Outer class cannot be static but inner class can be static.

Constructor cannot be static.

Local variables cannot be static

Constructors:

1. Object creation is not enough compulsory we should perform initialization then only the object is in a position to provide the response properly.
2. Whenever we are creating an object some piece of the code will be executed automatically to perform initialization of an object this piece of the code is nothing but constructor.
3. Hence the main objective of constructor is to perform initialization of an object.

Rules to write constructors:

1. Name of the constructor and name of the class must be same.
2. It is invoked by JVM automatically at time of object creation.
3. Return type concept is not applicable for constructor even void also by mistake if we are declaring the return type for the constructor we won't get any compile time error and runtime error compiler simply treats it as a method.

Example:

```
class Test
{
        void Test(){
                //it is not a constructor and it is a method
        }
}
```

3. It is legal (but should not be done) to have a method whose name is exactly same as class name.
4. The only applicable modifiers for the constructors are public, default, private, protected.
5. If we are using any other modifier we will get compile time error.

Types of Constructors:

There are three types of constructors :

1. No argument constructors
2. Parameterized constructors
3. Default constructors

1) **No argument constructors**

A constructor that doesn't accept any argument is called as no argument constructor.

```
package com.velocity.demo;

public class Test {

	int id;
	//program for no arg constructor
	Test() {
		id=10;
	}

	public static void main(String [] args) {
		Test test= new Test();
		System.out.println("Id value is "+test.id);
	}
}
```

Output-

Id value is 10

2) Default constructors :

A constructor that is automatically created by compiler is called as default constructor. Compiler will generate this constructor only if programmer doesn't write any constructor. If we write atleast one constructor then compiler will not generate default constructor.

```
package com.velocity.demo;

public class Test {

	int id;

	public static void main(String [] args) {
		Test test= new Test(); //Calling default constructor

	}
}
```

3) Parameterised Constructors:

The constructor with parameter is called as parameterized constructor.

```
package com.velocity.demo;

public class Test {

	int rollNum;
	String studentName;

	Test(int num, String name){
		rollNum=num;
		studentName=name;
	}
	public static void main(String [] args) {
		Test test= new Test(10, "Velocity"); //Calling
	parameterized constructor
	System.out.println("Student Roll Num is "+test.rollNum);
	System.out.println("Student Name is "+test.studentName);
	}
}
```

Output :

```
Student Roll Num is 10
Student Name is Velocity
```

Note:

1) Constructors doesn't have any return type
2) Constructors can be overloaded
3) For private constructors object creation is not possible
4) There are different ways to call constructors:
    a) Test test = new Test();
    b) new Test();
    c) super();
    d) this();

Encapsulation:

1. Binding of the data members in the single entity is called as encapsulation.

For ex.

class is the entity which contains data variables and method

```
class Employee{
        int salary;
        public int getSalary(){
                return 10000;
        }
}
```

2. Every data member should be declared as private and for every member we have to maintain public getter & Setter methods.

Why Encapsulation:

1. Suppose you have an employee class which contain salary as a variable.
2. If we do not follow encapsulation, we can create object of employee class and then this salary variable can to set to negative value also which is not valid in real world as salary cannot be negative.
3. So to overcome this problem and get more control and security over the data we need encapsulation

Example:

```
package com.velocity.demo;

public class Employee {

        public int salary;

}

public class MainTest{
        public static void main(String[] args) {
                Employee employee = new Employee();
                employee.salary=-12500; //salary cannot be negative
        }
}
```

To overcome this issue, we can encapsulate our class as:

```
package com.velocity.demo;

public class Employee {

	private int salary;

	public void setSalary(int sal) {
		//mechanism to check if salary is positive or negative
		if(sal>=0) {
			salary=sal;
		}
		else {
			salary=0;
			System.out.println("Salary cannot be negative");
		}
	}
}

class MainTest{
	public static void main(String[] args) {
		Employee employee = new Employee();
		employee.setSalary(-12500); //setter method won't set the
		salary as negative now
	}
}
```

Program for Encapsulation using Dynamic values.

```
package com.encapsulation;

public class Employee {

	private int employeeId;
	private String employeeName;
	private String employeeCity;


	public int getEmployeeId() {
		return employeeId;
	}

	public void setEmployeeId(int employeeId) {
		this.employeeId = employeeId;
	}

	public String getEmployeeName() {
		return employeeName;
	}

	public void setEmployeeName(String employeeName) {
		this.employeeName = employeeName;
	}

	public String getEmployeeCity() {
		return employeeCity;
	}

	public void setEmployeeCity(String employeeCity) {
		this.employeeCity = employeeCity;
	}
}
```

```
package com.encapsulation;

import java.util.Scanner;

public class TestMain {
	public static void getUserInput() {
		System.out.println("Enter the ID>>");
		Scanner scanner = new Scanner(System.in);
		int id = scanner.nextInt();
		System.out.println("Enter the Name>>");
		String name = scanner.next();
		System.out.println("Enter the City");
		String city = scanner.next();

		Employee employee = new Employee();
		employee.setEmployeeId(id);
		employee.setEmployeeName(name);
		employee.setEmployeeCity(city);

		System.out.println("Employee Id>>" +
employee.getEmployeeId());
		System.out.println("Employee Name>>" +
employee.getEmployeeName());
		System.out.println("Employee City>>" +
employee.getEmployeeCity());

	}

	public static void main(String[] args) {
		getUserInput();
	}

}
```

Output-
Enter the ID>>
10
Enter the Name>>
ram
Enter the City
pune
Employee Id>>10
Employee Name>>ram
Employee City>>pune

The main advantages of encapsulation are :

1. We can achieve security.
2. Enhancement will become very easy.
3. It improves maintainability and modularity of the application.
4. It provides flexibility to the user to use system very easily.

The main disadvantage of encapsulation is it increases length of the code and slows down execution.

Inheritance-

The process of creating the new class by using the existing class functionality called as Inheritance.

Inheritance means simply reusability.

It is called as - (IS Relationship)

Example- IS Relationship

```
Class Policy {


}
Class TermPolicy extends Policy {



}
```

In this example, Policy is super class and TermPolicy is sub class

Where TermPolicy **IS A** Policy.

Note-

All the parent members are derived into child class but they are depends upon the below

-To check the access specifiers

-Members does not exist into sub class.

Note-

1. Inherit the classes by using extends keywords.
2. Whenever we create the object of subclass then all the member will get called super class as well as sub class.
3. Why we use inheritance that is for code reusability, reusability means we can reuse existing class features such as variables and method, etc.

UML Diagram-

Parent -P

Child – C

Where p is parent class and c is the child class.

Super Class->Parent Class->Base Class-> Old Class

Sub Class-> Child Class-> Derived Class -> New Class

When to use?

If we want to extends or increase of features of class then go for inheritance.

Business requirement-

Why inheritance?

Suppose we have one class which contain the fields like, firstname, lastname, address, city, mobile number and

In future we got the requirement to add the PAN number then what option we have below-

1. Modify the attributes/fields in existing class but this is not good option it will increase the testing for that class.
2. Add the attributes in the new class, in this the good option we can also reduce the testing efforts for this.

How the class will looks like

```
Class Parent {

String firstname;
String lastname;
String address;
String city;
String mobilenumber;
}
Class Child extends Parent {

String pancard;

}
```

Rules-

We cannot assign parent class reference to child class-

We cannot extend the final class.

All the members of super class will be directly inherited into sub class and they are eligible and depends on access specifiers only.

Dynamic dispatch-

The process of assigning the child class reference to parent class called as "Dynamic dispatch."

Example-

```
Class X {

}
Class Y extends X {

}
```

Class Test {

Public static void main(string args[]){

X x= new Y(); // Here we are assigning the child reference new Y() to parent class .

}

Inheritance Example-

Scenario 1

```
package com.inheritance;

class X {

	int a = 10;
	int b = 20;

	void m1() {
		System.out.println("Class X- m1() method");
	}

	void m2() {
		System.out.println("Class X- m2() method");
	}
}
package com.inheritance;

class Y extends X {

	int b = 30;
	int c = 40;

	void m2() {
		System.out.println("Class Y- m2() method");
	}

	void m3() {
		System.out.println("Class Y- m3() method");
	}
}
```

```java
package com.inheritance;

public class TestMain {

	public static void main(String[] args) {

		//Scenario- 1
		X x=new X();
		System.out.println(x.a);
		System.out.println(x.b);
		System.out.println(x.c);
		x.m1();
		x.m2();
		x.m3();

		//Scenario-2
		Y y = new Y();
		System.out.println(y.a);
		System.out.println(y.b);
		System.out.println(y.c);
		y.m1();
		y.m2();
		y.m3();

	   //Scenario-3
		X x = new Y();
		System.out.println(x.a);
		System.out.println(x.b);
		//System.out.println(x.c);
		x.m1();
		x.m2();
		//x.m3();
```

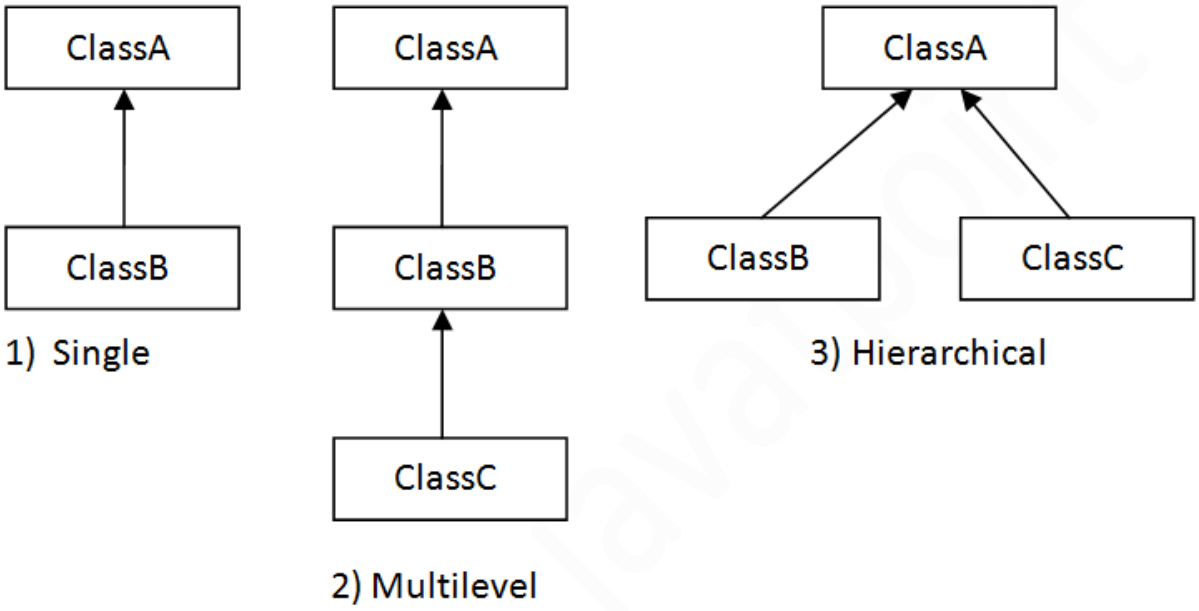
```
        //Scenario-4 (Note 3rd and 4th scenario are same)
        X x = new X();
        Y y = new Y();
        x = y;
        System.out.println(x.a);
        System.out.println(x.b);
        System.out.println(x.c);
        x.m1();
        x.m2();
        x.m3();

       //Scenario-5- Note- this is equivalent to 2nd
scenario
         X x = new Y();
        Y y = new Y();
        y = (Y) x;
        System.out.println(y.a);
        System.out.println(y.b);
        System.out.println(y.c);
        y.m1();
        y.m2();
        y.m3();

 //Scenario-6
        Y y= new X(); // Not Valid
    }
}
```

Note:

1) Whatever the parent has by default available to the child but whatever the child has by default not available to the parent. Hence on the child reference we can call both parent and child class methods. But on the parent reference we can call only methods available in the parent class and we can't call child specific methods.
2) Parent class reference can be used to hold child class object but by using that reference we can call only methods available in parent class and child specific methods we can't call.
3) Child class reference cannot be used to hold parent class object
4) For all java classes the most commonly required functionality is define inside object class hence object class acts as a root for all java classes.

There the five types of inheritance as below

1. Single inheritance
2. Multiple inheritance
3. Multilevel inheritance
4. Hybrid inheritance
5. Hierarchical inheritance



1. Simple or Single inheritance
   In this only one super class and only one sub class called as single.

```
package com.single.inheritance;

public class A {

     void m1() {
          System.out.println("super class- m1 () method");
     }
}

package com.single.inheritance;
```

```
public class B extends A {

	void m2() {
		System.out.println("sub class - m2() method");
	}

}
```

```
package com.single.inheritance;

public class TestMain {

	public static void main(String[] args) {

		B b = new B();
		b.m1();
		b.m2();
	}
}
```

2. Multilevel inheritance

   It has only one base class and multiple derived class called as multilevel. Or it refers to the concept of one class extending (Or inherits) more than one base class.

```
package com.multilevel.inheritance;

public class A {

	void m1() {
		System.out.println("Class A- m1 () method");
	}
}
```

```
package com.multilevel.inheritance;

public class B extends A{
```

```
    void m2() {
        System.out.println("Class B- m2 method");
    }
}
```

```
package com.multilevel.inheritance;

public class C extends B {

    void m3() {
        System.out.println("Class c- m3 () method");
    }

    public static void main(String[] args) {

        C c= new C();
        c.m1();
        c.m2();
        c.m3();
    }
}
```

3. Multiple inheritance
   One class has many super classes called as multiple inheritance.

   Why multiple inheritance not supported in java in case of classes?

   Class base has test () method and class derived has also test () method. Class test extends Base, Derived, which test method It will called, so it create the ambiguity so that's why multiple inheritance does not supports in java.

   Draw diagram here.

```
package com.multiple.inheritance;

public class A {

    void m1() {
```

```
    }
}
```

```
package com.multiple.inheritance;

public class B {

    void m1() {

    }
}
```

```
package com.multiple.inheritance;

 class C extends A,B  {

    public static void main(String[] args) {

        C c= new C();
        c.m1();
    }
}
```

Note- it will get the compile time error.

4. Hierarchical inheritance
   One class is inherited by many sub classes called as.

```
package com.hierachical.inheritance;

public class A {

    void m1() {
        System.out.println("Class A- m1 () method");
    }
}
```

```java
package com.hierachical.inheritance;

public class B extends A {

	void m2() {
		System.out.println("Class B- m2() method");
	}
}
```

```java
package com.hierachical.inheritance;

public class C extends A{

	void m3() {
		System.out.println("Class c m3 method");
	}
}
```

```java
package com.hierachical.inheritance;

public class D {

	public static void main(String[] args) {

		B b = new B();
		C c = new C();

		b.m1();
		b.m2();
		c.m3();
	}
}
```

5. Hybrid inheritance

   It is the combination of single and multiple inheritance. So it is not allowed in java.

Polymorphism-

One entity that behaves differently in different cases called as polymorphism. Or Same name with different forms is the concept of polymorphism

Example- Light button, we are using that button to on or off the lights. Calculation of Income Tax as per income, etc

We can use same abs() method to calculate the absolute value for int type, long type, float type etc.

Example:

1. abs(int)

2. abs(long)

3. abs(float)

How to achieve polymorphism in java?

We can achieve polymorphism by using two ways.

1. Method overloading-
2. Method overriding-

1. Method overloading-

   It is the same method name with different argument called as Method overloading. There is no need of super and sub class relationship. It is called as early binding, compile time polymorphism or static binding.

   Having overloading concept in java reduces complexity of the programming.

   Rules-

   1) Method name must be same.
   2) Parameter or argument must be different.(sequence of argument, number of argument or datatype should be different)
   3) Return type can be anything
   4) Access specifier can be anything
   5) Exception thrown can be anything

Example-1

```
package com.tests;

public class TestMain {

	void add(int a, int b) {
		System.out.println(a + b);

	}
```

```
        void add(double a, double b) {
                System.out.println(a + b);
        }

        void add(double a) {
                System.out.println(a);
        }

        void add(int a, int b, int c) {
                System.out.println(a + b + c);
        }
}
```

```
package com.tests;

public class ExampleMain {

        public static void main(String[] args) {

                TestMain testmain = new TestMain();
                testmain.add(10.5);
                testmain.add(10.5, 11.5);
                testmain.add(2, 4);
                testmain.add(5, 10, 15);
        }
}
```

Output is

```
10.5
22.0
6
30
```

Why?

Suppose we got the business requirement from the client in last year

```
Class Employee {

Void addStudent (String firstname, string lastname, string city) { }

End user is calling the class as below

//End User 1

addStudent (“ram”,”pawar”,”Pune”);

//End User 2

addStudent (“ram”,”deshmukh”,”Mumbai”);  }
```

After that I got the new requirement from the client in current year, to update the pan card details.

What options we have?

1. Modified into the existing method.

2. Create the new method with new parameter.

First way modifying into existing method is not good approach, it will increase the unit testing of it. If we are make the changes into existing method, then how existing user will calls the method I mean they need to add one more extra attribute, in future again, you got requirement to add one more attributes so every time user need to change at their side, this is not the good thing.

Second way, create the same method in that class and add the new attribute into it. If client second want pan card details so he can call that method otherwise calls the first method if pan card is not required.

Example -2

```
package com.poly;

class X { }

class Y extends X { }

class Z extends Y { }

public class Overloading {

	void test1(X x) {
		System.out.println("test1- X"); }

	void test1(Y y) {
		System.out.println("test1- Y"); }

	void test1(Z z) {
		System.out.println("test1- Z"); }
} //package name remove
public class OverloadTestMain {

	public static void main(String[] args) {

		Overloading overloading= new Overloading();
		X x= new X();
		overloading.test1(x);
		Y y= new Y();
		overloading.test1(y);
		Z z= new Z();
		overloading.test1(z);

		Y y1= new Z();
		overloading.test1(y1);
```

```
        } }
```

```
test1- X
test1- Y
test1- Z
test1- Y
```

Example- 3

```
package com.poly;

public class A{

        void test(Object object) {
                System.out.println("test- Object");
        }

        void test(String string) {
                System.out.println("test- String");
        }

        public static void main(String[] args) {
                A a = new A();
                a.test(new Object());
                a.test("Jeevan");
                a.test(new X());
                a.test(new String());
        }
}
```

Output :

```
test- Object
test- String
test- Object
test- String
```

Why it is called as compile time polymorphism?

Because it is decided at compile time which one method should get called that's why it is called as compile time polymorphism.

In overloading compiler is responsible to perform method resolution (decision) based on the reference type (but not based on run time object). Hence overloading is also considered as compile time polymorphism (or) static polymorphism (or) early biding.

In overloading method resolution is always based on reference type and runtime object won't play any role in overloading.

2. **Method overriding-**

It is the same method name with same argument called as method overriding.

There is need of super and sub relationship. It is called as late binding, run time polymorphism or dynamic binding.etc.

Whatever the Parent has by default available to the Child through inheritance, if the Child is not satisfied with Parent class method implementation then Child is allow to redefine that Parent class method in Child class in its own way this process is called overriding.

In overriding method resolution is always takes care by JVM based on runtime object hence overriding is also considered as runtime polymorphism or dynamic polymorphism or late binding.

Rules-

1) Method name must be same. That is method signature must be same. Method signature means method name with argument (return type and access specifier are not part of method signature).
2) Until 1.4 version the return types must be same but from 1.5 version onwards covariant return types are allowed.
3) Method parameters must be same.

Note-

1) We can extend the method scope in overriding but not reduce the visibility of it.
2) While overriding if the child class method throws any checked exception compulsory the parent class method should throw the same checked exception or its parent otherwise we will get compile time error.
   But there are no restrictions for un-checked exceptions.

Why?
Flexibility
Maintainability
Readability of code.

Example-

```
package com.override.demo;

public class A {

    void m1() {
        System.out.println("class - A- m1 () method");
    }
```

```
}
```

```
package com.override.demo;

public class B extends A {

	@Override
	void m1() {
		System.out.println("class - B- m1 () method");
	}

	void m7() {
		System.out.println("class- B- m7() method");
	}
}
```

```
package com.override.demo;

public class TestMain {

	public static void main(String[] args) {

		B b= new B();
		b.m1();
		b.m7();

	}
}
```

Output-

```
class - B- m1 () method
class- B- m7() method
```

Program Explaination-

- In the above program, B is implementing the method m1 () with the same signature as super class A i.e m1 () of class B is overriding m1() of class A.
- If you want to add new features to existing class, then you should not disturb the existing class. You should always write the subclass of that class that is the best practice.

Why we write the sub class

1. To add the new features
2. To inherit the existing functionality.

Subclass method's access modifier must be the same or higher than the superclass method access modifier

superclass	In subclass, we can have access specifier
public	public
protected	protected, public
default	default, protected, public
private	We cannot override the private

Note:

1) Until 1.4 version the return types must be same but from 1.5 version onwards covariant return types are allowed.
   According to this Child class method return type need not be same as Parent class method return type its Child types also allowed.

```
class Parent {
        public Object methodOne() {
                return null;
        }
}

class Child extends Parent {
        public String methodOne() {
                return null;
        }
}
```

2) Parent class final methods we can't override in the Child class.
3) Parent class non final methods we can override as final in child class. We can override native methods in the child classes.
4) Private methods are not visible in the Child classes hence overriding concept is not applicable for private methods.
5) We should override Parent class abstract methods in Child classes to provide implementation.
6) While overriding we can't reduce the scope of access modifier.
   Example:

```
class Parent {
        public void methodOne() { }
 }
```

```
class Child extends Parent {
        protected void methodOne( ) { }
}
```

Output:
Compile time error

Method Overloading- Live Example-1

```
class MobilePattern {
        void getMobilePattern(Thumb thumb) {
                // logic here
        }

        void getMobilePattern(int number) {
                //logic here
        }

        void getMobilePattern(int x1, int y1, int x2, int y2) {
                //logic here
        }
}
```

Live Example-2

```
class Account{

        void getBanking(CreditCard creditCard) {

                //logic here

        }

        void getBanking(Netbanking netBanking) {

                //logic here

        }

        void getBanking(DebitCard debitCard) {

                //logic here

        }
```

```
	void getBanking(UPI upi) {

		//logic here

	}
}
```

Method Overriding- Live Example-1

```
class RBI {
	void getSimpleIntereset(float simpleRate) {

//logic here

	}
}

class Axis extends RBI {

	void getSimpleIntereset(float simpleRate) {

//logic here

	}
}

class HDFC extends RBI {

	void getSimpleIntereset(float simpleRate) {

//logic here

	}
}
```

Live Example-2

```
class Policy {

	void getPremium(Customer customer) {
		//logic here
	}
}
```

```
class TermPolicy extends Policy {

	void getPremium(Customer customer) {
		//logic here

	}
}

class RiderProtection extends TermPolicy {

	void getPremium(Customer customer) {
		//logic here

	}
}
```

Abstraction-

It is the process of hiding the certain details and showing the important information to the end user called as "Abstraction". Or Hide internal implementation and just highlight the set of services, is called abstraction.
Example- By using ATM GUI screen bank people are highlighting the set of services what they are offering without highlighting internal implementation.

How to achieve the Abstraction in java?

There are two ways to achieve the abstraction in java.

1. Abstract class
2. Interface

Abstract class

- For any java class if we are not allow to create an object such type of class we have to declare with abstract modifier that is for abstract class instantiation is not possible.
- Abstract class have constructor
- It contains abstract methods or concrete methods or empty class or combination of both methods.
- To use abstract method of class, we should extends the abstract class and use that methods.
- If we don't want to implement or override that method, make those methods as abstract.
- If any method is abstract in a class then that class must be declared as abstract.
- We cannot create the object of abstract class.
- Even though class doesn't contain any abstract methods still we can declare the class as abstract that is an abstract class can contain zero no. of abstract methods also.

Note- Multiple inheritances are not allowed in abstract class but allowed in interfaces

Example-

```
public class Test {
    abstract void demo ();
}
```

Here, method is the abstract then class should be abstract only.

```
public abstract class Test {
    abstract void test();
    void demo(){
     //concrete methods here
     }
```

```
package com.abstraction;

public abstract class Test {

    abstract void example(); // abstract method

    abstract void demo();
}
```

How to implement that methods?

We need to create the class which extends from abstract class as shown in below.

```
package com.abstraction;

public class C extends Test {

    @Override
    void example() {
```

```
        System.out.println("this is the example method");

    }

    @Override
    void demo() {

        System.out.println("this is the demo method");

    }

}

package com.abstraction;

public class TestMain {

    public static void main(String[] args) {

        C c= new C();
        c.demo();
        c.example();

    }
}
```

Note- If in subclass I don't want to override the parent class abstract methods then make that subclass also as abstract.

Interface-

1. Any service requirement specification (SRS) or any contract between client and service provider or 100% pure abstract classes is considered as an interface.
2. Every method in interface is public abstract methods and public static final variables by default.
3. We must follow I to C design principle in java. It means every class must implement some interfaces.
4. In company, Team Lead or Manager level people can design the interface then give it to developer for implementing it.

5. Before 1.7, interface does not have any method body.

6. From 1.8 we can declare the default & static method with body in interface.

7. 1.9 we can define the private methods in interface also.

8. We cannot create the object of interface.

9. In interface, we can just define the method only but implemented that methods into implemented class.

10. Java supports multiple inheritance in the terms of interfaces but not classes.

11. Interface does not have constructor.

Syntax

interface interface_name {

}

Example-

```
package com.abstra.interf;

 interface A {

     public abstract void demo();

     public abstract void example();

}
```

```
interface A{
   public abstract void demo ();  //allowed
   public void demo (); //allowed
   void demo (); //allowed
   abstract void demo (); //allowed
}
```

Note- if we don’t write public or abstract in interface then JVM will insert it automatically.

}

```
package com.abstra.interf;

 interface A {

     public abstract void demo();

     public abstract void example();

}
```

```
package com.abstra.interf;

public class Z implements A {

     @Override
     public void demo() {

          System.out.println("this is demo method");

     }

     @Override
     public void example() {

          System.out.println("this is example method");

     }

}
```

```
package com.abstra.interf;

public class TestMain {

     public static void main(String[] args) {
```

```
            Z z = new Z();
            z.demo();
            z.example();
            //A a= new A();
        }
}
```

Example-2

interface A {

}

interface B {

}

Interface C extends A, B {

}

This is allowed in java.

Below are the list of possible scenario regarding the interface and

Note- Try this from your end on laptop or desktop.

interface can extend interface1 and interface2

Interface can extends interface

Interface can extends the multiple interface

class extends class implements interface

class implements interface

class extends class implements interface1 and interface2

Why interface?

Suppose there is a requirement for Amazon to integrate SBI bank code into their shopping cart. Their customers want to make payment for products they purchased.

Let's say SBI develops code like below:

```
class Transaction {
   void withdrawAmt(int amtToWithdraw) {
         //logic of withdraw
         // SBI DB connection and updating in their DB
   }
}
```

Amazon needs this class so they request SBI bank for the same. The problem with SBI is that if they give this complete code to amazon they risk exposing everything of their own database to them as well as their logic, which cause a security violation.

Now the solution is for SBI to develop an Interface of Transaction class as shown below:

```
interface Transactioni {
    void withdrawAmt(int amtToWithdraw) ;
}
class TransactionImpl implements Transactioni {
   void withdrawAmt(int amtToWithdraw) {
      //logic of withdraw
      //SBI DB connection and updating in their DB
   }
}
```

Now how amazon will do this as below as-

```
Transactioni ti = new TransactionImpl();
ti.withdrawAmt(500);
```

In this case, both application can achieve their aims.

Note:

We can’t create an object from abstract class and interface still abstract class has a constructor and interface doesn’t. The reason for this is in abstract class variables can be non final and hence to provide the initialization of this global variables abstract class will need constructors while creating object of child class.
But in interface, global variables are by default final and when we write a variable as final then you have to initialize that variable at the same time when we declare it so interface doesn’t require constructors for initialization of their global variables and hence interface doesn’t have constructor.

Difference between Interface and Abstract class:

Interface	Abstract class
If we don't know anything about implementation just we have requirement specification then we should go for interface.	If we are know about implementation but not completely (partial implementation) then we should go for abstract class.
Every method present inside interface is always public and abstract whether we are declaring or not.(java 1.8 and 1.9 onwards some exception in this, already mentioned above)	Every method present inside abstract class need not be public and abstract.
We can't declare interface methods with the modifiers private, protected, final, static, synchronized, native, strictfp.	There are no restrictions on abstract class method modifiers.
Every interface variable is always public static final whether we are declaring or not.	Every abstract class variable need not be public static final.
Every interface variable is always public static final we can't declare with the following modifiers. Private, protected, transient, volatile.	There are no restrictions on abstract class variable modifiers.
For the interface variables compulsory we should perform initialization at the time of declaration otherwise we will get compile time error because they are final.	It is not require to perform initialization for abstract class variables at the time of declaration.
Inside interface we can't have static and instance blocks.	Inside abstract class we can have both static and instance blocks.
Inside interface we don’t have constructor.	Inside abstract class we can have constructor.

Object class-

It is the parent class of all the classes in java. It is called as topmost class of java which is present in java.lang package.

If our class doesn't extends any other class then it is the direct child class of object. If our class extends any other class then it is the indirect child class of Object.

There are different methods of object class are as follows.

1. **Public final Class getClass()-**
   This class is used to get the metadata of class.i.e. returns runtime class definition of an object

```
package com.wipro.jpmorgan;

public class Example {

	public static void main(String[] args) {

		Example example = new Example();
		System.out.println(example.getClass().getName());
		System.out.println(example.getClass().getSimpleName());

	}
}
```

2. **Public int hashCode()-**

   For every object unique number is generated by JVM called as hashcode. It is based on address of the object but it doesn't mean hashCode represents address of the object. Jvm will be using hashCode while saving objects into hashing related data structures like HashSet, HashMap, and Hashtable etc.

   The most of java native method are written in c or c++ that why not possible to show the body.

Note-

1) If two objects are equal, their hashcode will be same.

2) If two object hashcode are same, you cannot guaranty that objects are equal.

3) Overriding hashCode() method is said to be proper if and only if for every object we have to generate a unique number as hashcode for every object

Example

```
package com.wipro.jpmorgan;

public class Example {

	public static void main(String[] args) {

		Example example1 = new Example();
		Example example2 = new Example();

		System.out.println(example1.hashCode());
		System.out.println(example2.hashCode());
	}
}
```

3) Public Boolean equals (Object obj)-

It compare the given object to this object.

If our class doesn't contain .equals() method then object class .equals() method will be executed which is always meant for reference comparison[address comparison]. i.e., if two references pointing to the same object then only .equals() method returns true .

Example-

```
package com.velocity;

public class Employee {

	int empId;
	String empName;
```

```
	public static void main(String[] args) {
		Employee emp1 = new Employee();
		emp1.empId = 1;
		emp1.empName = "Ashok";

		Employee emp2 = new Employee();
		emp2.empId = 2;
		emp2.empName = "Sachin";

		System.out.println(emp1.equals(emp2));
	}
}
```

Output- false

4) protected Object clone() throws CloneNotSupportedException-

It creates and returns the exact copy (clone) of this object.

The main objective of cloning is to maintain backup purposes.(i.e., if something goes wrong we can recover the situation by using backup copy.)

Example-

```
package com.wipro.jpmorgan;

public class Example implements Cloneable {

	int x;

	public static void main(String[] args) throws
CloneNotSupportedException {

		Example example1 = new Example();
		example1.x = 50;

		System.out.println("First Object data is>>" + example1.x);

		Object example2 = example1.clone();

		System.out.println("Second Object data is>>" + example2);
	}
}
```

5) **public String toString() –**

It returns the string representation of this object.

Whenever we are try to print any object reference internally toString() method will be executed.

```
package com.wipro.jpmorgan;

public class Example {

	int x;

	@Override
	public String toString() {
		return "Example [x=" + x + "]";
	}

	public static void main(String[] args) throws
CloneNotSupportedException {

		Example example1 = new Example();
		example1.x = 50;

		System.out.println("First Object data is>>" + example1);

	}
}
```

6) public final void notify()-

It wakes up single thread, waiting on this object's monitor.

7) **public final void notifyAll()-**

It wakes up all the threads, waiting on this object's monitor.

8) public final void wait(long timeout)throws InterruptedException()-

It causes the current thread to wait for the specified milliseconds, until another thread notifies (invokes notify() or notifyAll() method).

9) public final void wait(long timeout,int nanos)throws InterruptedException-

It causes the current thread to wait for the specified milliseconds and nanoseconds, until another thread notifies (invokes notify() or notifyAll() method).

10) public final void wait()throws InterruptedException

It causes the current thread to wait, until another thread notifies (invokes notify() or notifyAll() method).

11) protected void finalize()throws Throwable

It is invoked by the garbage collector before object will be destroyed to perform clean up activity.

Int ques : Contract between equals and hashcode method

1) If 2 objects are equal by .equals() method compulsory their hashcodes must be equal (or) same. That is If r1.equals(r2) is true then r1.hascode()==r2.hashcode( ) must be true.
2) If 2 objects are not equal by .equals() method then there are no restrictions on hashCode() methods. They may be same (or) may be different. That is If r1.equals(r2) is false then r1.hashCode()==r2.hashCode() may be same (or) may be different.
3) If hashcodes of 2 objects are equal we can't conclude anything about .equals() method it may returns true (or) false. That is If r1.hashCode()==r2.hashCode() is true then r1.equals(r2) method may returns true (or) false.
4) If hashcodes of 2 objects are not equal then these objects are always not equal by .equals() method also. That is If r1.hashCode()==r2.hashCode() is false then r1.equals(r2) is always false.

To maintain the above contract between .equals() and hashCode() methods whenever we are overriding .equals() method compulsory we should override hashCode() method.

Final keyword-

We can apply final to variables, method and class.

Final variable-

A variable which is declared with final keyword called as final variables.

Once you assigned any value to that variables then it won't be changed. It works like constants in java.

How to declare the final variables-

final int a=5;

Example-1

```
class FinalDemo {
	public static void main(String args[]) {
		final int a = 5;
		System.out.println(a);
	}
}
```

Output-

5

Example-2

```
class FinalDemo {
	public static void main(String args[]) {
		final int a = 5;
		for (int a = 5; a <= 10; a++) {
			System.out.println(a);
		}
	}
}
```

In this example, we will get compile time error, final variable values does not changed.

Final method-

Method which is defined with final keyword called as final method.

How to declare the final method-

```
public final void test(){
        //business logic here.
}
```

Note- Final method cannot be overridden.

Example-3

```
class X {
final void test(){
System.out.println("This is x class-test method");
}

        class Y extends X {
final void test(){
System.out.println("This is y class-test method");
}

                public static void main(String args[]) {
                        X x = new Y();
                        x.test();
                }
}
```

Output-

Nothing

In this example, we will get compile time error final method cannot be override final method from X

Final class-

The class which is defined with final keyword called as final class.

How to declare the final class

```
final class Test {
        // business logic
}
```

How you stop others from inheriting your class-

By making class as final.

```
final class X {
	public final void test() {
		System.out.println("x class-test method");
	}
}

class Y extends X {

}

public class FinalKeyword {
	public static void main(String[] args) {
		Y y = new Y();
		y.test();
	}
}
```

Wrapper class-

It provides the mechanism to convert primitive's data type into object and object into primitives data type called as wrapper class.

Process of converting primitive's data type into object called as "Autoboxing." And process of converting object into primitive's data type called as "Unboxing."

There are 8 classes of java.lang.package are known as wrapper classes in java.

Integer

Short

Byte

Long

Double

Character

Boolean

Float

Example-

```
package com.object;

public class WrapperDemo {

    public static void main(String[] args) {

        int a = 20; // primitive data type

        Integer i = new Integer(a); // autoboxing
        System.out.println("i>>" + i);

        int b = i.intValue();   //unboxing
        System.out.println("b>>"+b);
    }
}
```