Java Introduction

Java is one of the world's most important and widely used computer languages, and it has held this distinction for many years. Unlike some other computer languages whose influence has weared with passage of time, while Java's has grown.

Java is a high level, robust, object-oriented and a secure and stable programming language but it is not a pure object-oriented language because it supports primitive data types like int, char etc.

Java is a platform-independent language because it has runtime environment i.e JRE and API. Here platform means a hardware or software environment in which anapplication runs.

Java codes are compiled into byte code or machine-independent code. This byte code is run on JVM (Java Virtual Machine).

The syntax is Java is almost the same as C/C++. But java does not support low-level programming functions like pointers. The codes in Java is always written in the form of Classes and objects.

As of 2020, Java is one of the most popular programming languages in use, especially for client-server web applications.Its has been estimated that there are around nine million Java developers inside the world.

Creation of Java

Java was developed by James Ghosling, Patrick Naughton, Mike Sheridan at Sun Microsystems Inc. in 1991. It took 18 months to develop the first working version.

The initial name was **Oak** but it was renamed to **Java** in 1995 as OAK was a registered trademark of another Tech company.

History of Java

Originally Java was designed for Interactive television, but this technology was very much advanced for the industry of digital cable television at that time. Java history was started with the **Green Team**. The Green Team started a project to develop a language for digital devices such as television. But it works best for internet programming. After some time Java technology was joined by Netscape.

The objective to create Java Programming Language was it should be "Simple, Robust, Portable, Platform-independent, Secured, High Performance, Multithreaded, Object-Oriented, Interpreted, and Dynamic".

Java was developed in Sun Microsystem by James Ghosling, Patrick Naughton, Mike Sheridan in 1991. It took 18 months to develop the first working version. James Ghosling is also known as the Father of Java.

Initially, Java was called **"Greentalk"** by James Gosling and at that time the file extension was **.gt**.

Later on Oak was developed as a part of the Green Team project. Oak is a symbol for strength and Oak is also a national tree in many countries like the USA, Romania etc.

Oak was renamed as Java in 1995 because Oak was already a trademark by Oak Technologies. Before selecting the Java word the team suggested many names like **dynamic**, **revolutionary**, **Silk**, **jolt**, **DNA**, etc.

Java is an island in Indonesia, here the first coffee was produced or we call Java coffee. Java coffee is a type of espresso bean. James gosling chose this name while having coffee near his office.

The word JAVA does not have an acronym. It is just a name.

In 1995 Java was one of the best product by the Time magazine.

Java Version History

Here we have listed down all the versions of Java along with the main features introduced in those versions.

Version Name	**Coad Name**	**Release Date**	**Description**
Java Alpha and Beta		1995	• It was the 1st version but was having unstable APIs and ABIs. • It was the 1st version but was having unstable APIs and ABIs.
JDK 1.0	Oak	January 1996	• 1st stable version
JDK 1.1		February 1997	• AWT Event modelling retooling. • Added Inner class, Java Beans, JDBC, RMI, Reflection, JIT • Added Inner class, Java Beans, JDBC, RMI, Reflection, JIT
J2SE 1.2	Playground	December 1998	• JDK replaced by J2SE. • Support strictfp keyword. • Swing API integrated with core classes. • Collection framework.
J2SE 1.3	Kestrel	May 2000	• HotSPot JVM included • RMI Modified. • JNDI(Java Naming and Directory Interface) Supported • JPDA(Java Platform Debugger Architecture).

			• Included Proxy Classes.
J2SE 1.4	Merin	February 2002	• Support assert Keyword. • Improvement in libraries. • Support Regular expression. • Support Exception Chaining. • Support Exception Chaining. • Included Java Web Start. • Support API Preferences (java.util.prefs).
J2SE 5.0	Tiger	September 2004	• Included Generics, Metadata, Autoboxing/Unboxing, Enumerations, Varargs. • Enhanced for each loop. • Support static imports.
Java SE 6	Mustang	December 2006	• Support Win9x version. • Support Scripting languages. • Improved Swing performance. • Support JDBC 4.0 • Upgrade of JAXB to 2.0. • Improvement in GUI and JVM.
Java SE 7	Dolphine	July 2011	• Support of dynamic language in JVM. • Included 64-bit pointers. • Support string in the switch. • Support resource management in the try block. • Support binary integer literals. • Support underscore in numeric literals. • Support multiple exceptions.

			• Included I/O library.
Java SE 8(LTS)		March 2014	• Support of JSR 335 and JEP 126. • Support unsigned integer. • Support Date and time API. • Included JavaFX. • Support Windows XP.
Java SE 9		September 2017	• Support multiple gigabyte heaps. • Included garbage collector.
Java SE 10		March 2018	• Support local variables type inference. • Support local variables type inference. • Included Application class.
Java SE 11(LTS)		September 2018	• Support bug fixes. • Include long term support(LTS). • Support transport layer security.
Java SE 12		March 2019	• Support JVM Constant API. • Include CDS Archives.
Java SE 13		September 2019	• Updated Switch Expressions. • Include Text Blocks. • Support Legacy socket API.
Java SE 14		March 2020	• Support Event Streaming. • Improved NullPointerException. • Removal of the Concurrent Mark Sweep (CMS) in the garbage collector.
Java SE 15		September 2020	
Java SE 16		March 2021	
Java SE 17(LTS)		September 2021	

Evolution of Java

Java was initially launched as Java 1.0 but soon after its initial release, Java 1.1 was launched. Java 1.1 redefined event handling, new library elements were added.

In **Java 1.2** Swing and Collection framework was added and `suspend()`, `resume()` and `stop()` methods were deprecated from **Thread** class.

No major changes were made into **Java 1.3** but the next release that was **Java 1.4** contained several important changes. Keyword `assert`, chained exceptions and channel based I/O System was introduced.

Java 1.5 was called **J2SE 5**, it added following major new features :

- Generics
- Annotations
- Autoboxing and autounboxing
- Enumerations
- For-each Loop
- Varargs
- Static Import
- Formatted I/O
- Concurrency utilities

Next major release was **Java SE 7** which included many new changes, like :

- Now **String** can be used to control Switch statement.
- Multi Catch Exception
- *try-with-resource* statement
- Binary Integer Literals
- *Underscore* in numeric literals, etc.

Java SE 8 was released on March 18, 2014. Some of the major new features introduced in JAVA 8 are,

- Lambda Expressions
- New Collection Package `java.util.stream` to provide Stream API.
- Enhanced Security
- Nashorn Javascript Engine included
- Parallel Array Sorting
- The JDBC-ODBC Bridge has been removed etc.

Java SE 9 was released on September 2017. Some of the major new features introduced in JAVA 9 are,

- Platform Module System (Project Jigsaw)
- Interface Private Methods

- Try-With Resources
- Anonymous Classes
- @SafeVarargs Annotation
- Collection Factory Methods
- Process API Improvement

Java SE 10 was released on March 2018. Some of the major new features introduced in JAVA 10 are,

- Support local variables type inference.
- Support local variables type inference.
- Included Application class.

Java SE 11 was released on September 2018. Some of the major new features introduced in JAVA 11 are,

- Support bug fixes.
- Include long term support(LTS).
- Support transport layer security.

Java SE 12 was released on March 2019. Some of the major new features introduced in JAVA 12 are,

- Support JVM Constant API.
- Include CDS Archives.

Java SE 13 was released on September 2019. Some of the major new features introduced in JAVA 13 are,

- Updated Switch Expressions.
- Include Text Blocks.
- Support Legacy socket API.

Java SE 14 was released on March 2020. Some of the major new features introduced in JAVA 14 are,

- Support Event Streaming.
- Improved NullPointerException.
- Removal of the Concurrent Mark Sweep (CMS) in the garbage collector.

Application of Java

Java is widely used in every corner of world and of human life. Java is not only used in softwares but is also widely used in designing hardware controlling software components. There are more than 930 million JRE downloads each year and 3 billion mobile phones run java.

Following are some other usage of Java :

1. Developing Desktop Applications
2. Web Applications like Linkedin.com, Snapdeal.com etc
3. Mobile Operating System like Android
4. Embedded Systems
5. Robotics and games etc.

Types of Java Application

Following are different types of applications that we can develop using Java:

1. Standalone Applications

Standalone applications are the application which runs on separate computer process without adding any file processes. The standalone application is also known as Java GUI Applications or **Desktop Applications** which uses some standard GUI components such as AWT(Abstract Windowing Toolkit), swing and JavaFX and this component are deployed to the desktop. These components have buttons, menu, tables, GUI widget toolkit, 3D graphics etc. using this component a traditional software is developed which can be installed in every machine.

Example: Media player, antivirus, Paint, POS Billing software, etc.

2. Web Applications

Web Applications are the client-server software application which is run by the client. Servlets, struts, JSP, Spring, hibernate etc. are used for the development of a client-server application. eCommerce application is also developed in java using eCommerce platform i.e Broadleaf.

Example: mail, e-commerce website, bank website etc.

3. Enterprise Application

Enterprise application is middleware applications. To use software and hardware systems technologies and services across the enterprises. It is designed for the corporate area such as banking business systems.

Example: e-commerce, accounting, banking information systems etc.

4. Mobile Application

For mobile applications, Java uses ME or J2ME framework. This framework are the cross platform that runs applications across phones and smartphones. Java provides a platform for **application development in Android** too.

Example: WhatsApp, Xender etc.

Download JDK

For running Java programs in your system you will have to download and install **JDK kit from here** (recommended Java Version is Java 11 but you can download Java 13 or Java 14).

Features of Java

The prime reason behind creation of Java was to bring portability and security feature into a computer language. Beside these two major features, there were many other features that played an important role in moulding out the final form of this outstanding language. Those features are :

1) Simple

Java is easy to learn and its syntax is quite simple, clean and easy to understand.The confusing and ambiguous concepts of C++ are either left out in Java or they have been re-implemented in a cleaner way.

Eg : Pointers and Operator Overloading are not there in java but were an important part of C++.

2) Object Oriented

In java, everything is an object which has some data and behaviour. Java can be easily extended as it is based on Object Model. Following are some basic concept of OOP's.

i. Object
ii. Class
iii. Inheritance
iv. Polymorphism
v. Abstraction
vi. Encapsulation

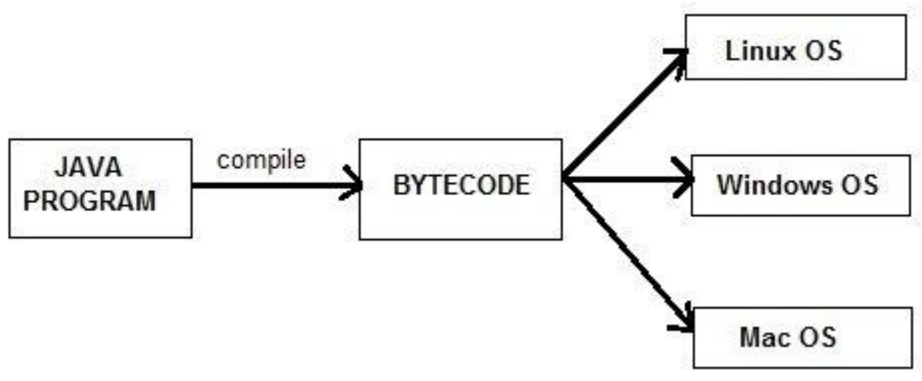
3) Robust

Java makes an effort to eliminate error prone codes by emphasizing mainly on compile time error checking and runtime checking. But the main areas which Java improved were Memory Management and mishandled Exceptions by introducing automatic **Garbage Collector** and **Exception Handling**.

4) Platform Independent

Unlike other programming languages such as C, C++ etc which are compiled into platform specific machines. Java is guaranteed to be write-once, run-anywhere language.

On compilation Java program is compiled into bytecode. This bytecode is platform independent and can be run on any machine, plus this bytecode format also provide security. Any machine with Java Runtime Environment can run Java Programs.



5) Secure

When it comes to security, Java is always the first choice. With java secure features it enable us to develop virus free, temper free system. Java program always runs in Java runtime environment with almost null interaction with system OS, hence it is more secure.

6) Multi Threading

Java multithreading feature makes it possible to write program that can do many tasks simultaneously. Benefit of multithreading is that it utilizes same memory and other resources to execute multiple threads at the same time, like While typing, grammatical errors are checked along.

7) Architectural Neutral

Compiler generates bytecodes, which have nothing to do with a particular computer architecture, hence a Java program is easy to intrepret on any machine.

8) Portable

Java Byte code can be carried to any platform. No implementation dependent features. Everything related to storage is predefined, example: size of primitive data types

9) High Performance

Java is an interpreted language, so it will never be as fast as a compiled language like C or C++. But, Java enables high performance with the use of just-in-time compiler.

10) Distributed

Java is also a distributed language. Programs can be designed to run on computer networks. Java has a special class library for communicating using TCP/IP protocols. Creating network connections is very much easy in Java as compared to C/C++.

New Features of JAVA 8

Below mentioned are some of the core upgrades done as a part of Java 8 release. Just go through them quickly, we will explore them in details later.

- Enhanced Productivity by providing Optional Classes feature, Lamda Expressions, Streams etc.
- Ease of Use
- Improved Polyglot programming. A **Polyglot** is a program or script, written in a form which is valid in multiple programming languages and it performs the same operations in multiple programming languages. So Java now supports such type of programming technique.
- Improved Security and performance.

New Features of JAVA 11

Java 11 is a recommended LTS version of Java that includes various important features. These features includes new and upgrades in existing topic. Just go through them quickly, we will explore them in details later.

- includes support for Unicode 10.0.0
- The HTTP Client has been standarized
- Lazy Allocation of Compiler Threads
- Updated Locale Data to Unicode CLDR v33
- JEP 331 Low-Overhead Heap Profiling
- JEP 181 Nest-Based Access Control

- Added Brainpool EC Support (RFC 5639)
- Enhanced KeyStore Mechanisms
- JEP 332 Transport Layer Security (TLS) 1.3
- JEP 330 Launch Single-File Source-Code Programs

Java Editions

Java Editions or we can say the platform is a collection of programs which helps to develop and run the programs that are written in Java Programming language. Java Editions includes execution engine, compiler and set of libraries. As Java is Platform independent language so it is not specific to any processor or operating system.

1. Java Standard Edition

Java Standard edition is a computing platform which is used for development and deployment of portable code that is used in desktop and server environments. Java Standard Edition is also known as Java 2 Platform, Standard Edition (J2SE).

Java Standard Edition has a wide range of APIs such as Java Class Library etc. the best implementation of Java SE is Oracle Corporation's Java Development Kit (JDK).

2. Java Micro Edition

Java Micro Edition is a computing platform which is used for the development and deployment of portable codes for the embedded and mobile devices. Java Micro Edition is also known as Java 2 Platform Micro Edition (J2ME). The Java Micro Edition was designed by Sun Microsystems and then later on Oracle corporation acquired it in 2010.

Example: micro-controllers, sensors, gateways, mobile phones, printers etc.

3. Java Enterprise Edition

Java Enterprise Edition is a set of specifications and extending Java SE 8 with features such as distributed computing and web services. The applications of Java Enterprise Edition run on reference runtimes. This reference runtime handle transactions, security, scalability, concurrency and the management of components to be deployed. Java Enterprise Edition is also known as Java 2 Platform Enterprise Edition (J2EE), and currently, it has been rebranded as Jakarta EE.

Example: e-commerce, accounting, banking information systems.

4. JavaFX

JavaFX is used for creating desktop applications and also rich internet applications(RIAs) which can be run on a wide variety of devices. JavaFX has almost replaced Swing as the standard GUI library for Java Standard Edition. JavaFX support for desktop computers and web browsers.

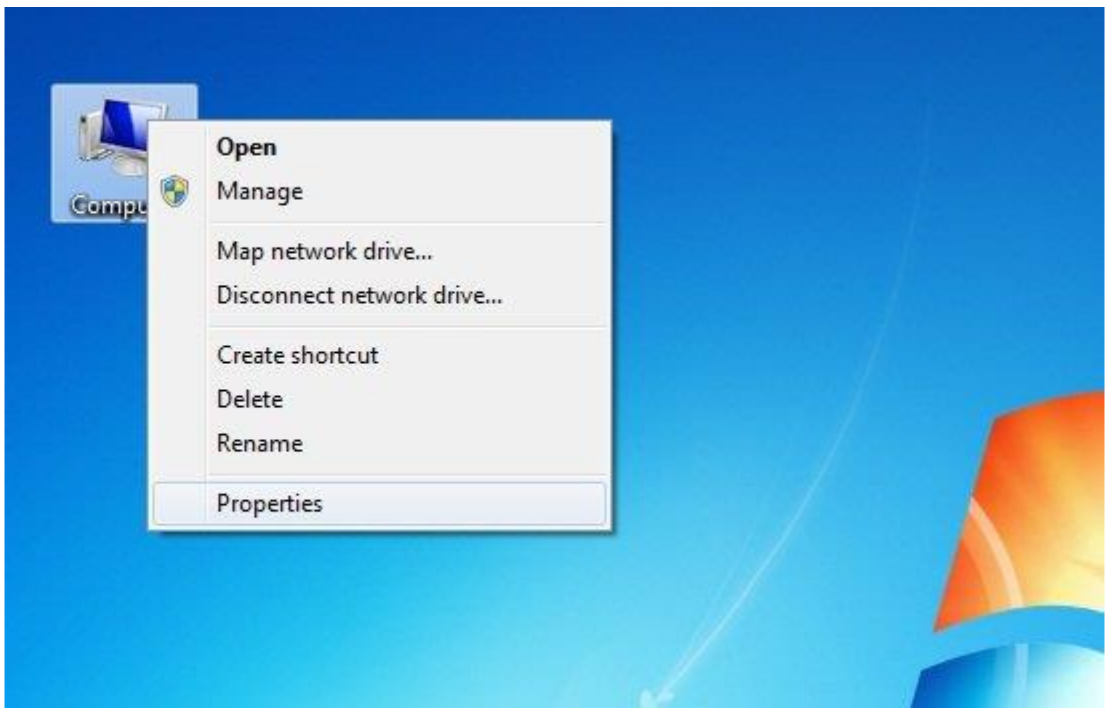
Setting Java Environment and Classpath

An **Environment variable** is a dynamic "object" on a computer that stores a value(like a key-value pair), which can be referenced by one or more software programs in Windows. Like for Java, we will set an environment variable with name "**java**" and its value will be the path of the **/bin** directory present in Java directory. So whenever a program will require Java environment, it will look for the **java** environment variable which will give it the path to the execution directory.

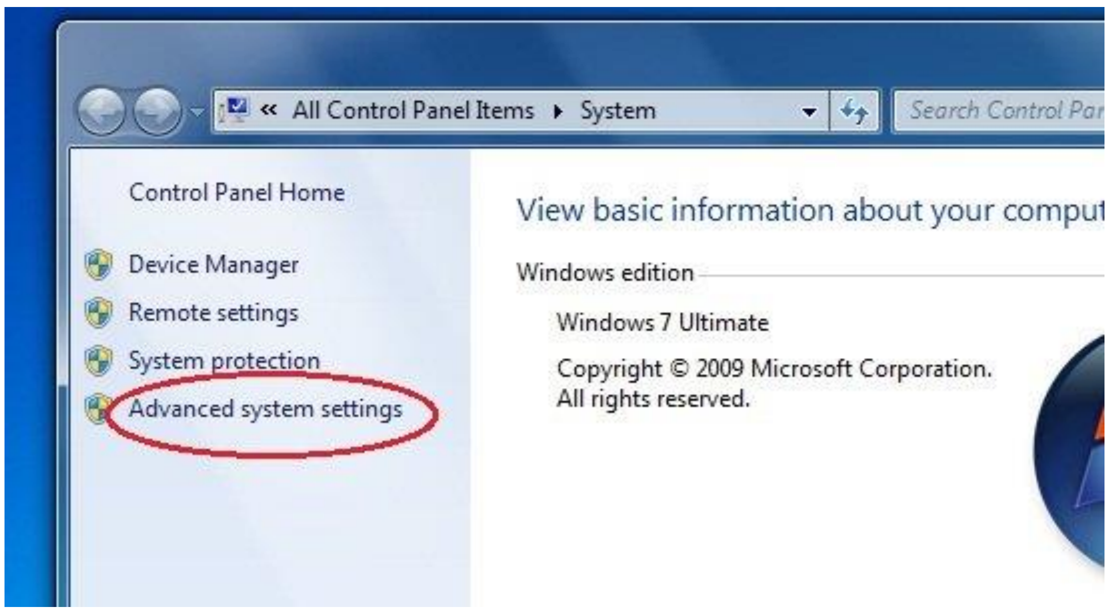
Setting up path for windows (2000/XP/vista/Window 7,8)

Assuming that you have installed Java in **C:\ Program files/ Java / JDK directory**

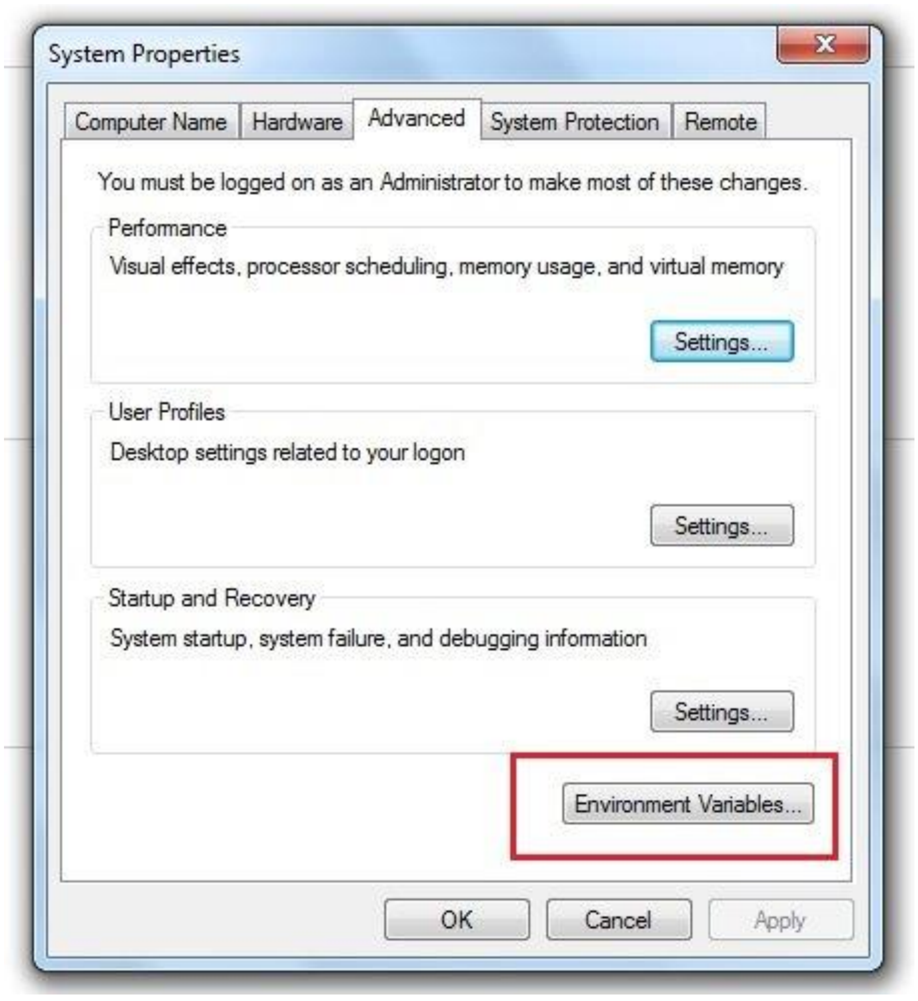
Step 1: Right click on my computer and select properties.



Step 2: Go to the Advance System Settings tab.



Step 3: Click on Environment Variables button.


System Properties

Computer Name | Hardware | Advanced | System Protection | Remote

You must be logged on as an Administrator to make most of these changes.

Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...

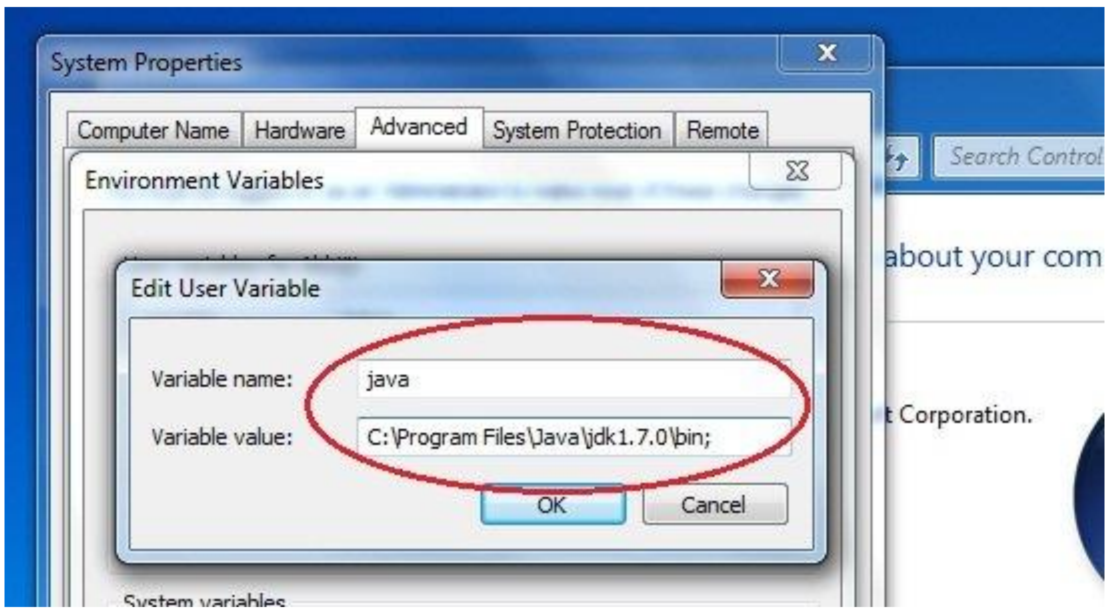
User Profiles
Desktop settings related to your logon
Settings...

Startup and Recovery
System startup, system failure, and debugging information
Settings...

Environment Variables...

OK Cancel Apply

Step 4: Now alter the path variable so that it also contains the path to JDK installed directory.


Edit User Variable

Variable name: java

Variable value: C:\Program Files\Java\jdk1.7.0\bin;

OK Cancel

For e.g:- Change **C:\windows/ system 32.** to **C:\windows/system 32; C:\program files / Java/ JDK**.

Setting up path for window 95/98/ME

Assuming that you have installed Java in C:\program files\ java\ JDK directory, do the following:

Step 1: Edit the **C:\autoexec.bat** file and add the following line at the end.

```
SET PATH =% PATH% C:\ PROGRAM FILE/JAVA/JDK/bin
```

Setting up path for Linux , Unix , Solaris, free BSD

Step 1: Environment variable path should be set to point where java binaries have been installed. Refer to your shell if you have trouble doing this.

For Example: If you use bash as your shell, then you would add following line to the end

```
bash mc: export PATH=/ Path/to/java
```

We recommend you to download latest Java 11 version. Java 11 is a LTS(Long Term Support) version and currently widely in used. You can refer our step by step installation guide to install the Java 11.

Recommended Java Version: Download the latest version of JDK

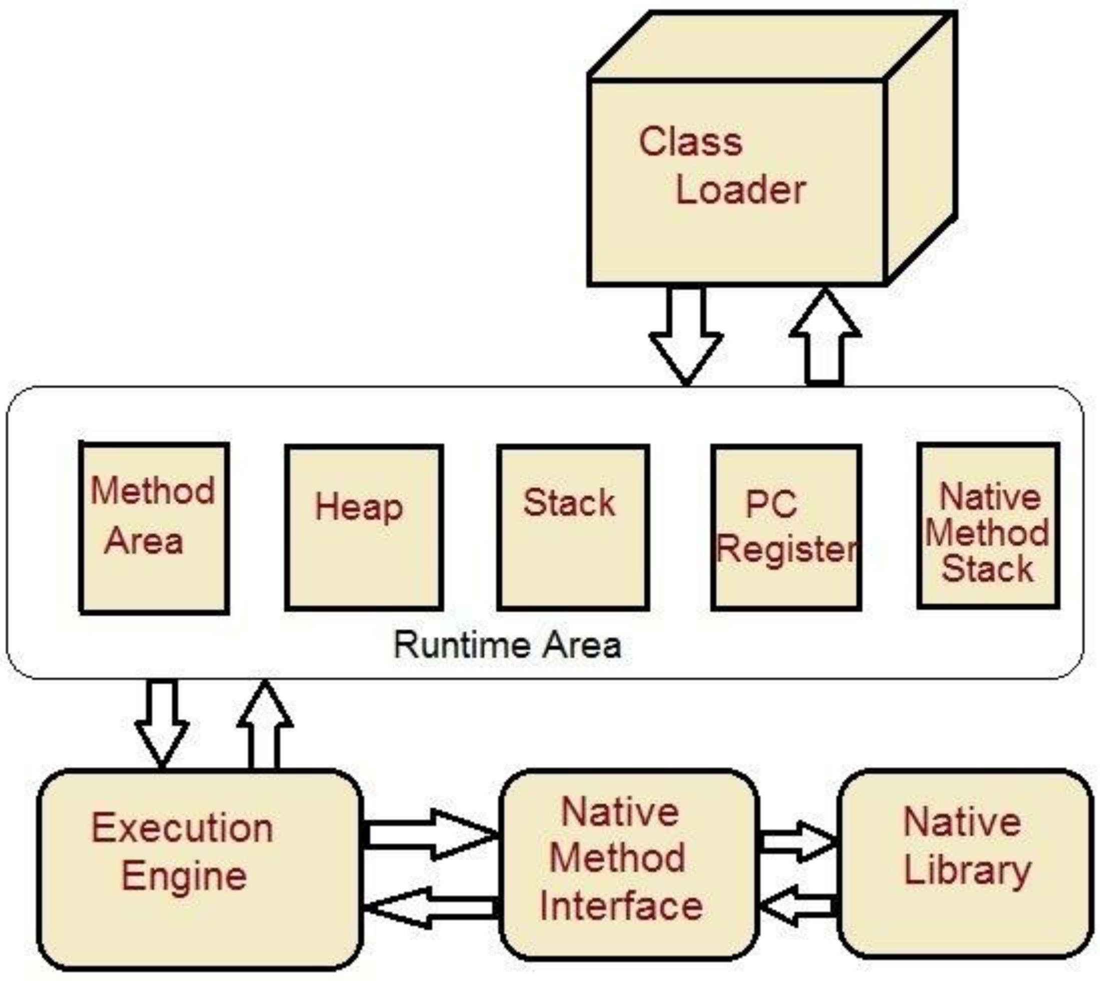
Java JVM, JDK and JRE

In this tutorial we will cover what Java Virtual Machine is, and what is JRE and JDK.

Java virtual Machine(JVM) is a virtual Machine that provides runtime environment to execute java byte code. The JVM doesn't understand Java typo, that's why you compile your `*.java` files to obtain `*.class` files that contain the bytecodes understandable by the JVM.

JVM control execution of every Java program. It enables features such as automated exception handling, Garbage-collected heap.

JVM Architecture



Class Loader : Class loader loads the Class for execution.

Method area : Stores pre-class structure as constant pool.

Heap : Heap is a memory area in which objects are allocated.

Stack : Local variables and partial results are store here. Each thread has a private JVM stack created when the thread is created.

Program register : Program register holds the address of JVM instruction currently being executed.

Native method stack : It contains all native used in application.

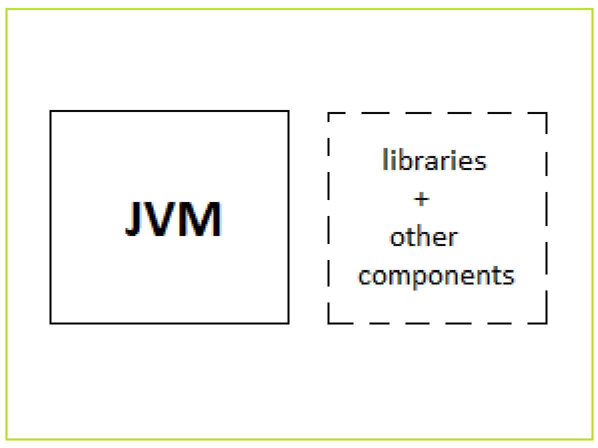
Executive Engine : Execution engine controls the execute of instructions contained in the methods of the classes.

Native Method Interface : Native method interface gives an interface between java code and native code during execution.

Native Method Libraries : Native Libraries consist of files required for the execution of native code.

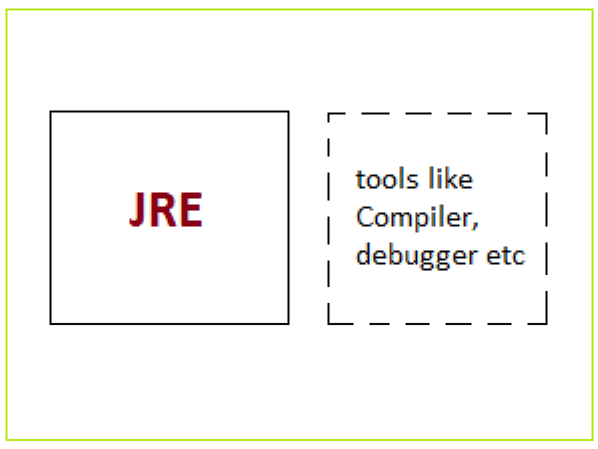
Difference between JDK and JRE

JRE : The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applets and applications written in the Java programming language. JRE does not contain tools and utilities such as compilers or debuggers for developing applets and applications.



JRE - Java Runtime Environment

JDK : The JDK also called Java Development Kit is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applets and applications.



JDK - Java Development Kit

Java Hello World! Program

Creating a Hello World Program in Java is not a single line program. It consists of various other lines of code. Since Java is a Object-oriented language so it require to write a code inside a class. Let us look at a simple java program.

```
class Hello

{
     public static void main(String[] args)

     {

     System.out.println ("Hello World program");

     }

}
```

Copy

Lets understand what above program consists of and its keypoints.

class : class keyword is used to declare classes in Java

public : It is an access specifier. Public means this function is visible to all.

static : static is again a keyword used to make a function static. To execute a static function you do not have to create an Object of the class. The **main()** method here is called by JVM, without creating any object for class.

void : It is the return type, meaning this function will not return anything.

main : main() method is the most important method in a Java program. This is the method which is executed, hence all the logic must be inside the main() method. If a java class is not having a main() method, it causes compilation error.

String[] args : This represents an array whose type is String and name is args. We will discuss more about array in Java Array section.

System.out.println : This is used to print anything on the console like *printf* in C language.

Steps to Compile and Run your first Java program

Step 1: Open a text editor and write the code as above.

Step 2: Save the file as Hello.java

Step 3: Open command prompt and go to the directory where you saved your first java program assuming it is saved in C drive.

Step 4: Type `javac Hello.java` and press Return**(Enter KEY)** to compile your code. This command will call the Java Compiler asking it to compile the specified file. If there are no errors in the code the command prompt will take you to the next line.

Step 5: Now type `java Hello` on command prompt to run your program.

Step 6: You will be able to see **Hello world program** printed on your command prompt.

Ways to write a Java Program

Following are some of the ways in which a Java program can be written:

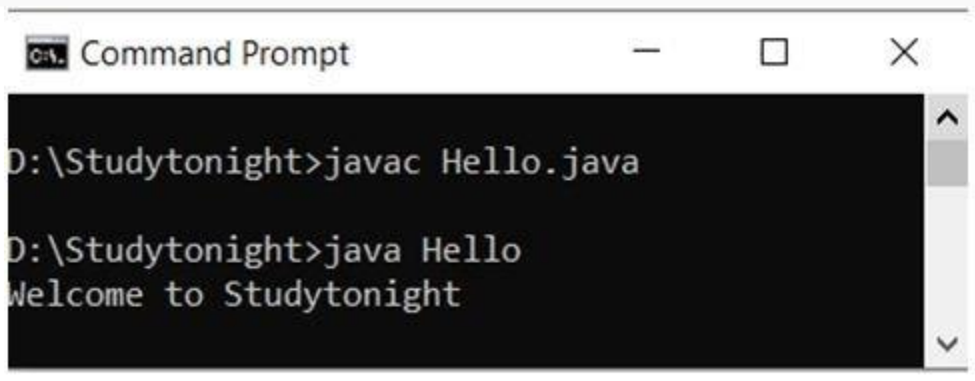
Changing the sequence of the modifiers and methods is accepted by Java.

Syntax: static public void main(String as[])

Example:

```
class Hello
{
	static public void main(String as[])
	{
	System.out.println ("Welcome to Studytonight");
	}
}
```

Copy



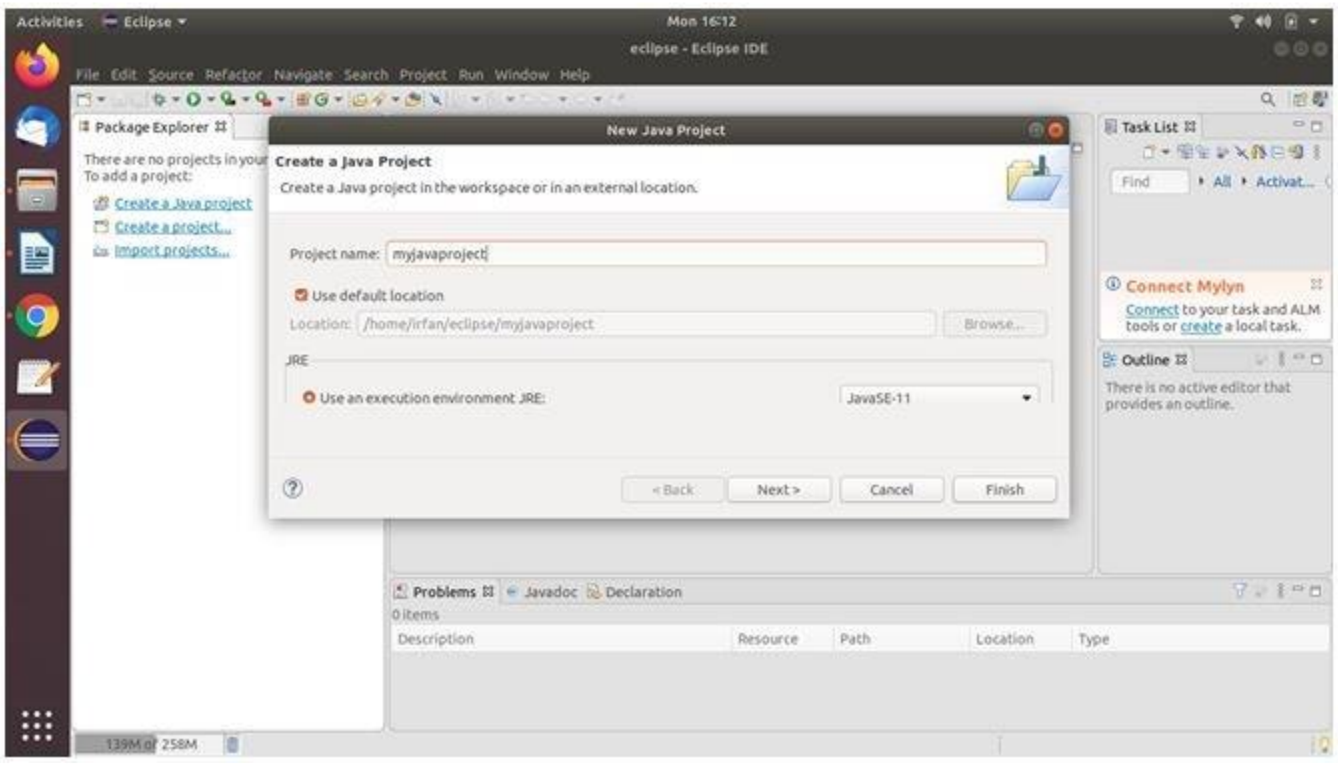
```
D:\Studytonight>javac Hello.java

D:\Studytonight>java Hello
Welcome to Studytonight
```

Hello World Program using Eclipse

Eclipse is an IDE (Integrated Development Environment) which is used to develop applications. It is design and developed by Eclipse foundation, if you don't have eclipse download, then download it from its official site by following this download link Download Eclipse from here Here we will see how to create and run **hello world** program using eclipse IDE. It require following steps that consists of **creating project, class file, writing code, running code etc**.

Run Eclipse and Create Project

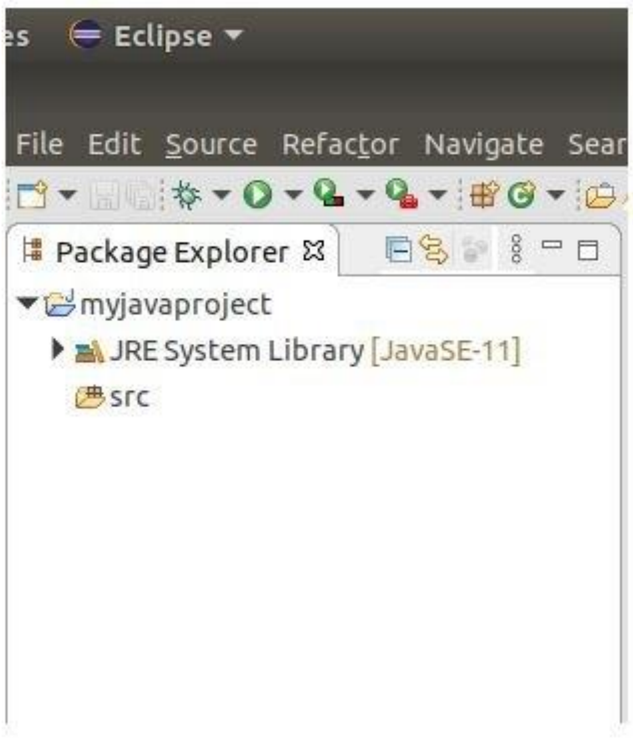
Open eclipse startup and then create new project. To create project click on **File** menu and select **Java project** option. It will open a window that ask for project name. Provide the project name and click on the finish button. See the below screenshot.



Create a Java Project
Create a Java project in the workspace or in an external location.

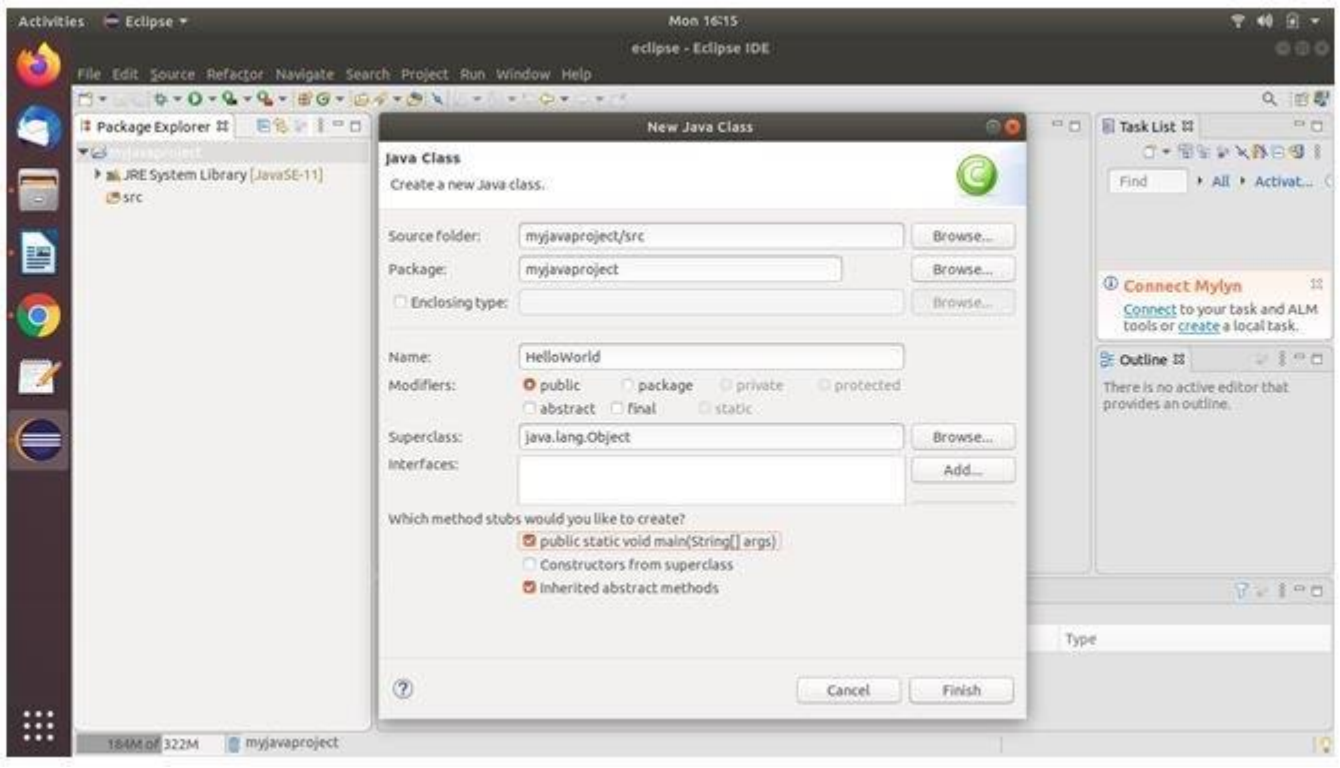
Project name: myjavaproject
[✓] Use default location
Location: /home/irfan/eclipse/myjavaproject
JRE
[✓ Use an execution environment JRE:] JavaSE-11

After creating project, we can see our new created project in the left side bar that looks like below.


Eclipse
File Edit Source Refactor Navigate Sear
Package Explorer
myjavaproject
JRE System Library [JavaSE-11]
src

Create Java Class

Now create Java class file by **right click** on the **project** and **select class** file option. It will open a window to ask for class name, provide the class name and click on finish button.

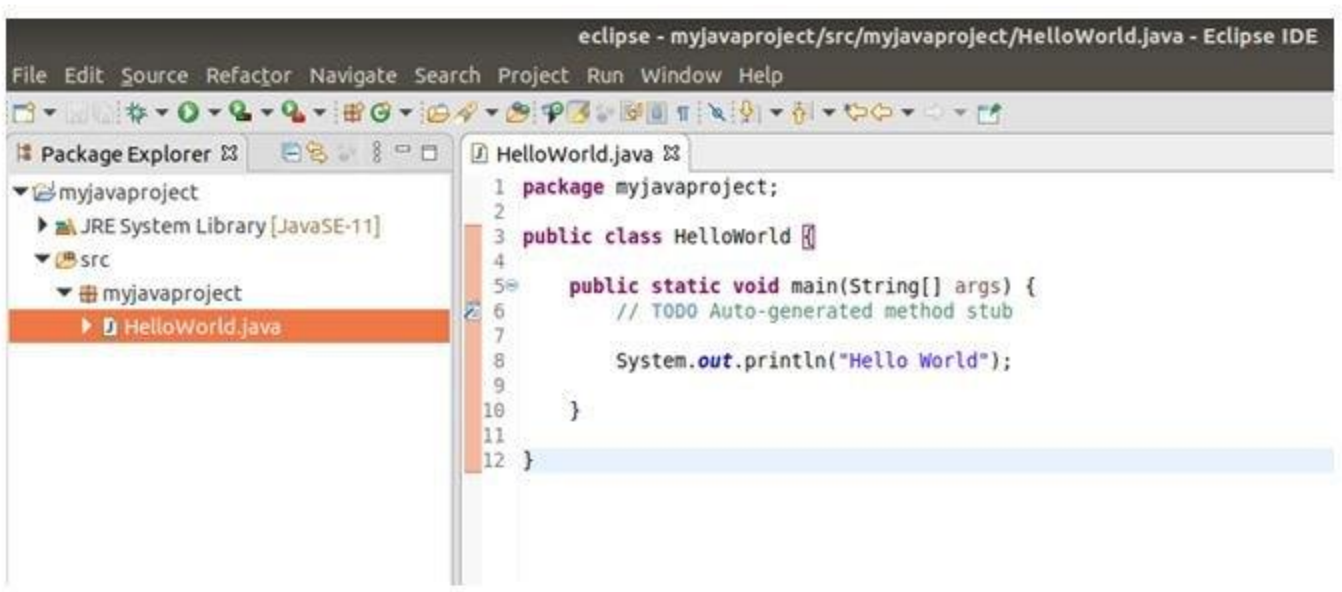

Activities
Eclipse
Mon 16:15
eclipse - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
myjavaproject
JRE System Library [JavaSE-11]
src
New Java Class
Task List
Connect Mylyn
Connect to your task and ALM tools or create a local task.
Outline
There is no active editor that provides an outline.
Type
184M of 322M
myjavaproject
Browse...
Add...
Cancel
Finish

Java Class
Create a new Java class.

Source folder: myjavaproject/src
Package: myjavaproject
[ ] Enclosing type:
Name: HelloWorld
Modifiers: [✓ public] package private protected
[ ] abstract [ ] final [ ] static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
[✓] public static void main(String[] args)
[ ] Constructors from superclass
[✓] Inherited abstract methods

Write Hello World

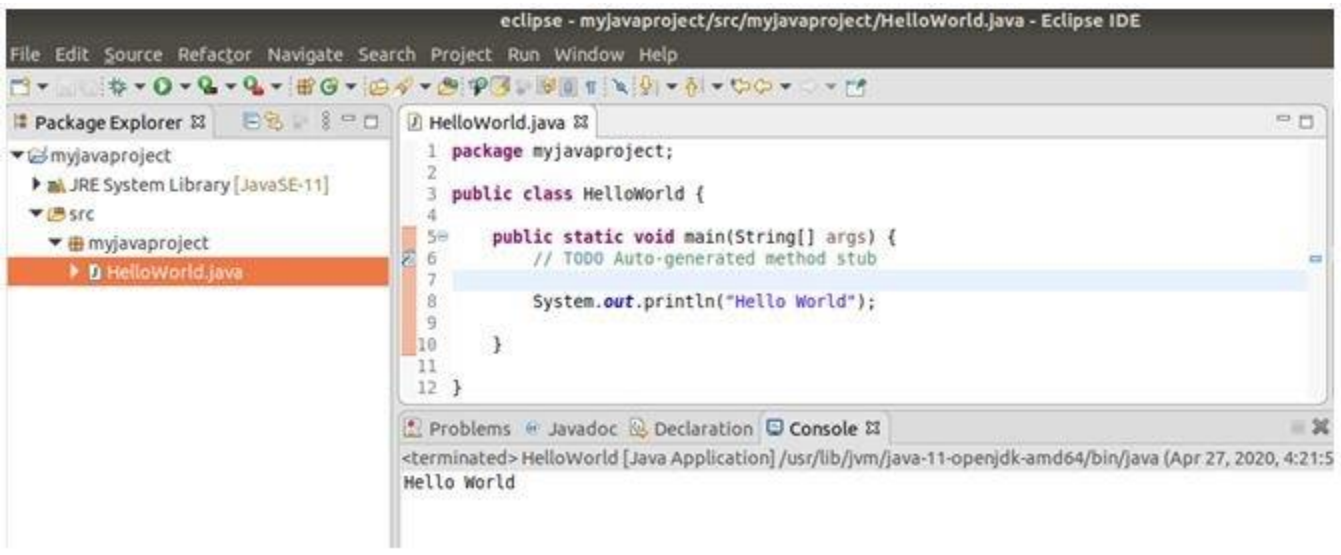
The above created class file includes some line of codes including main method as well. Now we need to write just print statement to print Hello World message.


eclipse - myjavaproject/src/myjavaproject/HelloWorld.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
myjavaproject
JRE System Library [JavaSE-11]
src
myjavaproject
HelloWorld.java
HelloWorld.java

```
package myjavaproject;

public class HelloWorld {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        System.out.println("Hello World");

    }

}
```

Run The Program

Now run the program by selecting **Run** menu from the menu bar or use **Ctrl+F11** button combination. After running, it will print Hello World to the console which is just bottom to the program window.


eclipse - myjavaproject/src/myjavaproject/HelloWorld.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Package Explorer
myjavaproject
JRE System Library [JavaSE-11]
src
myjavaproject
HelloWorld.java
HelloWorld.java
Problems
Javadoc
Declaration
Console

```
package myjavaproject;

public class HelloWorld {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        System.out.println("Hello World");

    }

}
```

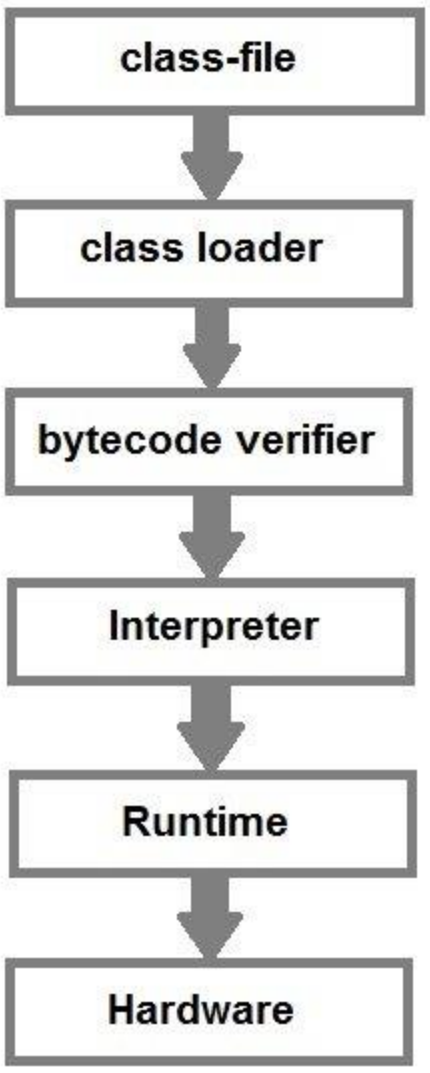
```
<terminated> HelloWorld [Java Application] /usr/lib/jvm/java-11-openjdk-amd64/bin/java (Apr 27, 2020, 4:21:5
Hello World
```

This is a simple program that we run here while using IDE we can create and build large scale of applications. If you are a beginner and not familiar to the Eclipse then don’t worry it is very easy to operate just follow the above steps to create the program.

Now let us see What happens at Runtime

After writing your Java program, when you will try to compile it. Compiler will perform some compilation operation on your program.

Once it is compiled successfully byte code(.class file) is generated by the compiler.



After compiling when you will try to run the byte code(.class file), the following steps are performed at runtime:-

1. Class loader loads the java class. It is subsystem of JVM Java Virtual machine.
2. Byte Code verifier checks the code fragments for illegal codes that can violate access right to the object.
3. Interpreter reads the byte code stream and then executes the instructions, step by step.

Variable in Java

In this tutorial we will learn about Java variables, various types of variables along with code examples to understand Java variables.

What is a Variable?

When we want to store any information, we store it in an address of the computer. Instead of remembering the complex address where we have stored our information, we name that address.The naming of an address is known as variable. Variable is the name of memory location.

In other words, variable is a name which is used to store a value of any type during program execution.

To declare the variable in Java, we can use following syntax

```
datatype variableName;
```

Copy

Here, **datatype** refers to type of variable which can any like: **int, float** etc. and **variableName** can be any like: **empId, amount, price** etc.

Java Programming language defines mainly three kind of variables.

1. Instance Variables
2. Static Variables (Class Variables)
3. Local Variables

Instance variables in Java

Instance variables are variables that are declare inside a class but outside any method,constructor or block. Instance variable are also variable of object commonly known as field or property. They are referred as object variable. Each object has its own copy of each variable and thus, it doesn't effect the instance variable if one object changes the value of the variable.

```
class Student

{

    String name;

    int age;

}
```

Copy

Here **name** and **age** are instance variable of Student class.

Static variables in Java

Static are class variables declared with static keyword. Static variables are initialized only once. Static variables are also used in declaring constant along with final keyword.

```
class Student

{

    String name;

    int age;

    static int instituteCode=1101;

}
```

Copy

Here **instituteCode** is a static variable. Each object of Student class will share instituteCode property.

Additional points on static variable:

- static variable are also known as class variable.

- static means to remain constant.
- In Java, it means that it will be constant for all the instances created for that class.
- static variable need not be called from object.
- It is called by *classname.static_variable_name*

Note: A static variable can never be defined inside a method i.e it can never be a local variable.

Example:

Suppose you make 2 objects of class Student and you change the value of static variable from one object. Now when you print it from other object, it will display the changed value. This is because it was declared static i.e it is constant for every object created.

```
package studytonight;


class Student{

    int a;

    static int id = 35;


    void change(){


        System.out.println(id);

    }

}


public class StudyTonight {

    public static void main(String[] args) {


        Student o1 = new Student();

        Student o2 = new Student();


        o1.change();


        Student.id = 1;

        o2.change();

    }
}
```

Copy

```
35

1
```

Local variables in Java

Local variables are declared in method, constructor or block. Local variables are initialized when method, constructor or block start and will be destroyed once its end. Local variable reside in stack. Access modifiers are not used for local variable.

```
float getDiscount(int price)

{

 float discount;

 discount=price*(20/100);

 return discount;

}
```

Copy

Here **discount** is a local variable.

Variable Scope in Java

Scope of a variable decides its accessibility throughout the program. As we have seen variables are different types so they have their own scope.

Local variable: Scope of local variable is limited to the block in which it is declared. For example, a variables declared inside a function will be accessible only within this function.

Instance variable: scope of instance variable depends on the access-modifiers **(public, private, default)**. If variable is declared as **private** then it is accessible within class only.

If variable is declared as **public** then it is accessible for all and throughout the application.

If variable is declared as **default** the it is accessible with in the same package.

For more details about accessibility you can refer our detailed tutorial. Click here to know more about Variable Scope

Example:

In this example, we created two variables **a** and **i**, first is declared inside the function and second is declared inside for loop. Both variables have their own scope in which they are declared, so accessing outside the block reports an error.

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

        int a = 10;

        for(int i = 0; i<5; i++) {
            System.out.println(i);
        }

        System.out.println("a = "+a);
        System.out.println("i = "+i); // error

    }

}
```
Copy

```
OUTPUT
error: cannot find symbol i
```

Data Types in Java

Java language has a rich implementation of data types. Data types specify size and the type of values that can be stored in an identifier.

In java, data types are classified into two catagories :

1. Primitive Data type
2. Non-Primitive Data type

---

1) Primitive Data type

A primitive data type can be of eight types :

Primitive Data types

char	boolean	byte	short	int	long	float	double

Once a primitive data type has been declared its type can never change, although in most cases its value can change. These eight primitive type can be put into four groups

Integer

This group includes byte, short, int, long

byte : It is 1 byte(8-bits) integer data type. Value range from -128 to 127. Default value zero. example: byte b=10;

short : It is 2 bytes(16-bits) integer data type. Value range from -32768 to 32767. Default value zero. example: short s=11;

int : It is 4 bytes(32-bits) integer data type. Value range from -2147483648 to 2147483647. Default value zero. example: int i=10;

long : It is 8 bytes(64-bits) integer data type. Value range from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807. Default value zero. example: long l=100012;

Example:

Lets create an example in which we work with **integer type** data and we can get idea how to use datatype in the java program.

```
package corejava;

public class Demo{

    public static void main(String[] args) {
     // byte type
     byte b = 20;
     System.out.println("b= "+b);

     // short type
     short s = 20;
     System.out.println("s= "+s);

     // int type
     int i = 20;
     System.out.println("i= "+i);
```

```
      // long type

      long l = 20;

      System.out.println("l= "+l);




     }

}
```

Copy

```
b= 20

s= 20

i= 20

l= 20
```

Floating-Point Number

This group includes float, double

float : It is 4 bytes(32-bits) float data type. Default value 0.0f. example: float ff=10.3f;

double : It is 8 bytes(64-bits) float data type. Default value 0.0d. example: double db=11.123;

Example:

In this example, we used floating point type and declared variables to hold floating values. Floating type is **useful to store decimal point values.**

```
public class Demo{


    public static void main(String[] args) {

      // float type

      float f = 20.25f;

      System.out.println("f= "+f);


      // double type

      double d = 20.25;

      System.out.println("d= "+d);
    }

}
```

Copy

```
f= 20.25

d= 20.25
```

Characters

This group represent `char`, which represent symbols in a character set, like letters and numbers.

char : It is 2 bytes(16-bits) unsigned unicode character. Range 0 to 65,535. example: `char c='a';`

Char Type Example

Char type in Java uses 2 bytes to unicode characters. Since it works with unicode then we can store alphabet character, currency character or other characters that are comes under the unicode set.

```
public class Demo {

    public static void main(String[] args) {


        char ch = 'S';

        System.out.println(ch);


        char ch2 = '&';

        System.out.println(ch2);


        char ch3 = '$';

        System.out.println(ch3);


    }


}
```

```
```

Copy

```
S
&
$
```

Boolean

This group represent boolean, which is a special type for representing true/false values. They are defined constant of the language. example: boolean b=true;

Boolean Type Example

Boolean type in Java works with two values only either true or false. It is mostly use in conditional expressions to perform conditional based programming.

```
public class Demo {

    public static void main(String[] args) {

        boolean t = true;
        System.out.println(t);

        boolean f = false;
        System.out.println(f);

    }

}
```

Copy

```
true
false
```

2) Non-Primitive(Reference) Data type

A reference data type is used to refer to an object. A reference variable is declare to be of specific and that type can never be change.

For example: **String str**, here str is a reference variable of type String. String is a class in Java. We will talk a lot more about reference data type later in Classes and Object lesson.

Reference type are used to hold reference of an object. Object can be instance of any class or entity.

In object oriented system, we deal with object that stores properties. To refer those objects we use reference types.

We will talk a lot more about reference data type later in Classes and Object lesson.

Identifiers in Java

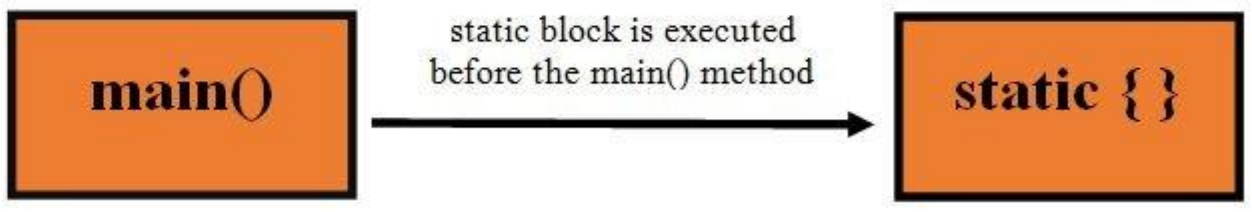
All Java components require names. Name used for classes, methods, interfaces and variables are called **Identifier**. Identifier must follow some rules. Here are the rules:

- All identifiers must start with either a letter(a to z or A to Z) or currency character($) or an underscore.
- After the first character, an identifier can have any combination of characters.
- Java **keywords** cannot be used as an identifier.
- Identifiers in Java are case sensitive, foo and Foo are two different identifiers.

Some valid identifiers are: **int a, class Car, float amount** etc.

Static Block in Java

In Java, the static keyword is used for the management of memory mainly. the static keyword can be used with Variables, Methods, Block and nested class. A static block in a program is a set of statements which are executed by the JVM (Java Virtual Machine) before the main method. At the time of class loading, if we want to perform any task we can define that task inside the static block, this task will be executed at the time of class loading. In a class, any number of a static block can be defined, and this static blocks will be executed from top to bottom.



Syntax:

```
static {

            *********

            *********

            // statements….

            *********

            *********

      }
```

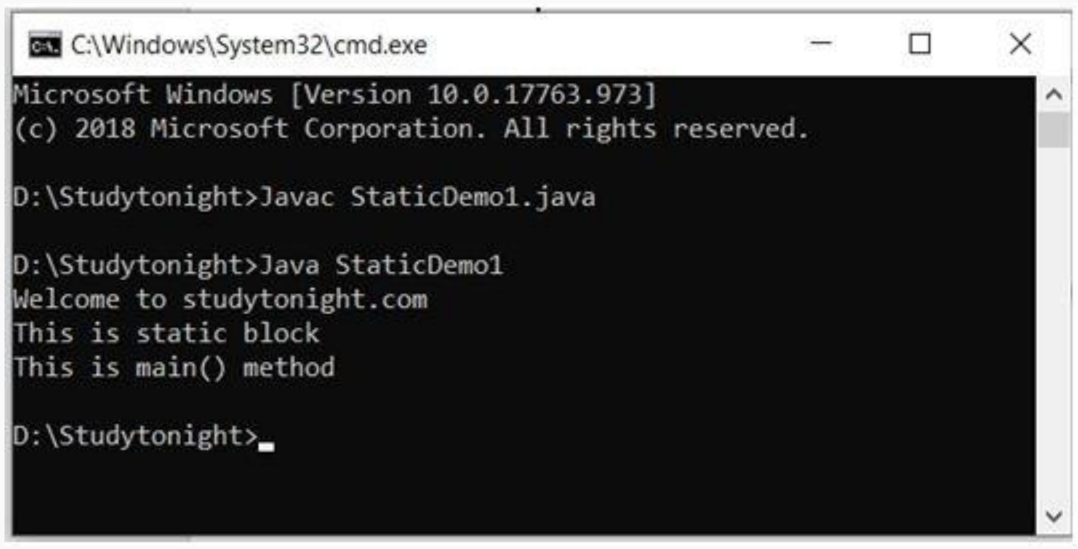
Copy

Example of a static block

Static block executes before the main method while executing program. Statements written inside the static block will execute first. However both are static.

```
class StaticDemo1

{

        static

        {

                System.out.println("Welcome to studytonight.com");

                System.out.println("This is static block");



        }

        public static void main(String as[])

        {

                System.out.println("This is main() method");

        }

}
```

Copy



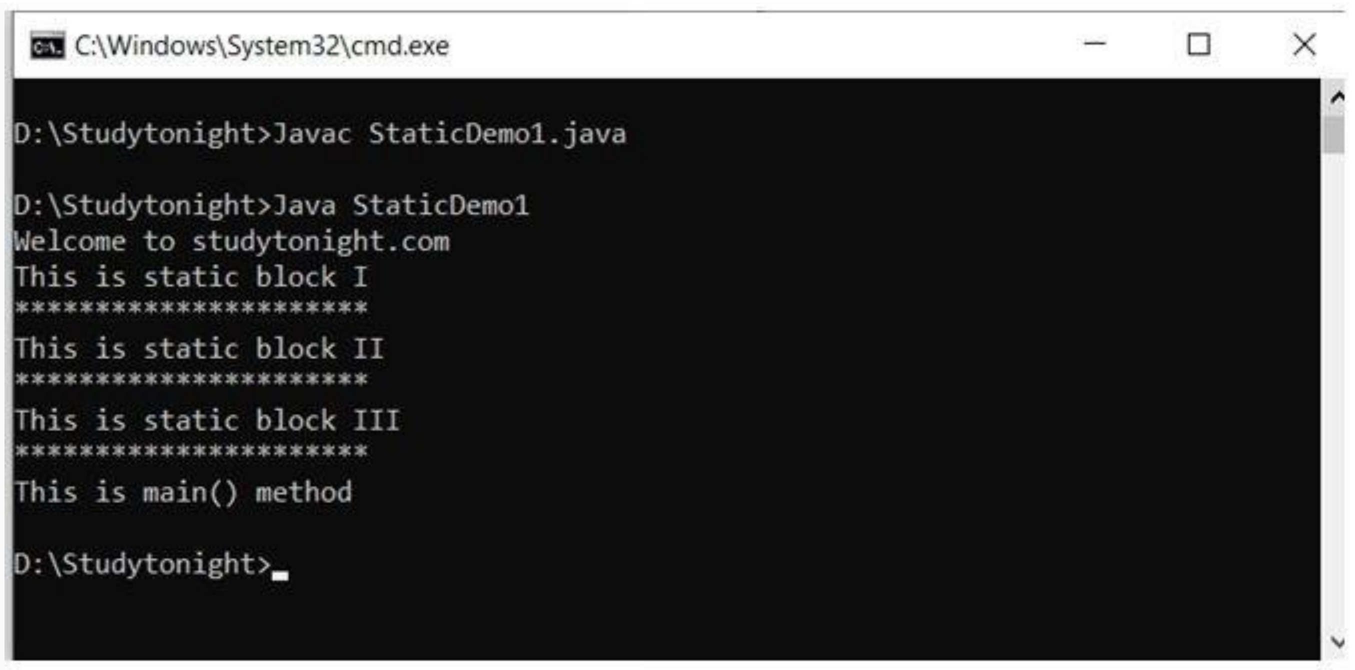
```
Microsoft Windows [Version 10.0.17763.973]
(c) 2018 Microsoft Corporation. All rights reserved.

D:\Studytonight>Javac StaticDemo1.java

D:\Studytonight>Java StaticDemo1
Welcome to studytonight.com
This is static block
This is main() method

D:\Studytonight>
```

Example of multiple static blocks

When we have multiple static blocks then each block executes in the sequence. First static block will execute first.

```
class StaticDemo1

{
        static

        {

                System.out.println("Welcome to studytonight.com");

                System.out.println("This is static block I");


        }

        public static void main(String as[])

        {

                System.out.println("*********************");

                System.out.println("This is main() method");

        }

        static

        {

                System.out.println("*********************");

                System.out.println("This is static block II");


        }

        static

        {

                System.out.println("*********************");

                System.out.println("This is static block III");


        }

}
```

Copy


C:\Windows\System32\cmd.exe

```
D:\Studytonight>Javac StaticDemo1.java

D:\Studytonight>Java StaticDemo1
Welcome to studytonight.com
This is static block I
**********************
This is static block II
**********************
This is static block III
**********************
This is main() method

D:\Studytonight>
```

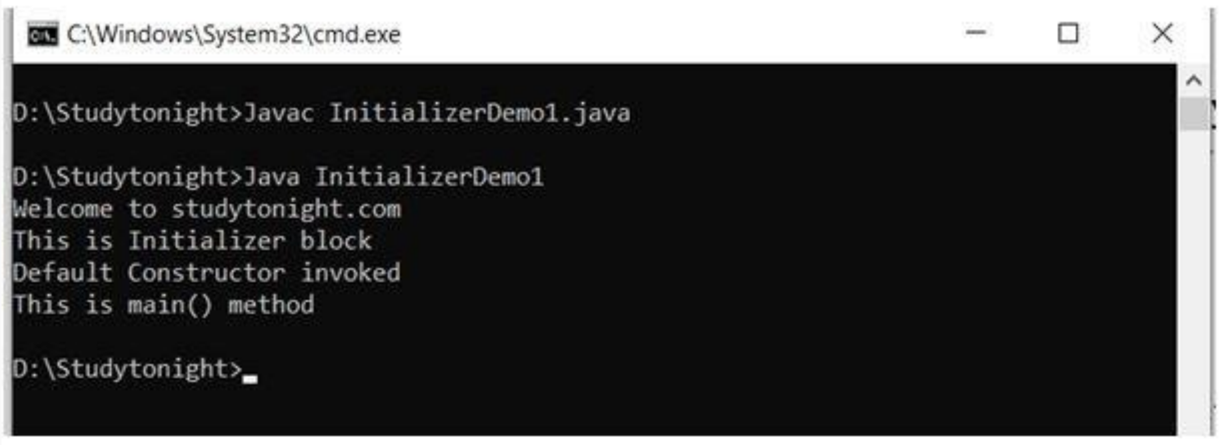
Initializer Block in Java

In Java, the initializer Block is used to initialize instance data members. The initializer block is executed whenever an object is created. The Initializer block is copied into Java compiler and then to every constructor. The initialization block is executed before the code in the constructor.

Example:

```java
class InitializerDemo1

{

        {

                System.out.println("Welcome to studytonight.com");

                System.out.println("This is Initializer block");



        }

        public InitializerDemo1()

        {

                System.out.println("Default Constructor invoked");

        }

        public static void main(String as[])

        {

                InitializerDemo1 obj = new InitializerDemo1();

                System.out.println("This is main() method");

        }

}
```

Copy



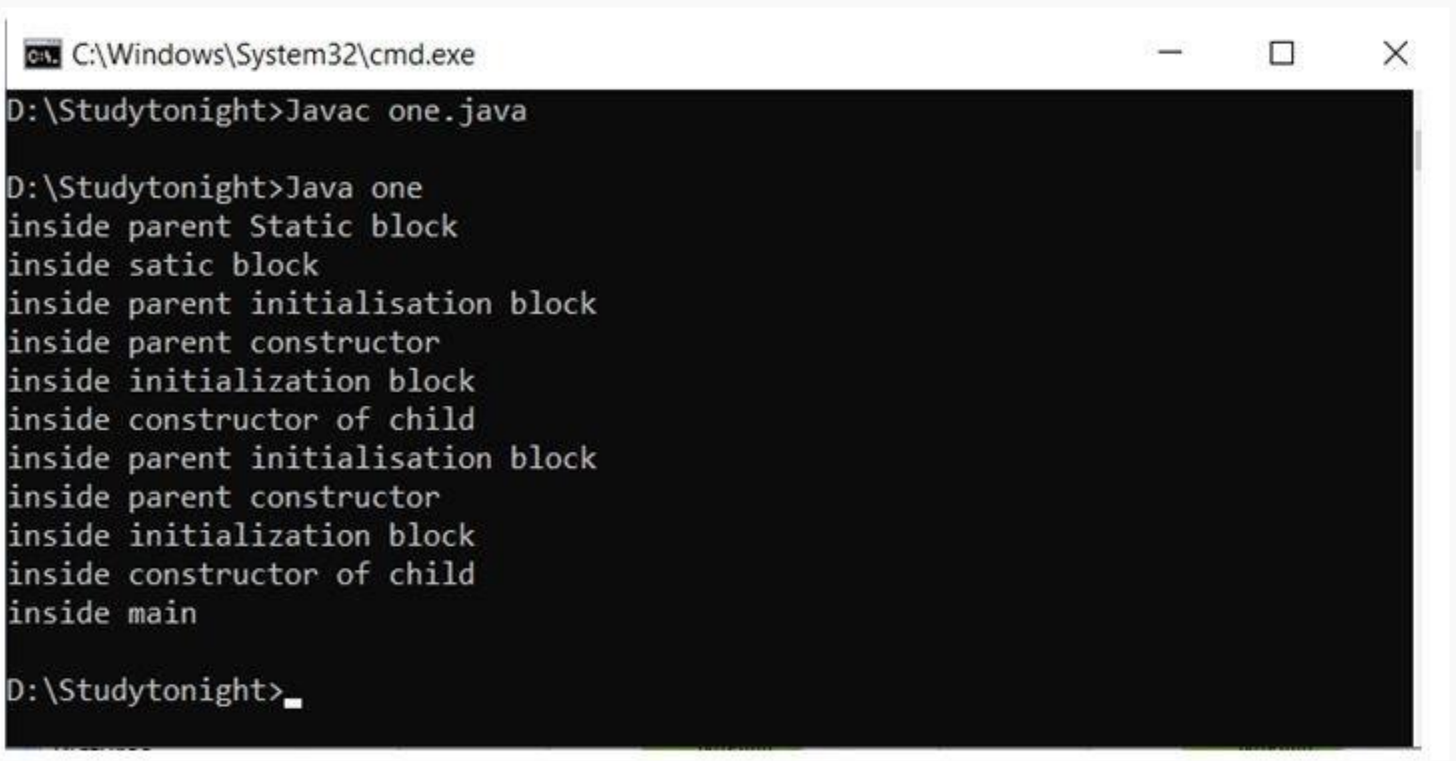
```
D:\Studytonight>Javac InitializerDemo1.java

D:\Studytonight>Java InitializerDemo1
Welcome to studytonight.com
This is Initializer block
Default Constructor invoked
This is main() method

D:\Studytonight>_
```

Example using static and initializer block

We can have both static and initializer blocks in a Java program. But static block will execute first even before initializer block. See the below example.

```
public class one extends two {

    static {

System.out.println("inside satic block");

    }



one() {

System.out.println("inside constructor of child");

    }



    {

System.out.println("inside initialization block");

    }



    public static void main(String[] args) {

        new one();

        new one();

System.out.println("inside main");

    }

}



class two{

    static {

System.out.println("inside parent Static block");

    }
    {

System.out.println("inside parent initialisation block");

    }



two() {

System.out.println("inside parent constructor");

    }

}
```

Copy


C:\Windows\System32\cmd.exe

```
D:\Studytonight>Javac one.java

D:\Studytonight>Java one
inside parent Static block
inside satic block
inside parent initialisation block
inside parent constructor
inside initialization block
inside constructor of child
inside parent initialisation block
inside parent constructor
inside initialization block
inside constructor of child
inside main

D:\Studytonight>
```

Type Casting in Java

Casting is a process of changing one type value to another type. In Java, we can cast one type of value to another type. It is known as type casting.

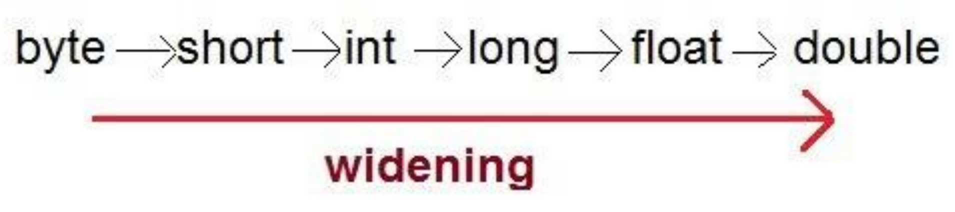
Example :

```
int x = 10;

byte y = (byte)x;
```

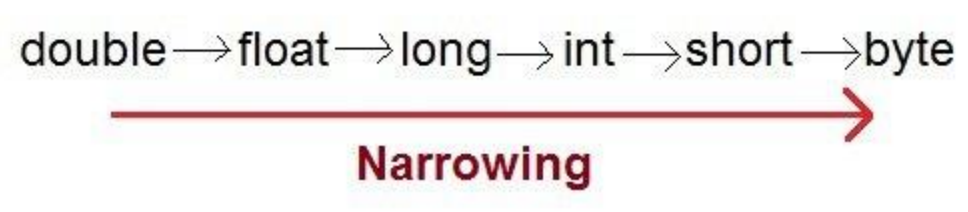
Copy

In Java, type casting is classified into two types,

- Widening Casting(Implicit)



- Narrowing Casting(Explicitly done)



Widening or Automatic type conversion

Automatic Type casting take place when,

- the two types are compatible
- the target type is larger than the source type

Example:

```
public class Test

{

    public static void main(String[] args)

    {

      int i = 100;

      long l = i; //no explicit type casting required

      float f = l;  //no explicit type casting required

      System.out.println("Int value "+i);

      System.out.println("Long value "+l);

      System.out.println("Float value "+f);

    }


}
```

Copy

```
Int value 100

Long value 100

Float value 100.0
```

Narrowing or Explicit type conversion

When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting. If we don't perform casting then compiler reports compile time error.

Example :

```
public class Test

{

    public static void main(String[] args)

    {

      double d = 100.04;

      long l = (long)d;  //explicit type casting required

      int i = (int)l; //explicit type casting required


      System.out.println("Double value "+d);

      System.out.println("Long value "+l);

      System.out.println("Int value "+i);

    }


}
```

Copy

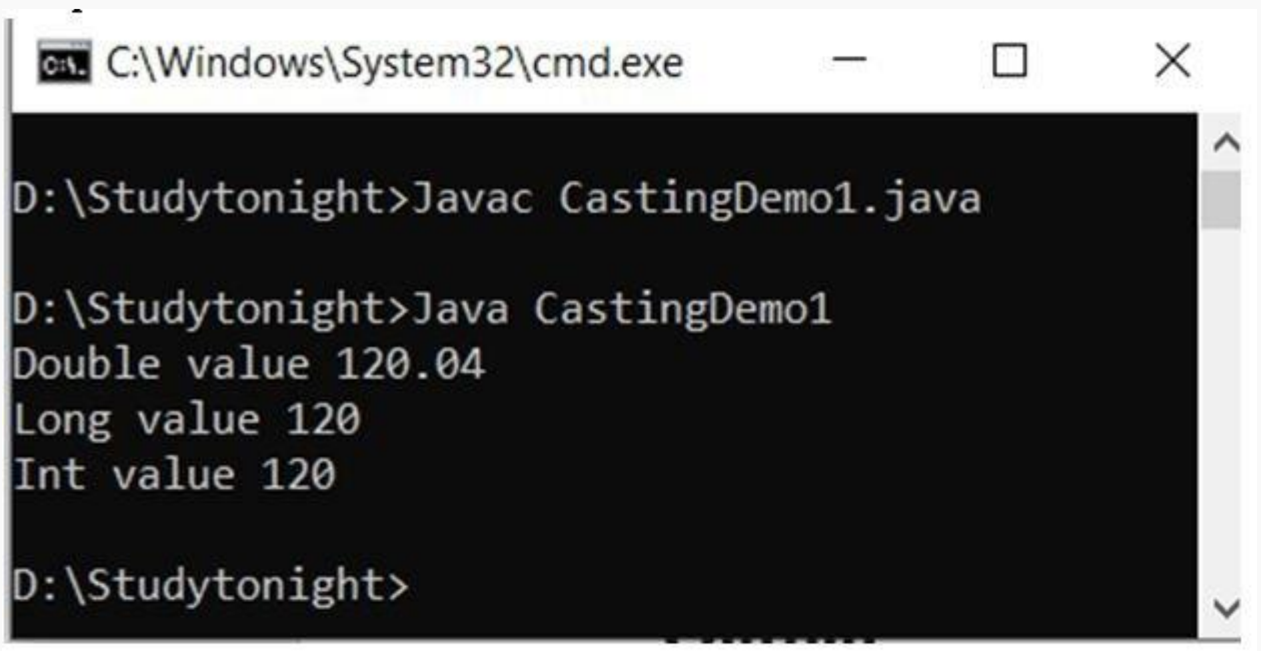
```
Double value 100.04

Long value 100

Int value 100
```

Example of Explicit Conversion

Here, we have one more example of explicit casting, double type is stored into long, long is stored into int etc.

```

class CastingDemo1

{

    public static void main(String[] args)

    {

      double d = 120.04;

      long l = (long)d;

      int i = (int)l;

      System.out.println("Double value "+d);
      System.out.println("Long value "+l);

      System.out.println("Int value "+i);

    }

}
```

Copy



```
D:\Studytonight>Javac CastingDemo1.java

D:\Studytonight>Java CastingDemo1
Double value 120.04
Long value 120
Int value 120

D:\Studytonight>
```

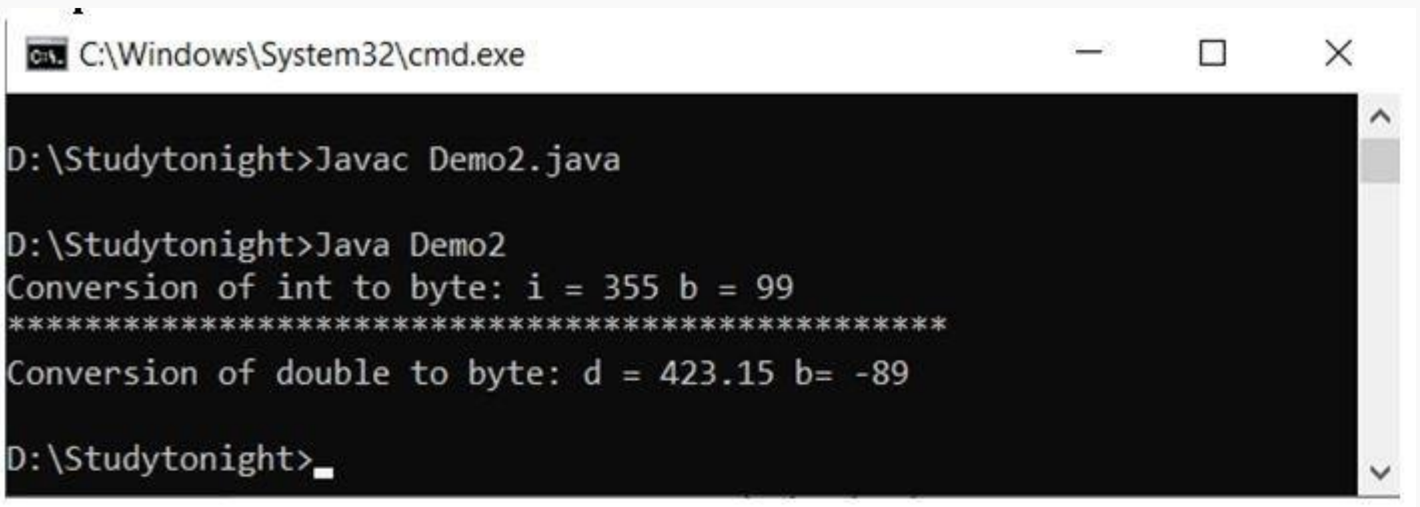
Example for Conversion of int and double into a byte

Here, we are converting int and double type to byte type by using explicit type casting.

```
class Demo2

{

    public static void main(String args[])

    {

      byte b;

      int i = 355;

      double d = 423.150;

      b = (byte) i;

      System.out.println("Conversion of int to byte: i = " + i + " b =
" + b);

System.out.println("*************************************************")
;

      b = (byte) d;
```

```
        System.out.println("Conversion of double to byte: d = " + d + "
b= " + b);

   }

}
```

Copy



```
C:\Windows\System32\cmd.exe

D:\Studytonight>Javac Demo2.java

D:\Studytonight>Java Demo2
Conversion of int to byte: i = 355 b = 99
**************************************************
Conversion of double to byte: d = 423.15 b= -89

D:\Studytonight>_
```

Java If Else Statement

In Java, if statement is used for testing the conditions. The condition matches the statement it returns true else it returns false. There are four types of If statement they are:

For example, if we want to create a program to **test positive integers** then we have to test the integer whether it is greater that zero or not.

In this scenario, if statement is helpful.

There are four types of if statement in Java:

i. if statement
ii. if-else statement
iii. if-else-if ladder
iv. nested if statement

if Statement

The if statement is a single conditional based statement that executes only if the provided condition is true.

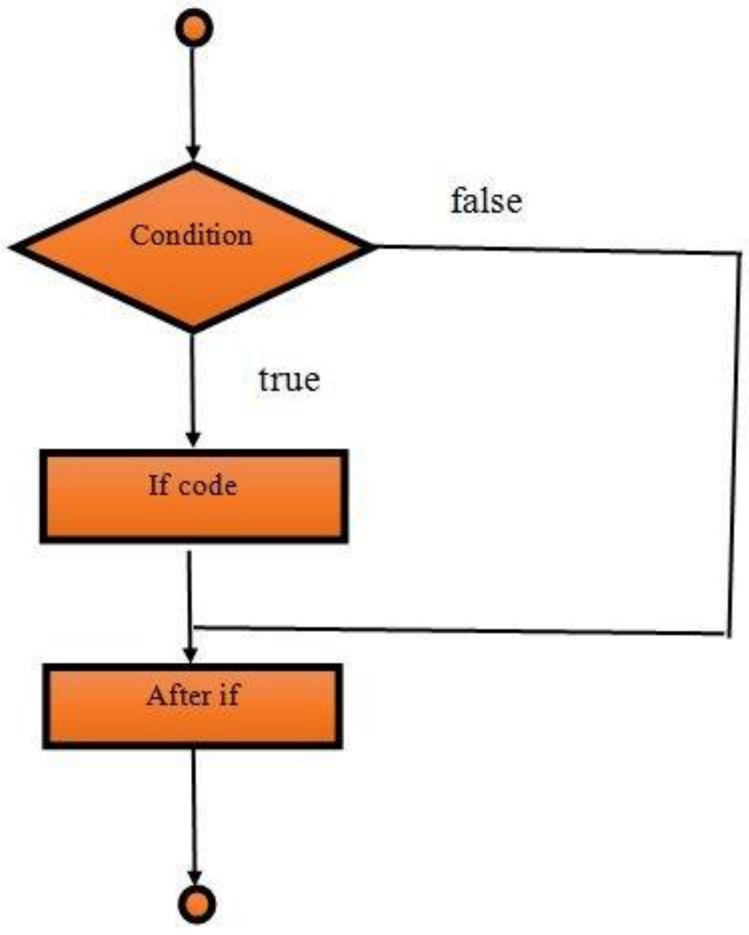
If Statement Syntax:

```
if(condition)

{

      //code

}
```

Copy

We can understand flow of if statement by using the below diagram. It shows that **code written inside the if will execute only if the condition is true.**

Data-flow-diagram of If Block

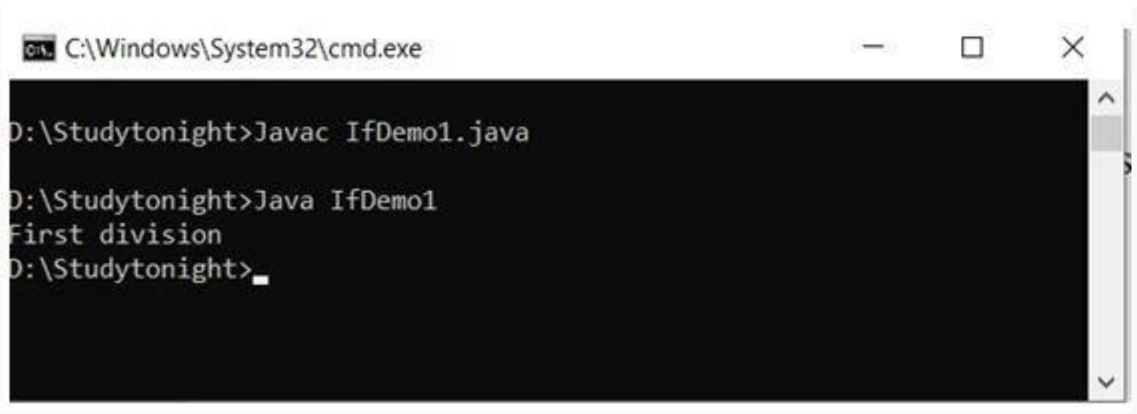


Example:

In this example, we are testing students marks. If the marks are greater than 65 then student will get first division.

```
public class IfDemo1 {

public static void main(String[] args)

     {

     int marks=70;

     if(marks > 65)

          {

          System.out.print("First division");

          }

     }

}
```

Copy



```
D:\Studytonight>Javac IfDemo1.java

D:\Studytonight>Java IfDemo1
First division
D:\Studytonight>
```

if-else Statement

The if-else statement is used for testing condition. If the condition is true, if block executes otherwise else block executes.

It is useful in the scenario when we want to perform some operation based on the **false** result.

The else block execute only when condition is **false**.

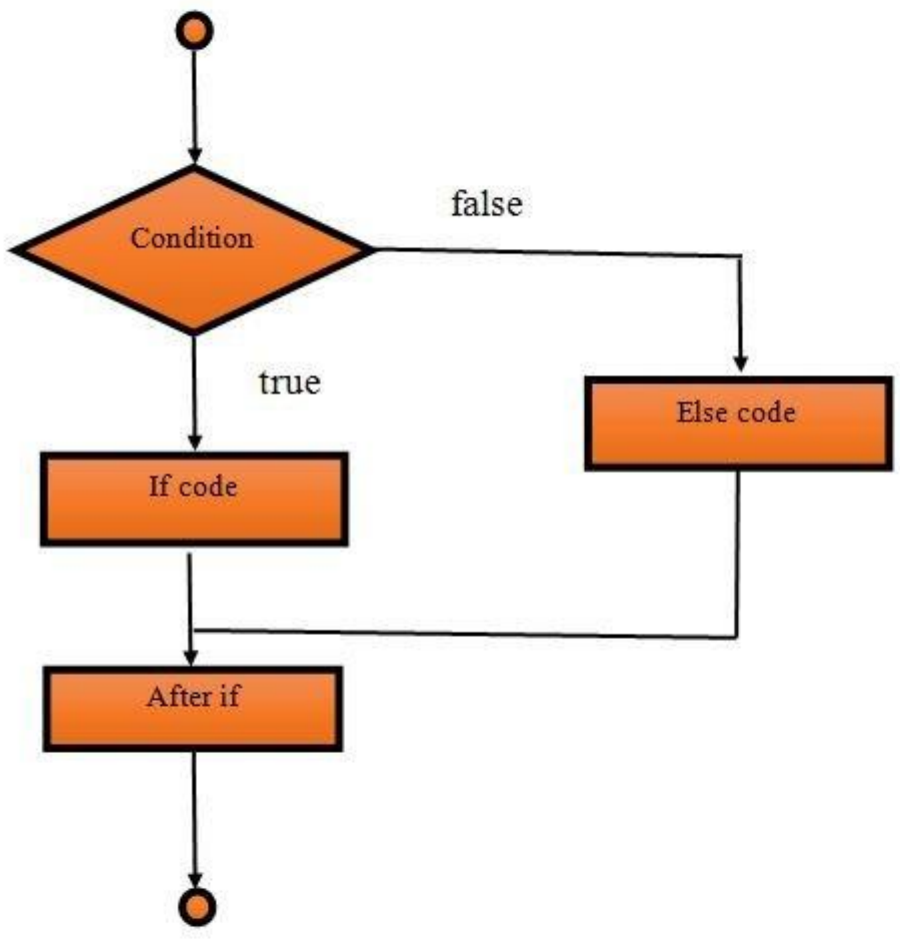
Syntax:

```
if(condition)

{

     //code for true

}

else

{

     //code for false

}
```

Copy

In this block diagram, we can see that when condition is true, if block executes otherwise else block executes.

Data-flow-diagram of If Else Block

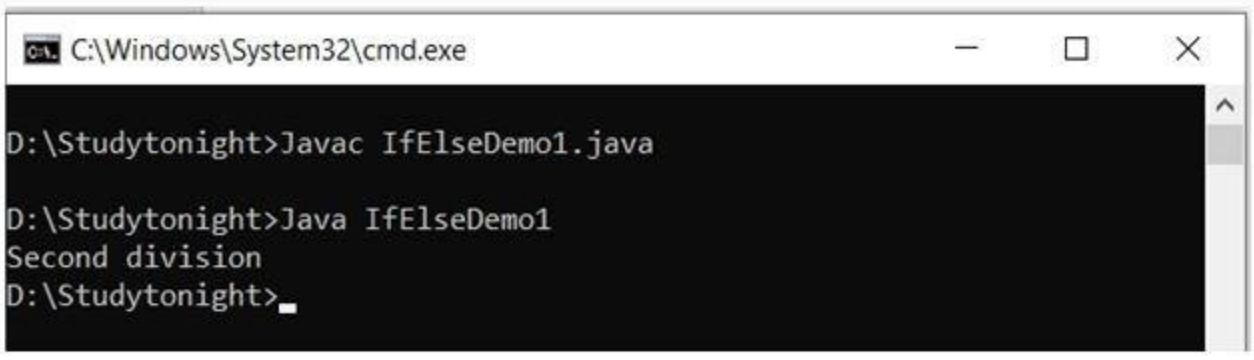


if else Example:

In this example, we are testing student marks, if marks is greater than 65 then if block executes otherwise else block executes.

```
public class IfElseDemo1 {
     public static void main(String[] args)
     {
          int marks=50;
          if(marks > 65)
          {
               System.out.print("First division");
          }
          else
          {
               System.out.print("Second division");
          }
     }
}
```

Copy



```
D:\Studytonight>Javac IfElseDemo1.java

D:\Studytonight>Java IfElseDemo1
Second division
D:\Studytonight>_
```

if-else-if ladder Statement

In Java, the if-else-if ladder statement is used for testing conditions. It is used for testing one condition from multiple statements.

When we have multiple conditions to execute then it is recommend to use if-else-if ladder.

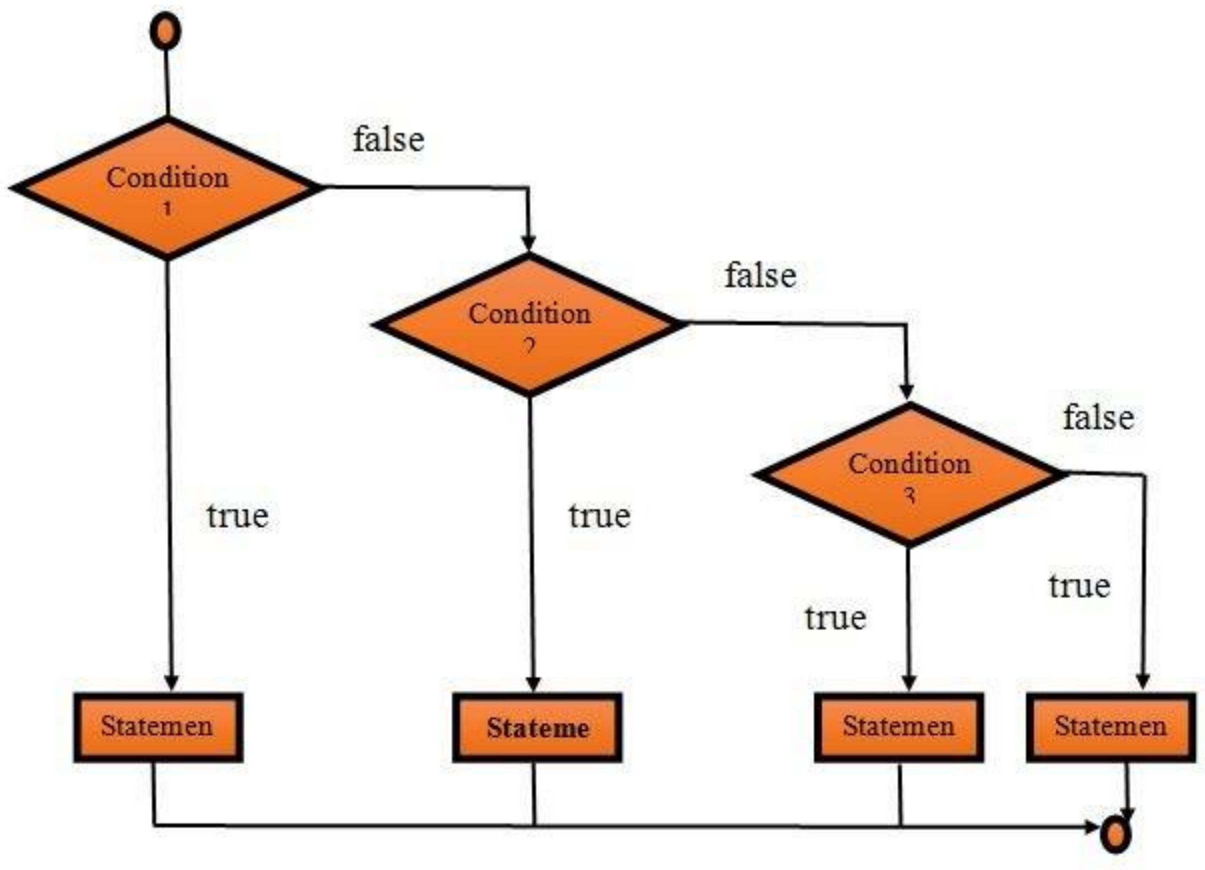
Syntax:

```
if(condition1)
{
     //code for if condition1 is true
}
else if(condition2)
{
     //code for if condition2 is true
}
else if(condition3)
{
     //code for if condition3 is true
}
...
else
{
     //code for all the false conditions
}
```

Copy

It contains multiple conditions and execute if any condition is true otherwise executes else block.

Data-flow-diagram of If Else If Block



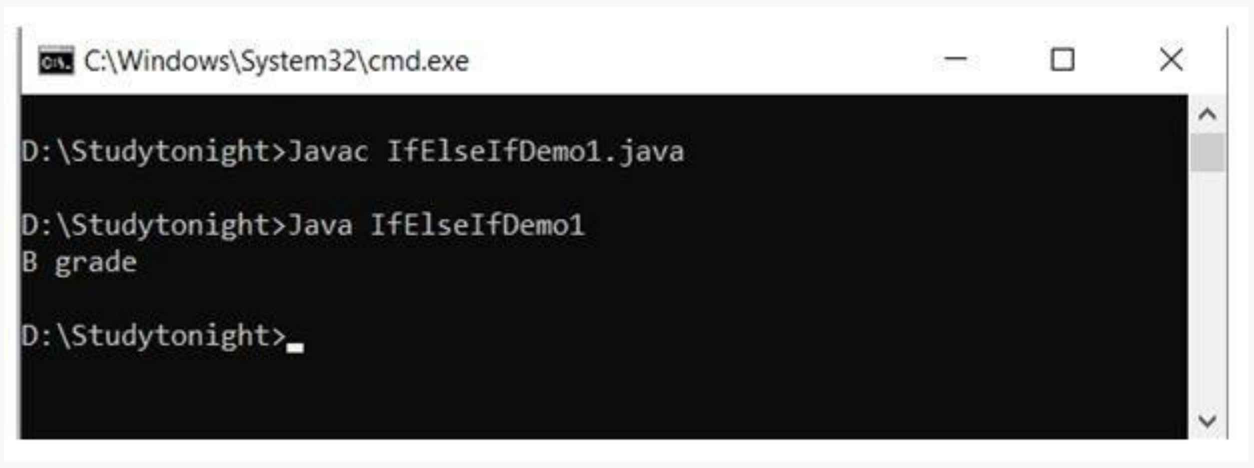
Example:

Here, we are testing student marks and displaying result based on the obtained marks. If marks are greater than 50 student gets his grades.

```
public class IfElseIfDemo1 {

public static void main(String[] args) {

int marks=75;

    if(marks<50){

System.out.println("fail");

    }

    else if(marks>=50 && marks<60){

System.out.println("D grade");

    }

    else if(marks>=60 && marks<70){

System.out.println("C grade");

    }

    else if(marks>=70 && marks<80){

System.out.println("B grade");

    }

    else if(marks>=80 && marks<90){
```

```
System.out.println("A grade");

}else if(marks>=90 && marks<100){

System.out.println("A+ grade");

}else{

System.out.println("Invalid!");

    }

}

}
```

Copy



```
D:\Studytonight>Javac IfElseIfDemo1.java

D:\Studytonight>Java IfElseIfDemo1
B grade

D:\Studytonight>_
```

Nested if statement

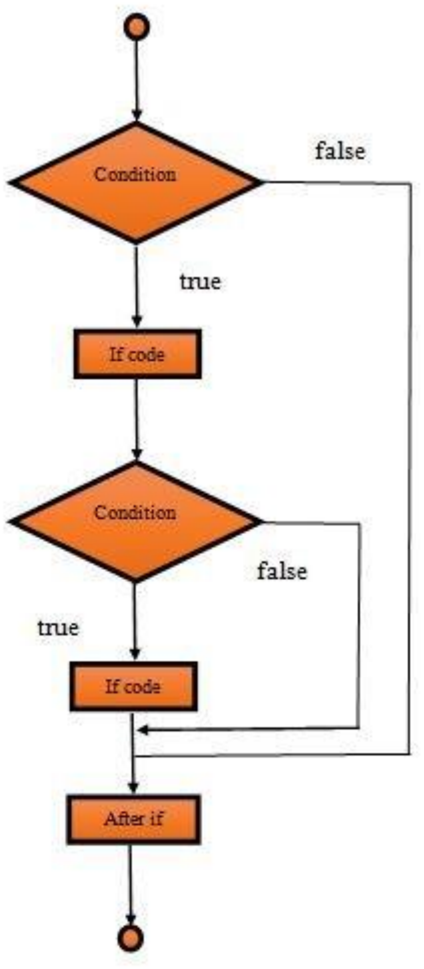
In Java, the Nested if statement is a if inside another if. In this, one if block is created inside another if block when the outer block is true then only the inner block is executed.

Syntax:

```
if(condition)

{

          //statement

     if(condition)

{

             //statement

    }

}
```

Copy

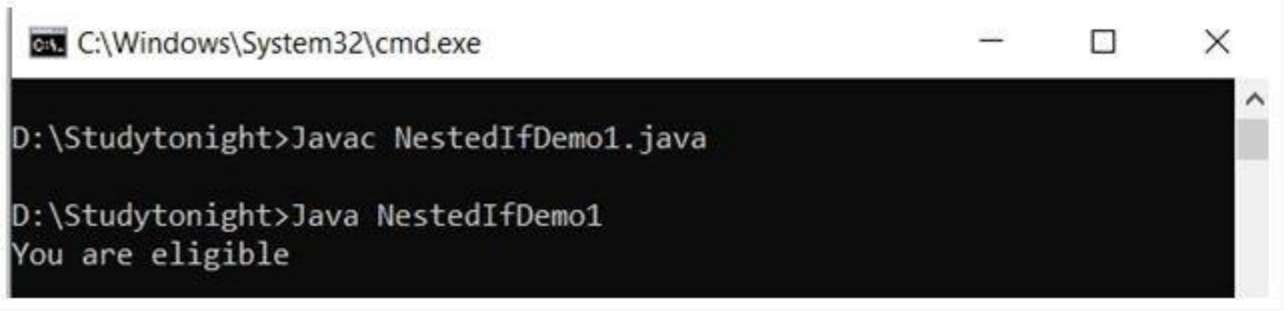
Data-flow-diagram of Nested If Block



Example:

```
public class NestedIfDemo1 {

public static void main(String[] args)

{

int age=25;

int weight=70;

if(age>=18)

{

if(weight>50)

{

System.out.println("You are eligible");

}

}

}

}
```

OUTPUT



```
D:\Studytonight>Javac NestedIfDemo1.java

D:\Studytonight>Java NestedIfDemo1
You are eligible
```

Java Switch Statement

In Java, the switch statement is used for executing one statement from multiple conditions. it is similar to an if-else-if ladder.

Switch statement consists of conditional based cases and a default case.

In a switch statement, the expression can be of **byte, short, char and int** type.

From **JDK-7, enum, String** can also be used in switch cases.

Following are some of the rules while using the switch statement:

1. There can be one or N numbers of cases.
2. The values in the case must be unique.
3. Each statement of the case can have a break statement. It is optional.

Syntax:

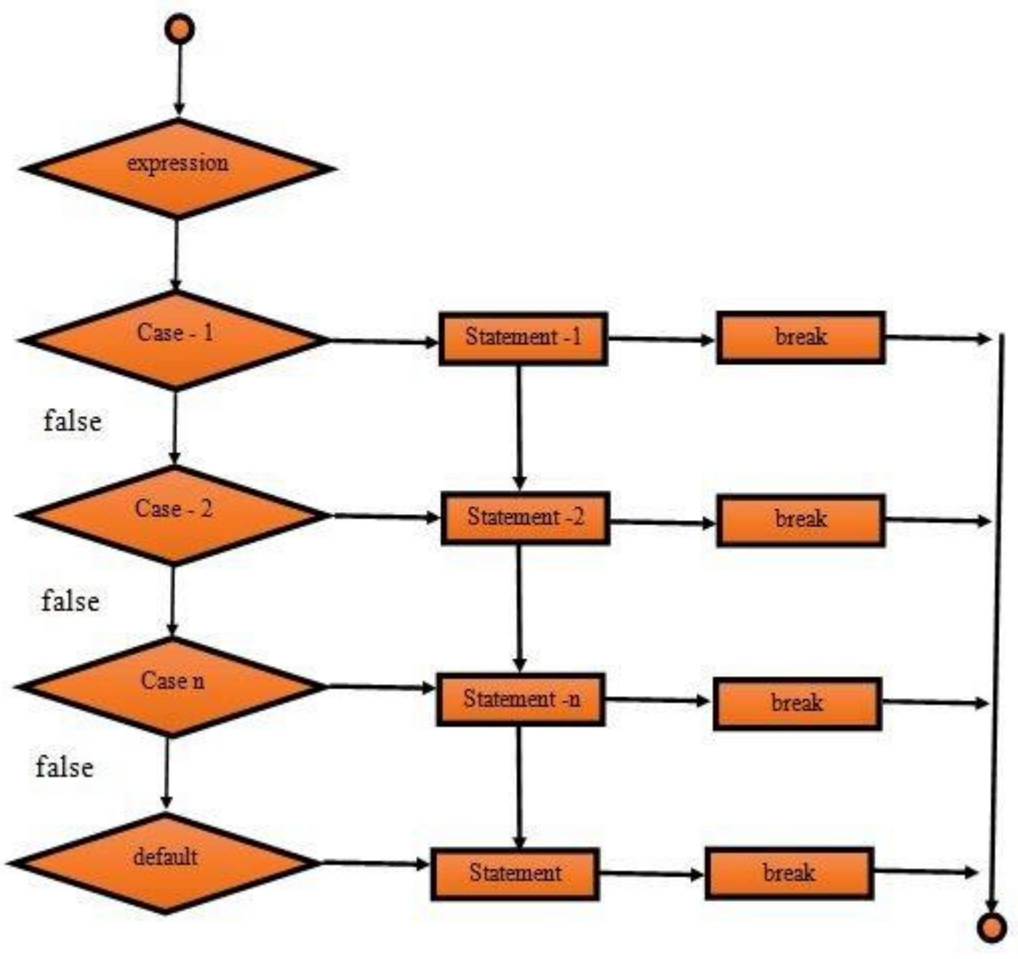
Following is the syntax to declare the switch case in Java.

```
switch(expression)

{

case value1:

               //code for execution;

               break;  //optional

case value2:

 // code for execution

 break;  //optional

......

......

......

......

Case value n:

// code for execution
```

```
 break;  //optional


default:

 code for execution when none of the case is true;

}
```

Copy

Data Flow Diagram Of switch Block

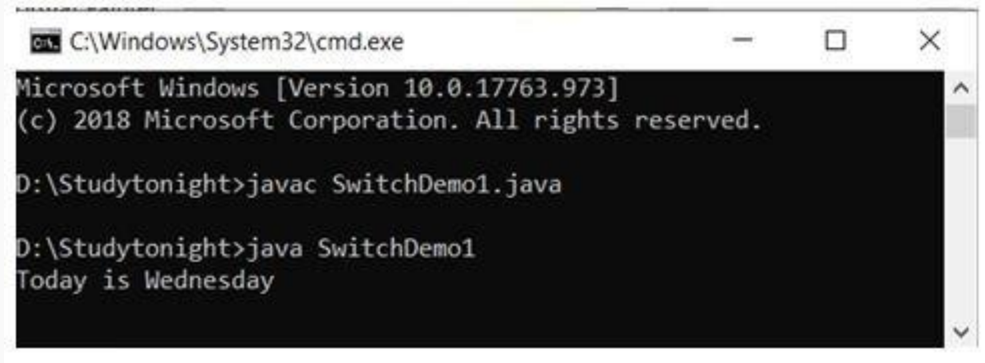


Example: Using integer value

In this example, we are using **int type** value to match cases. This example returns day based on the numeric value.

```

public class SwitchDemo1{

    public static void main(String[] args)

    {

int day = 3;

        String dayName;

        switch (day) {

        case 1:

dayName = "Today is Monday";

            break;
        case 2:

dayName = "Today is Tuesday";

            break;

        case 3:

dayName = "Today is Wednesday";

            break;

        case 4:

dayName = "Today is Thursday";

            break;

        case 5:

dayName = "Today is Friday";

            break;

        case 6:

dayName = "Today is Saturday";

            break;

        case 7:

dayName = "Today is Sunday";

            break;

        default:

dayName = "Invalid day";

            break;

        }

System.out.println(dayName);

    }

}
```

Copy



```
C:\Windows\System32\cmd.exe
Microsoft Windows [Version 10.0.17763.973]
(c) 2018 Microsoft Corporation. All rights reserved.

D:\Studytonight>javac SwitchDemo1.java

D:\Studytonight>java SwitchDemo1
Today is Wednesday
```

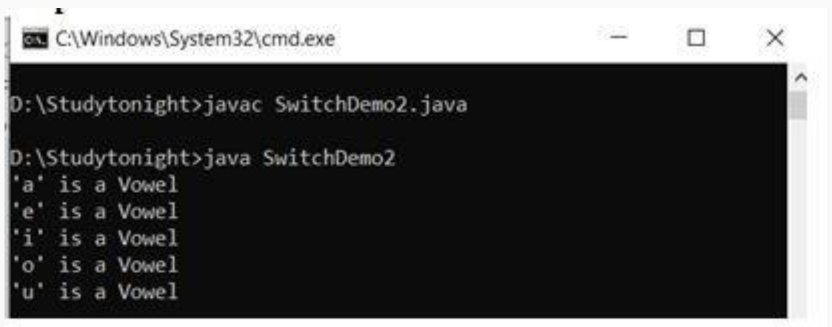
Example using Enum in Switch statement

As we have said, Java allows to use **enum** in switch cases. So we are creating an enum of vowel alphabets and using its elements in switch case.

```
public class SwitchDemo2{

       public enumvowel{a, e, i, o, u}

       public static void main(String args[])

       {

vowel[] character= vowel.values();

           for (vowel Now : character)

           {

                switch (Now)

                {

                    case a:

System.out.println("'a' is a Vowel");

                        break;

                    case e:

System.out.println("'e' is a Vowel");

                        break;

                    case i:

System.out.println("'i' is a Vowel");

                        break;

                    case o:

System.out.println("'o' is a Vowel");

                        break;

                    case u:

System.out.println("'u' is a Vowel");
```

```
                        break;

                    default:

System.out.println("It is a consonant");

                }

            }

        }

}
```

Copy



```
C:\Windows\System32\cmd.exe

D:\Studytonight>javac SwitchDemo2.java

D:\Studytonight>java SwitchDemo2
'a' is a Vowel
'e' is a Vowel
'i' is a Vowel
'o' is a Vowel
'u' is a Vowel
```

Example: String in switch case

Since Java has allowed to use string values in switch cases, so we are using string to create a string based switch case example.

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

    String name = "Mango";

    switch(name){

    case "Mango":

        System.out.println("It is a fruit");

        break;

    case "Tomato":

        System.out.println("It is a vegitable");

        break;

    case "Coke":

        System.out.println("It is cold drink");

    }

}
```

```
}
```

Copy

```
It is a fruit
```

Example: without break switch case

Break statement is used to **break the current execution** of the program. In switch case, break is used to **terminate the switch case** execution and transfer control to the outside of switch case.

Use of **break is optional in the switch case**. So lets see what happens if we don't use the break.

```
public class Demo{

    public static void main(String[] args) {
        String name = "Mango";
        switch(name){
        case "Mango":
            System.out.println("It is a fruit");
        case "Tomato":
            System.out.println("It is a vegitable");
        case "Coke":
            System.out.println("It is cold drink");
        }
    }
}
```

Copy

```
It is a fruit
It is a vegitable
It is cold drink
```

See, if we don't use break, it executes all the cases after matching case.

Java Loops

Loop is an important concept of a programming that allows to iterate over the sequence of statements.

Loop is designed to execute particular code block till the specified condition is true or all the elements of a collection(array, list etc) are completely traversed.

The most common use of loop is to perform repetitive tasks. For example if we want to print table of a number then we need to write print statement 10 times. However, we can do the same with a single print statement by using loop.

Loop is designed to execute its block till the specified condition is true.

Java provides mainly three loop based on the loop structure.

1. for loop
2. while loop
3. do while loop

We will explain each loop individually in details in the below.

For Loop

The for loop is used for executing a part of the program **repeatedly**. When the number of execution is fixed then it is suggested to use for loop. For loop can be categories into two type.

1. for loop
2. for-each loop

For Loop Syntax:

Following is the syntax for declaring for loop in Java.

```
for(initialization;condition;increment/decrement)

{

//statement

}
```

Copy

For loop Parameters:

To create a for loop, we need to set the following parameters.

1) Initialization

It is the initial part, where we set initial value for the loop. It is executed only once at the starting of loop. It is optional, if we don't want to set initial value.

2) Condition

It is used to test a condition each time while executing. The execution continues until the condition is false. It is optional and if we don't specify, loop will be inifinite.

3) Statement

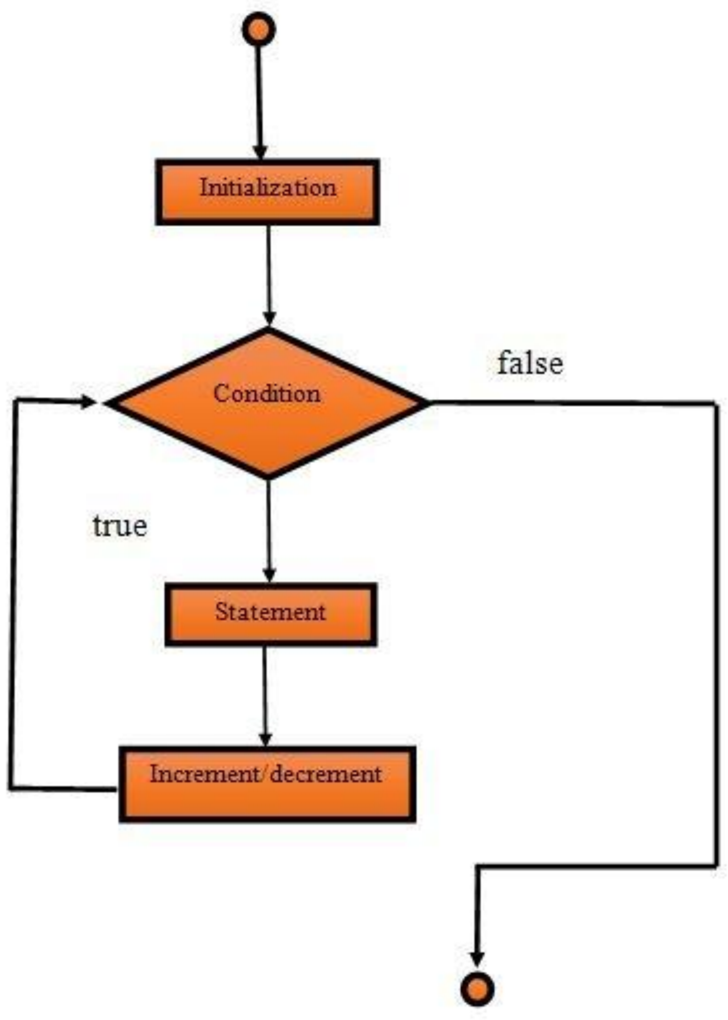
It is loop body and executed every time until the condition is false.

4) Increment/Decrement

It is used for set increment or decrement value for the loop.

Data Flow Diagram of for loop

This flow diagram shows flow of the loop. Here we can understand flow of the loop.

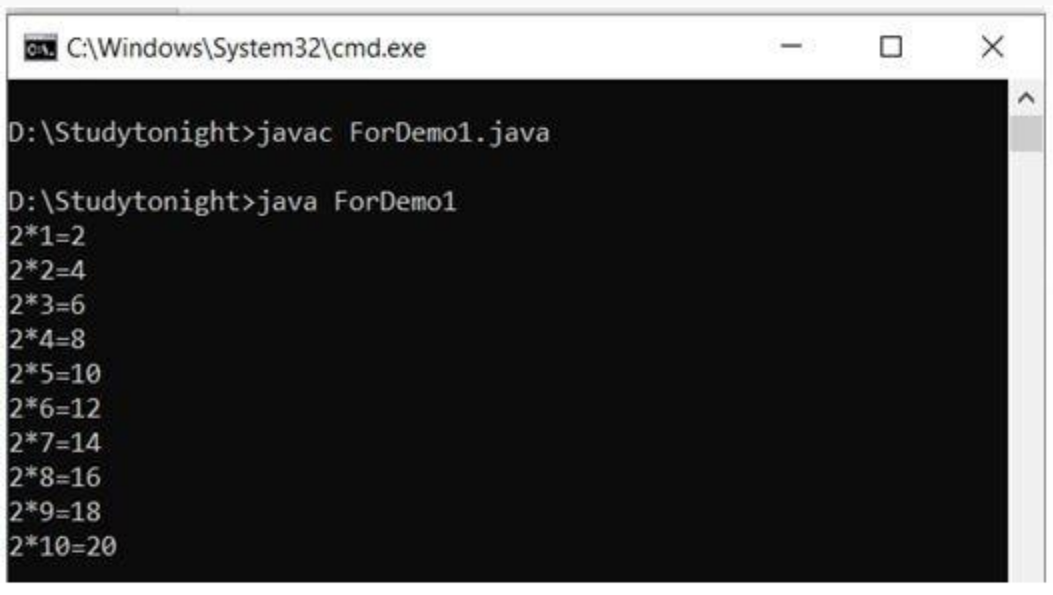


For loop Example

In this example, initial value of loop is set to 1 and incrementing it by 1 till the condition is true and executes 10 times.

```
public class ForDemo1

{

public static void main(String[] args)

     {

          int n, i;

          n=2;

          for(i=1;i<=10;i++)

          {

               System.out.println(n+"*"+i+"="+n*i);

          }

}
}
```

Copy



```
C:\Windows\System32\cmd.exe

D:\Studytonight>javac ForDemo1.java

D:\Studytonight>java ForDemo1
2*1=2
2*2=4
2*3=6
2*4=8
2*5=10
2*6=12
2*7=14
2*8=16
2*9=18
2*10=20
```

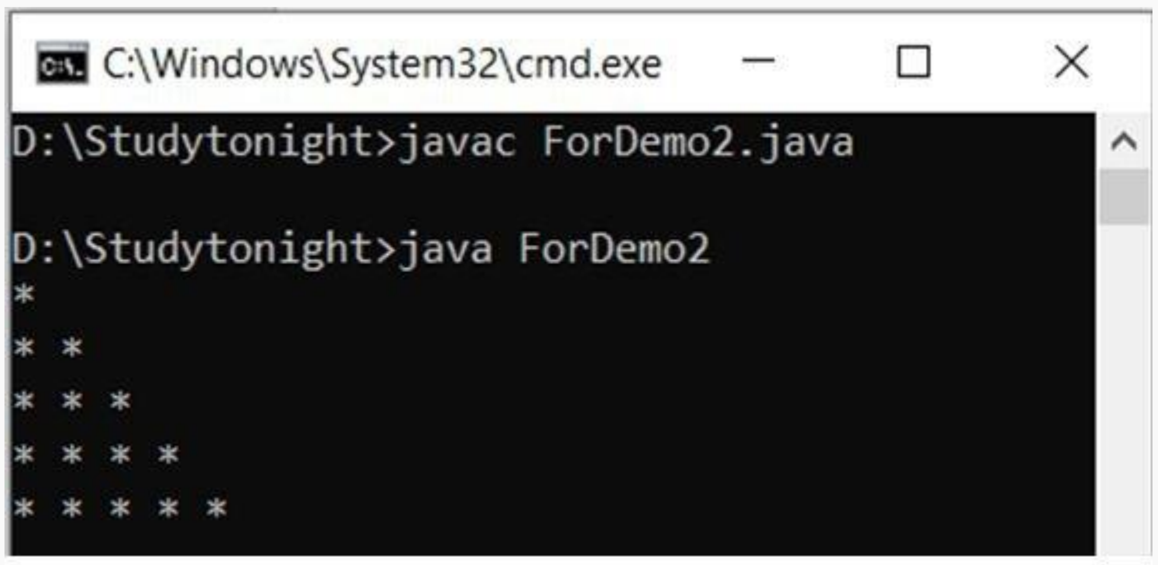
Example for Nested for loop

Loop can be nested, loop created **inside another loop** is called nested loop. Sometimes based on the requirement, we have to create nested loop.

Generally, nested loops are used to **iterate tabular data**.

```
public class ForDemo2

{

public static void main(String[] args)

{

for(inti=1;i<=5;i++)

{

for(int j=1;j<=i;j++)

{

                    System.out.print("* ");

}

System.out.println();

}

}

}
```

Copy



```
C:\Windows\System32\cmd.exe
D:\Studytonight>javac ForDemo2.java

D:\Studytonight>java ForDemo2
*
* *
* * *
* * * *
* * * * *
```

for-each Loop

In Java, for each loop is used for traversing array or collection elements. In this loop, there is no need for increment or decrement operator.

For-each loop syntax

Following is the syntax to declare for-each loop in the Java.

```
for(Type var:array)

{

//code for execution

}
```

Copy

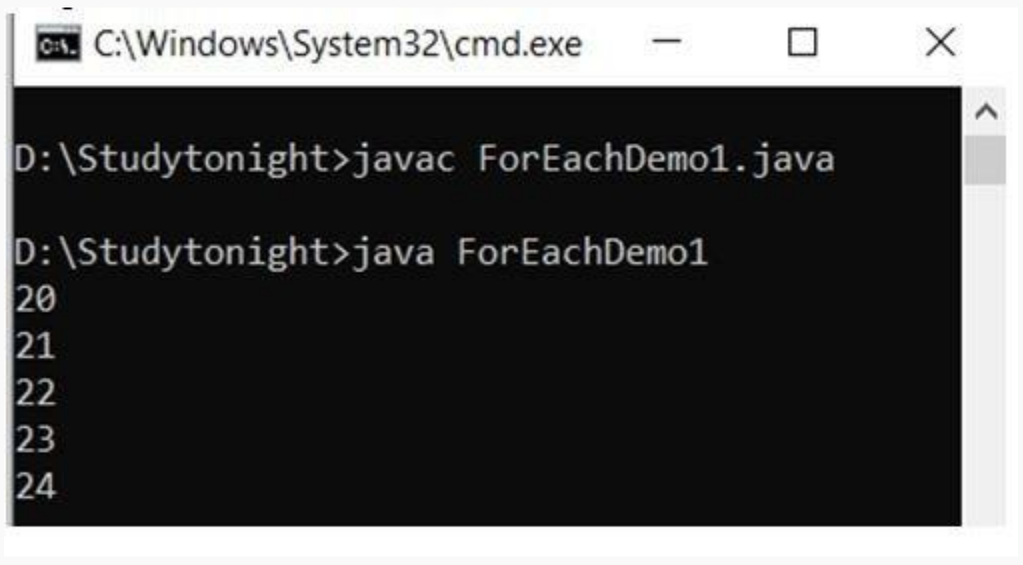
Example:

In this example, we are traversing array elements using the for-each loop. For-each **loop terminates automatically when no element is left in the array object**.

```
public class ForEachDemo1

{

public static void main(String[] args)

          {

               inta[]={20,21,22,23,24};

               for(int i:a)

{
```

```
                        System.out.println(i);

                  }

}

}
```

Copy



```
C:\Windows\System32\cmd.exe

D:\Studytonight>javac ForEachDemo1.java

D:\Studytonight>java ForEachDemo1
20
21
22
23
24
```

While Loop

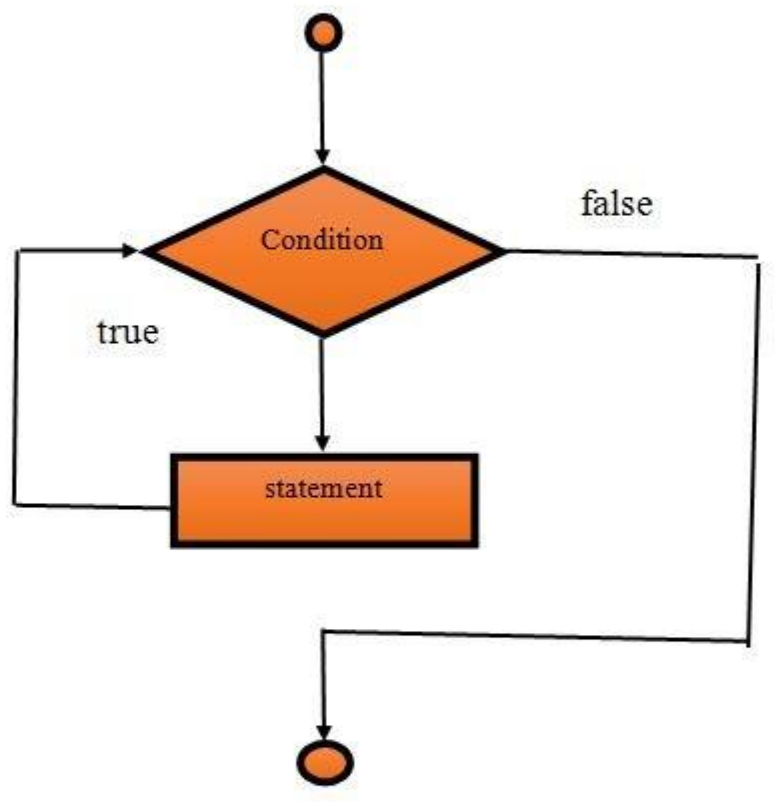
Like for loop, while loop is also used to execute code repeatedly. a control statement. It is used for iterating a part of the program several times. When the number of iteration is not fixed then while loop is used.

Syntax:

```
while(condition)

{

//code for execution

}
```

Copy

Data-flow-diagram of While Block

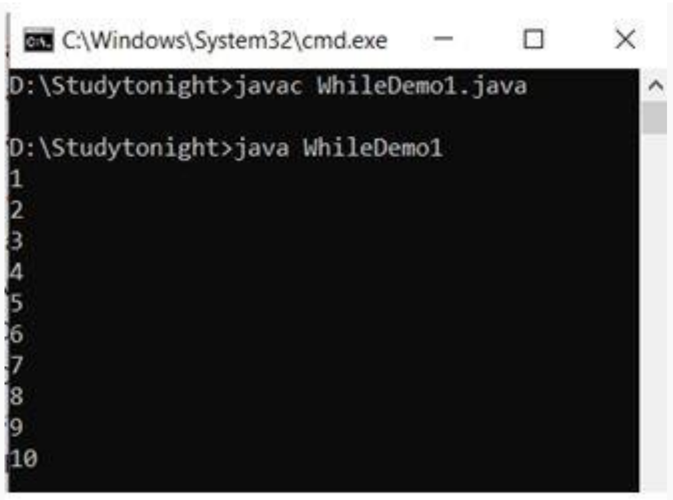


Example:

In this example, we are using while loop to print 1 to 10 values. In first step, we set conditional variable then test the condition and if condition is true execute the loop body and increment the variable by 1.

```
public class WhileDemo1
{
     public static void main(String[] args)
     {
          inti=1;
          while(i<=10)
          {
               System.out.println(i);
               i++;
          }
     }
}
```

Copy



```
C:\Windows\System32\cmd.exe
D:\Studytonight>javac WhileDemo1.java

D:\Studytonight>java WhileDemo1
1
2
3
4
5
6
7
8
9
10
```

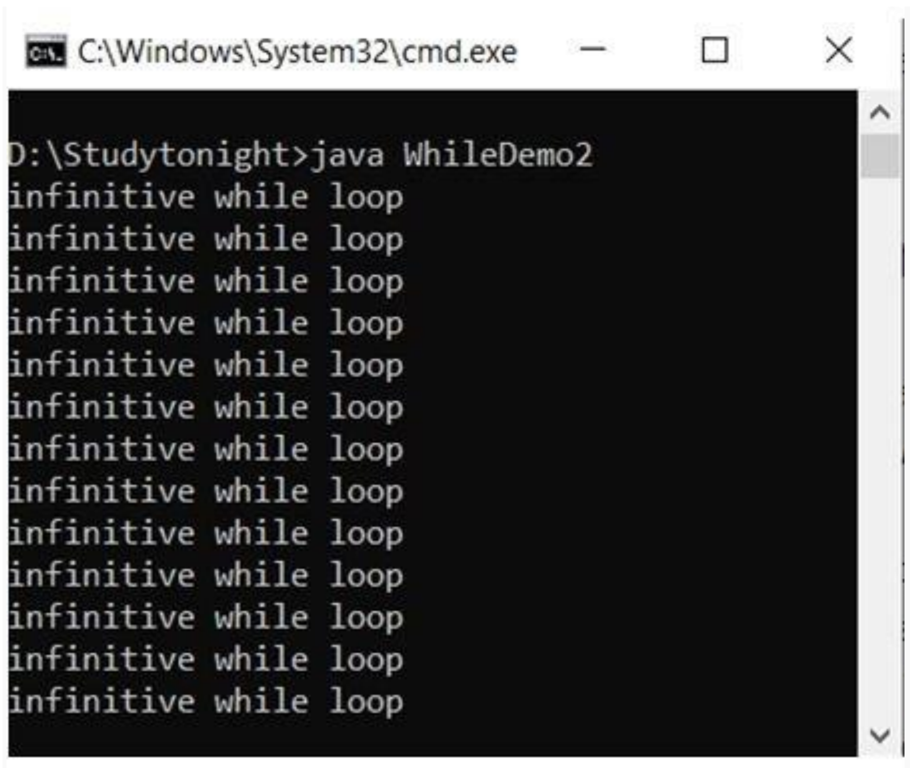
Example for infinite while loop

A while loop which conditional expression **always returns true** is called infinite while loop. We can also create infinite loop by passing **true** literal in the loop.

Be careful, when creating infinite loop because it can issue memory overflow problem.

```
public class WhileDemo2
{
    public static void main(String[] args)
    {
        while(true)
        {
        System.out.println("infinitive while loop");
        }
    }
}
```

Copy



```
D:\Studytonight>java WhileDemo2
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
infinitive while loop
```

do-while loop

In Java, the do-while loop is used to execute statements again and again. This loop **executes at least once** because the loop is executed before the condition is checked. It means loop **condition evaluates after executing** of loop body.

The main **difference between while and do-while loop** is, in do while loop condition evaluates after executing the loop.

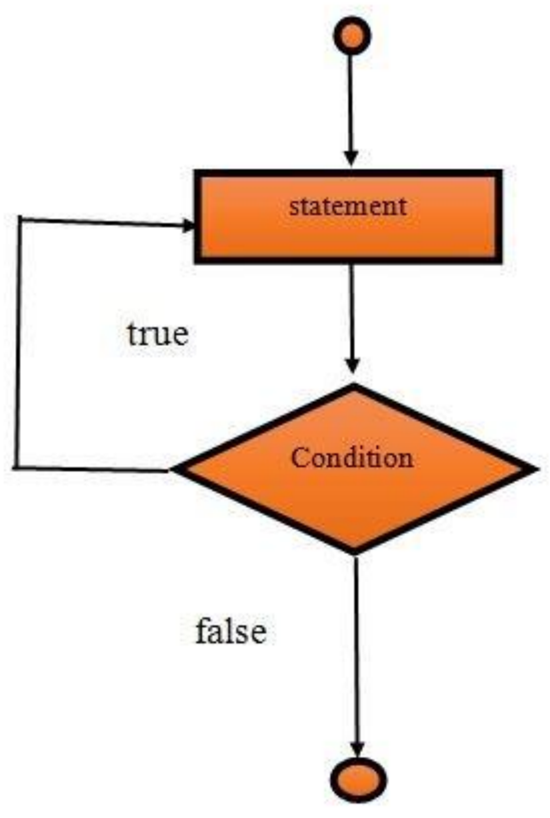
Syntax:

Following is the syntax to declare do-while loop in Java.

```
do

{

//code for execution

}

while(condition);
```

Copy

Data Flow Diagram of do-while Block

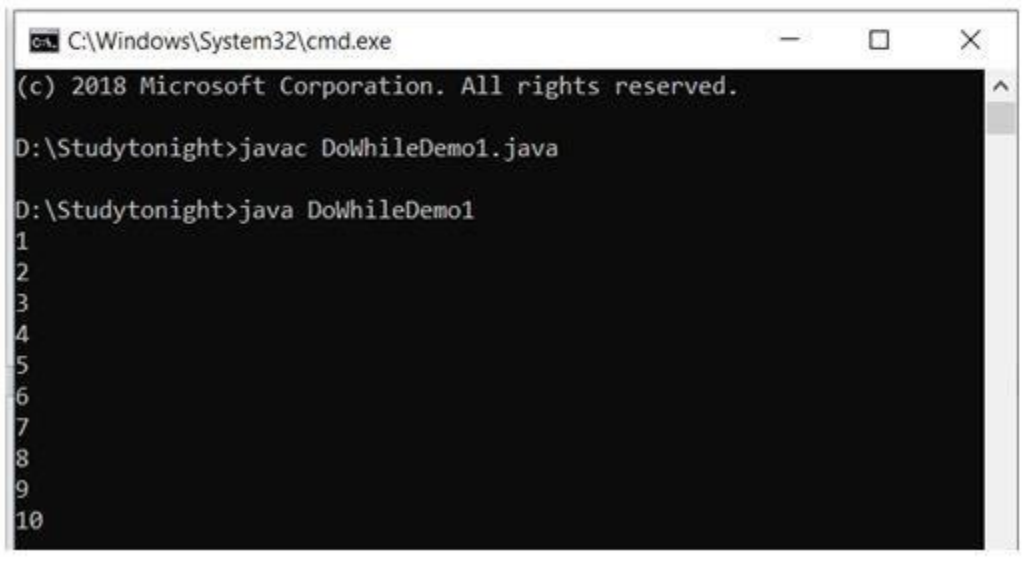


Example:

In this example, we are printing values from 1 to 10 by using the do while loop.

```
public class DoWhileDemo1

{

public static void main(String[] args)

{

inti=1;

do

{

System.out.println(i);

i++;

}while(i<=10);

        }

}
```

Copy



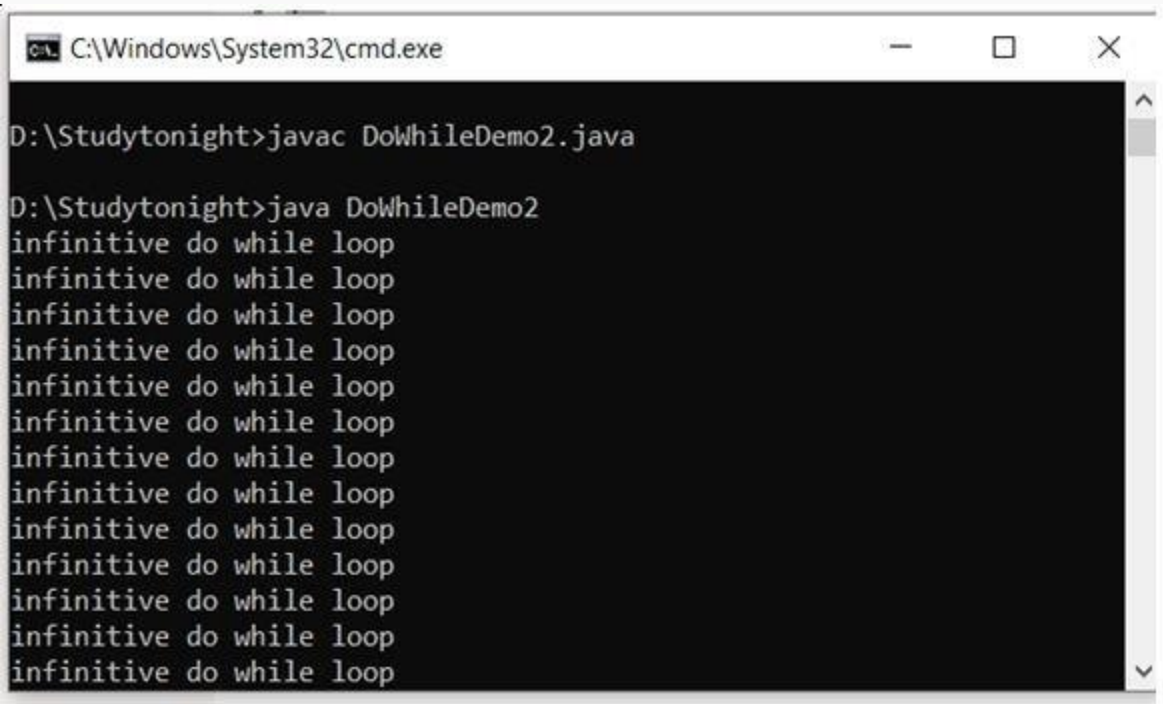
```
C:\Windows\System32\cmd.exe
(c) 2018 Microsoft Corporation. All rights reserved.

D:\Studytonight>javac DoWhileDemo1.java

D:\Studytonight>java DoWhileDemo1
1
2
3
4
5
6
7
8
9
10
```

Example for infinite do-while loop

Like infinite while loop, we can create infinite do while loop as well. To create an infinite do while loop just **pass the condition that always remains true.**

```
public class DoWhileDemo2

{

public static void main(String[] args)

{

        do

{

System.out.println("infinitive do while loop");

}while(true);

}

}
```

Copy



```
C:\Windows\System32\cmd.exe

D:\Studytonight>javac DoWhileDemo2.java

D:\Studytonight>java DoWhileDemo2
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
infinitive do while loop
```

Java break and continue Statements

Java break and continue statements are used to manage program flow. We can use them in a loop to control loop iterations. These statements let us to control loop and switch statements by enabling us to either break out of the loop or jump to the next iteration by skipping the current loop iteration.

In this tutorial, we will discuss each in details with examples.

Break Statement

In Java, break is a statement that is used to **break current execution** flow of the program.

We can use break statement inside loop, switch case etc.

If break is used **inside loop** then it will terminate the loop.

If break is used **inside the innermost loop** then break will terminate the innermost loop only and execution will start from the outer loop.

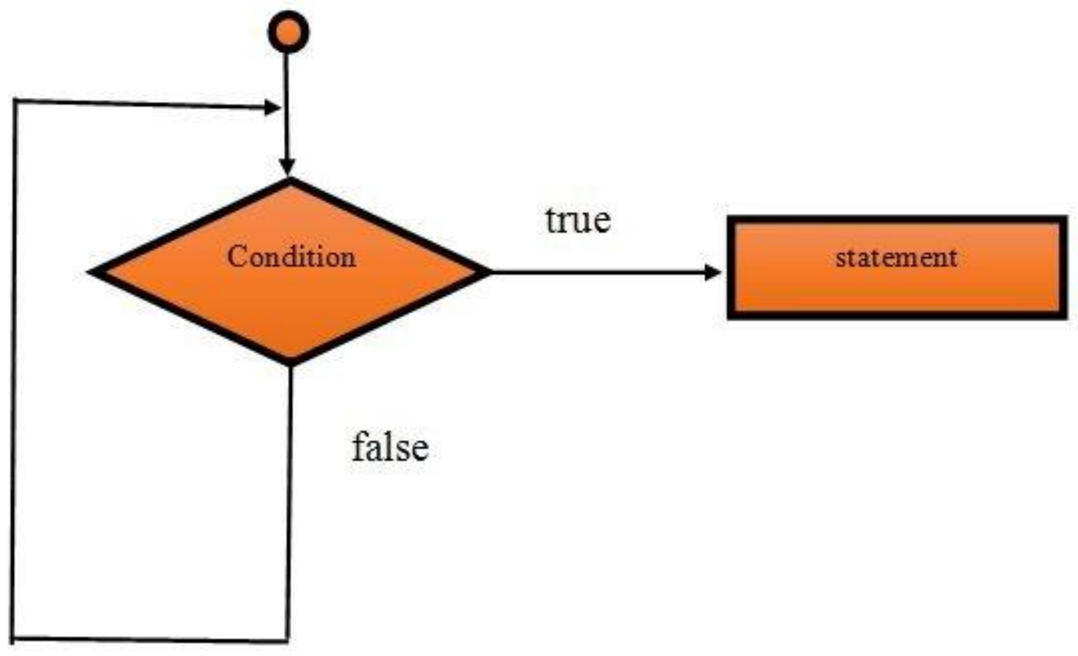
If break is used in switch case then it will terminate the execution after the matched case. Use of break, we have covered in our switch case topic.

Syntax:

```
jump-statement;

break;
```

Copy

Data Flow Diagram of break statement

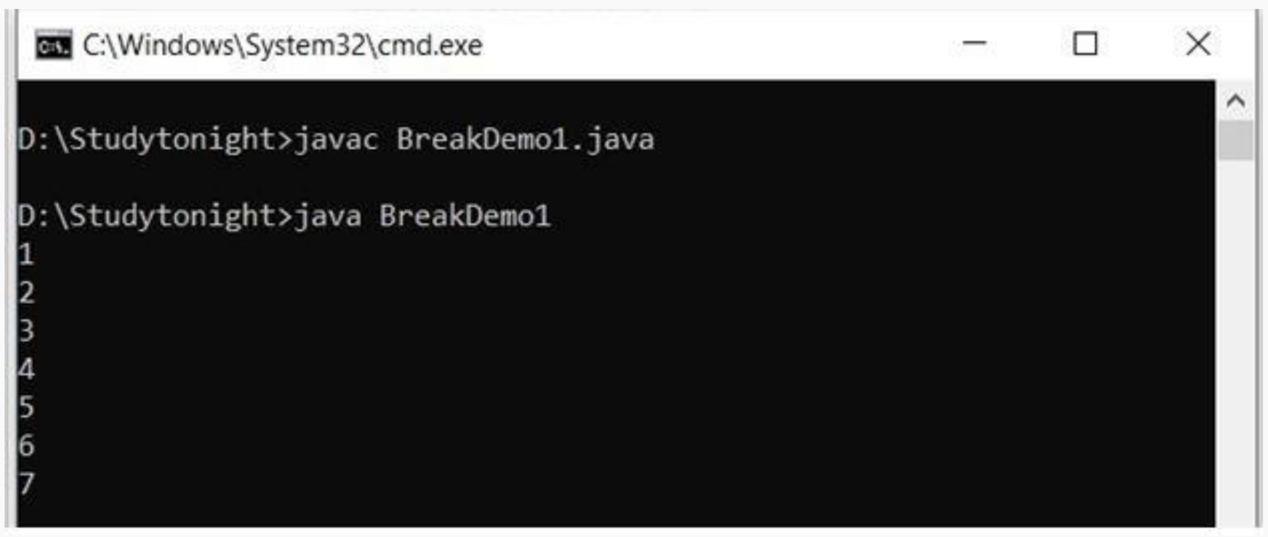


Example:

In this example, we are using break inside the loop, the loop will terminate when the value is 8.

```
public class BreakDemo1 {

public static void main(String[] args) {


for(inti=1;i<=10;i++){

        if(i==8){


            break;

        }

System.out.println(i);

    }

}

}
```

Copy



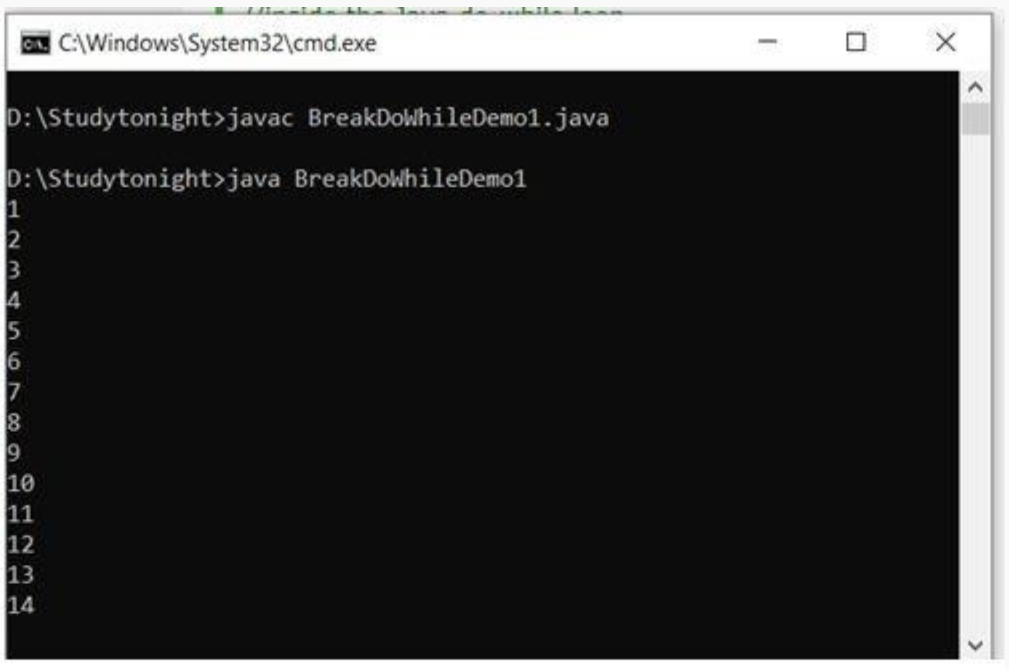
```
D:\Studytonight>javac BreakDemo1.java

D:\Studytonight>java BreakDemo1
1
2
3
4
5
6
7
```

Example using break in do while loop

Loop can be any one whether it is for or while, break statement will do the same. Here, we are using break inside the do while loop.

```
public class BreakDoWhileDemo1

{

public static void main(String[] args)

{

          inti=1;

          do
{
                  if(i==15)
{
                  i++;
                        break;
                  }
                  System.out.println(i);
                  i++;
            }while(i<=20);
 }
}
```

Copy



```
C:\Windows\System32\cmd.exe

D:\Studytonight>javac BreakDoWhileDemo1.java

D:\Studytonight>java BreakDoWhileDemo1
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

Example: Break in innermost loop

In this example, we are using **break inside the innermost loop**. But the loop breaks each time when j is equal to 2 and **control goes to outer loop** that starts from the next iteration.

```
public class Demo{
     public static void main(String[] args) {
      for(int i=1;i<=2;i++){
              for (int j = 0; j <=3; j++) {
                        if(j==2)
                              break;
                        System.out.println(j);

                  }

         }

     }

}
```

Copy

```
0

1

0

1
```

continue Statement

In Java, the continue statement is used to **skip the current iteration of the loop**. It **jumps to the next iteration** of the loop immediately. We can use continue statement with **for loop**, **while loop** and **do-while loop** as well.

```
jump-statement;

continue;
```

Copy

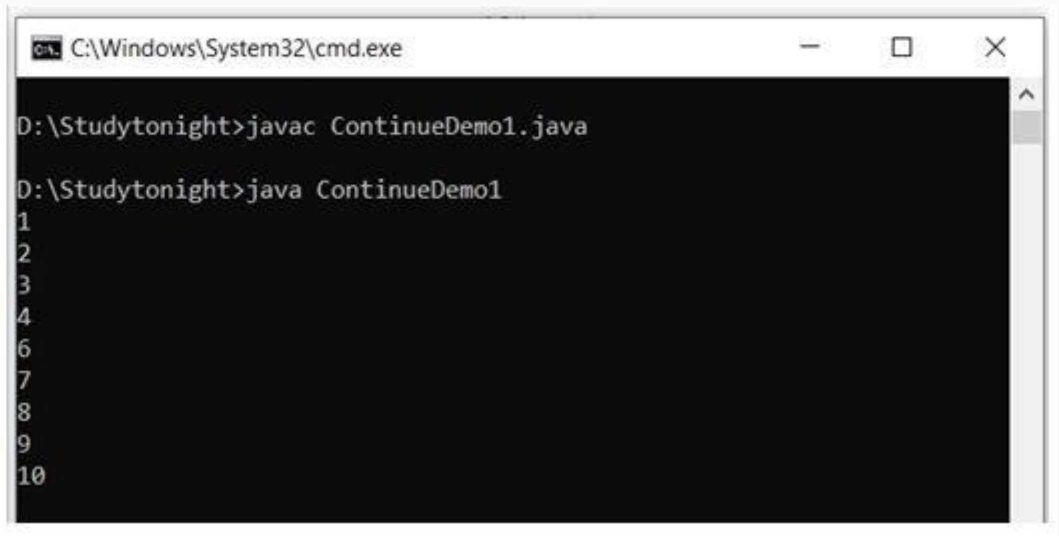
Example:

In this example, we are using continue statement inside the for loop. See, it does not print 5 to the console because at fifth iteration continue statement skips the iteration that’s why print statement does not execute.

```
public class ContinueDemo1

{

public static void main(String[] args)

{

            for(inti=1;i<=10;i++)

            {

                  if(i==5)

{
```

```
                    continue;

                    }

                    System.out.println(i);

            }

}

}
```

Copy



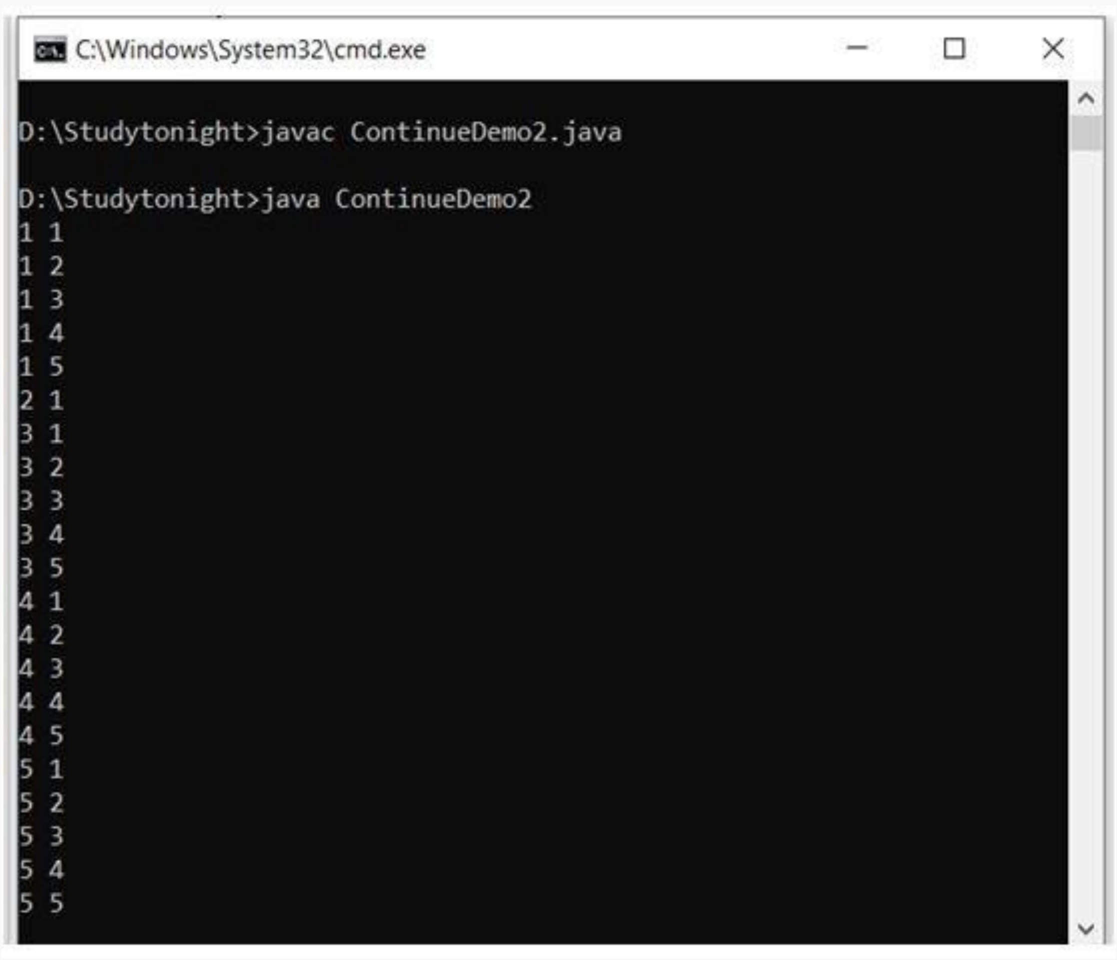
```
C:\Windows\System32\cmd.exe

D:\Studytonight>javac ContinueDemo1.java

D:\Studytonight>java ContinueDemo1
1
2
3
4
6
7
8
9
10
```

Example:

We can use **label along with continue statement** to **set flow control**. By using label, we can **transfer control at specified location.**

In this example, we are transferring control to outer loop by using label.

```
public class ContinueDemo2 {

public static void main(String[] args) {

xy:

for(inti=1;i<=5;i++){

pq:

for(int j=1;j<=5;j++){

                              if(i==2&&j==2){

                                   continue xy;

                              }

System.out.println(i+" "+j);

                         }

               }
}

}
```
Copy



```
C:\Windows\System32\cmd.exe

D:\Studytonight>javac ContinueDemo2.java

D:\Studytonight>java ContinueDemo2
1 1
1 2
1 3
1 4
1 5
2 1
3 1
3 2
3 3
3 4
3 5
4 1
4 2
4 3
4 4
4 5
5 1
5 2
5 3
5 4
5 5
```

Example: Continue in While loop

continue statement can be used with while loop to manage the flow control of the program. As we already know continue statement is used to skip the current iteration of the loop. Here too, it will skip the execution if the value of variable is 5.

```
public class Demo{


    public static void main(String[] args) {

     int i=1;


     while (i < 10) {

       if (i == 5) {

         i++;

         continue;

       }

       System.out.println(i);

       i++;
```

```
        }

      }

}
```

OUTPUT

```
1
2
3
4
6
7
8
9
```

we can see the output, 5 is missing because at fifth iteration due to continue statement JVM skipped the print statement.

Java Operators

Operator is a symbol which tells to the compiler to perform some operation. Java provides a rich set of operators do deal with various types of operations. Sometimes we need to perform arithmetic operations then we use plus (+) operator for addition, multiply(*) for multiplication etc.

Operators are always essential part of any programming language.

Java operators can be divided into following categories:

- Arithmetic operators
- Relation operators
- Logical operators
- Bitwise operators
- Assignment operators
- Conditional operators
- Misc operators

Arithmetic operators

Arithmetic operators are used to perform arithmetic operations like: addition, subtraction etc and helpful to solve mathematical expressions. The below table contains Arithmetic operators.

Operator	Description

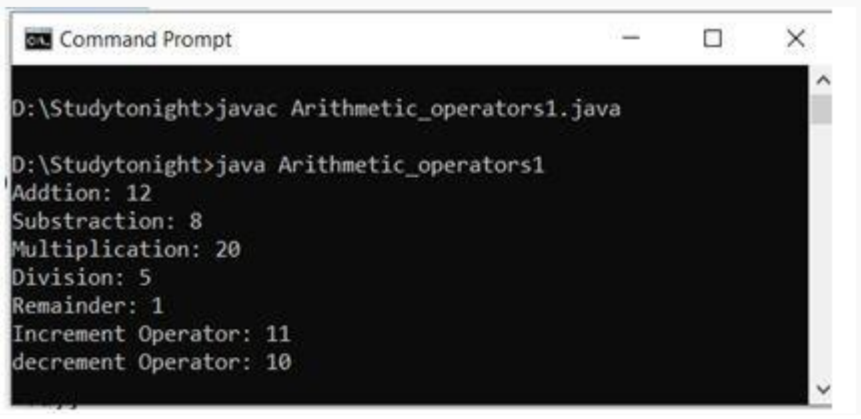
+	adds two operands
-	subtract second operands from first
*	multiply two operand
/	divide numerator by denumerator
%	remainder of division
++	Increment operator increases integer value by one
--	Decrement operator decreases integer value by one

Example:

Lets create an example to understand arithmetic operators and their operations.

```
class Arithmetic_operators1{
public static void main(String as[])
    {
        int a, b, c;
        a=10;
        b=2;
        c=a+b;
        System.out.println("Addtion: "+c);
        c=a-b;
        System.out.println("Substraction: "+c);
        c=a*b;
        System.out.println("Multiplication: "+c);
        c=a/b;
        System.out.println("Division: "+c);
        b=3;
        c=a%b;

        System.out.println("Remainder:  "+c);

        a=++a;

        System.out.println("Increment Operator:  "+a);

        a=--a;

        System.out.println("decrement Operator:  "+a);



    }

}
```

Copy



```
D:\Studytonight>javac Arithmetic_operators1.java

D:\Studytonight>java Arithmetic_operators1
Addtion: 12
Substraction: 8
Multiplication: 20
Division: 5
Remainder: 1
Increment Operator: 11
decrement Operator: 10
```

Relation operators

Relational operators are used to test comparison between operands or values. It can be use to test whether two values are equal or not equal or less than or greater than etc.

The following table shows all relation operators supported by Java.

Operator	Description
==	Check if two operand are equal
!=	Check if two operand are not equal.
>	Check if operand on the left is greater than operand on the right
<	Check operand on the left is smaller than right operand

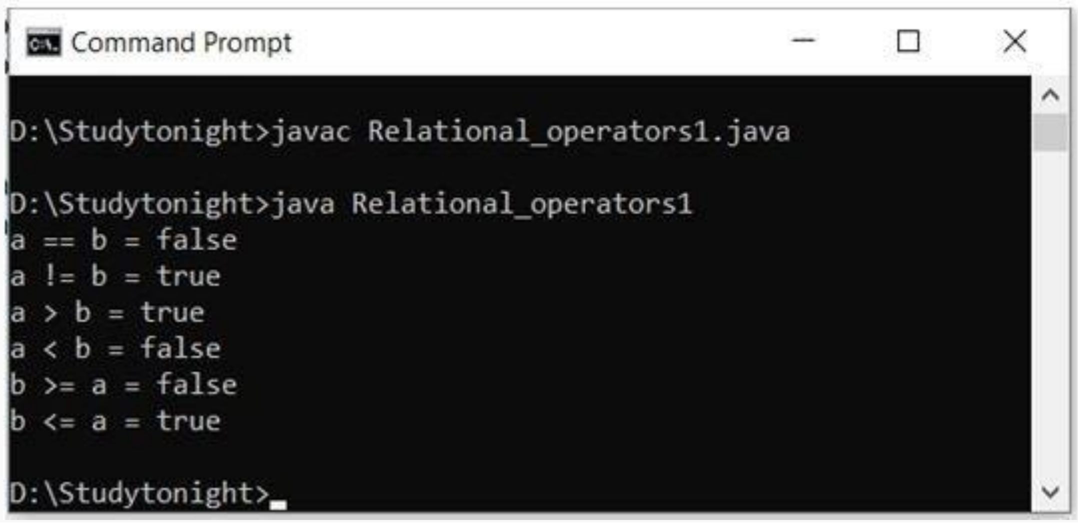
>=	check left operand is greater than or equal to right operand
<=	Check if operand on left is smaller than or equal to right operand

Example:

In this example, we are using relational operators to test comparison like less than, greater than etc.

```
class Relational_operators1{
public static void main(String as[])
    {
        int a, b;
        a=40;
        b=30;
        System.out.println("a == b = " + (a == b) );
        System.out.println("a != b = " + (a != b) );
        System.out.println("a > b = " + (a > b) );
        System.out.println("a < b = " + (a < b) );
        System.out.println("b >= a = " + (b >= a) );
        System.out.println("b <= a = " + (b <= a) );
    }
}
```

Copy



```
D:\Studytonight>javac Relational_operators1.java

D:\Studytonight>java Relational_operators1
a == b = false
a != b = true
a > b = true
a < b = false
b >= a = false
b <= a = true

D:\Studytonight>_
```

Logical operators

Logical Operators are used to check conditional expression. For example, we can use logical operators in if statement to evaluate conditional based expression. We can use them into loop as well to evaluate a condition.

Java supports following 3 logical operator. Suppose we have two variables whose values are: **a=true** and **b=false**.

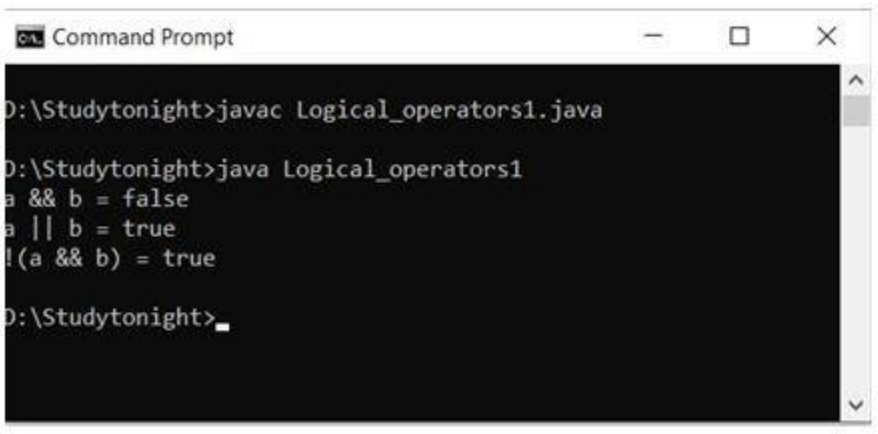
Operator	Description	Example
&&	Logical AND	(a && b) is false
||	Logical OR	(a || b) is true
!	Logical NOT	(!a) is false

Example:

In this example, we are using logical operators. There operators return either true or false value.

```
class Logical_operators1{
public static void main(String as[])
    {
        boolean a = true;
        boolean b = false;
        System.out.println("a && b = " + (a&&b));
        System.out.println("a || b = " + (a||b) );
System.out.println("!(a && b) = " + !(a && b));
    }
}
```

Copy



```
D:\Studytonight>javac Logical_operators1.java

D:\Studytonight>java Logical_operators1
a && b = false
a || b = true
!(a && b) = true

D:\Studytonight>_
```

Bitwise operators

Bitwise operators are used to perform operations bit by bit.

Java defines several bitwise operators that can be applied to the integer types long, int, short, char and byte.

The following table shows all bitwise operators supported by Java.

Operator	Description
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
<<	left shift
>>	right shift

Now lets see truth table for bitwise &, | and ^

a	b	a & b	a | b	a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

The bitwise **shift operators shifts the bit value**. The **left operand specifies the value to be shifted** and the **right operand specifies the number of positions** that the bits in the value are to be shifted. Both operands have the same precedence.

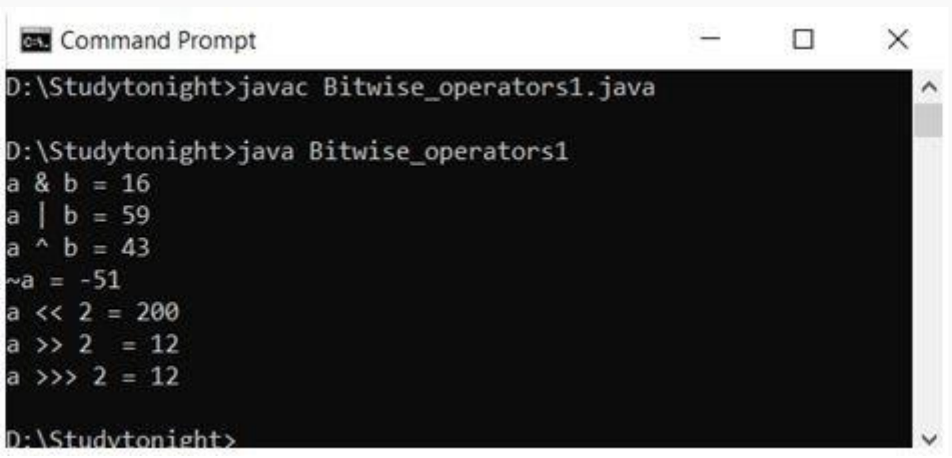
Example:
Lets create an example that shows working of bitwise operators.

```
a = 0001000

b = 2

a << b = 0100000

a >> b = 0000010
```

Copy

```


class Bitwise_operators1{

public static void main(String as[])

    {

        int a = 50;

        int b = 25;

        int c = 0;



        c = a & b;

        System.out.println("a & b = " + c );



        c = a | b;

        System.out.println("a | b = " + c );



        c = a ^ b;

        System.out.println("a ^ b = " + c );



        c = ~a;

        System.out.println("~a = " + c );



        c = a << 2;
        System.out.println("a << 2 = " + c );


        c = a >> 2;

        System.out.println("a >>2  = " + c );


        c = a >>> 2;

        System.out.println("a >>> 2 = " + c );

    }

}
```

Copy



```
D:\Studytonight>javac Bitwise_operators1.java

D:\Studytonight>java Bitwise_operators1
a & b = 16
a | b = 59
a ^ b = 43
~a = -51
a << 2 = 200
a >> 2  = 12
a >>> 2 = 12

D:\Studytonight>
```

Assignment Operators

Assignment operators are used to assign a value to a variable. It can also be used combine with arithmetic operators to perform arithmetic operations and then assign the result to the variable. Assignment operator supported by Java are as follows:

Operator	Description	Example
`=`	assigns values from right side operands to left side operand	`a = b`
`+=`	adds right operand to the left operand and assign the result to left	`a+=b` is same as `a=a+b`
`-=`	subtracts right operand from the left operand and assign the result to left operand	`a-=b` is same as a`=a-b`

*=	mutiply left operand with the right operand and assign the result to left operand	a*=b is same as a=a*b
/=	divides left operand with the right operand and assign the result to left operand	a/=b is same as a=a/b
%=	calculate modulus using two operands and assign the result to left operand	a%=b is same as a=a%b

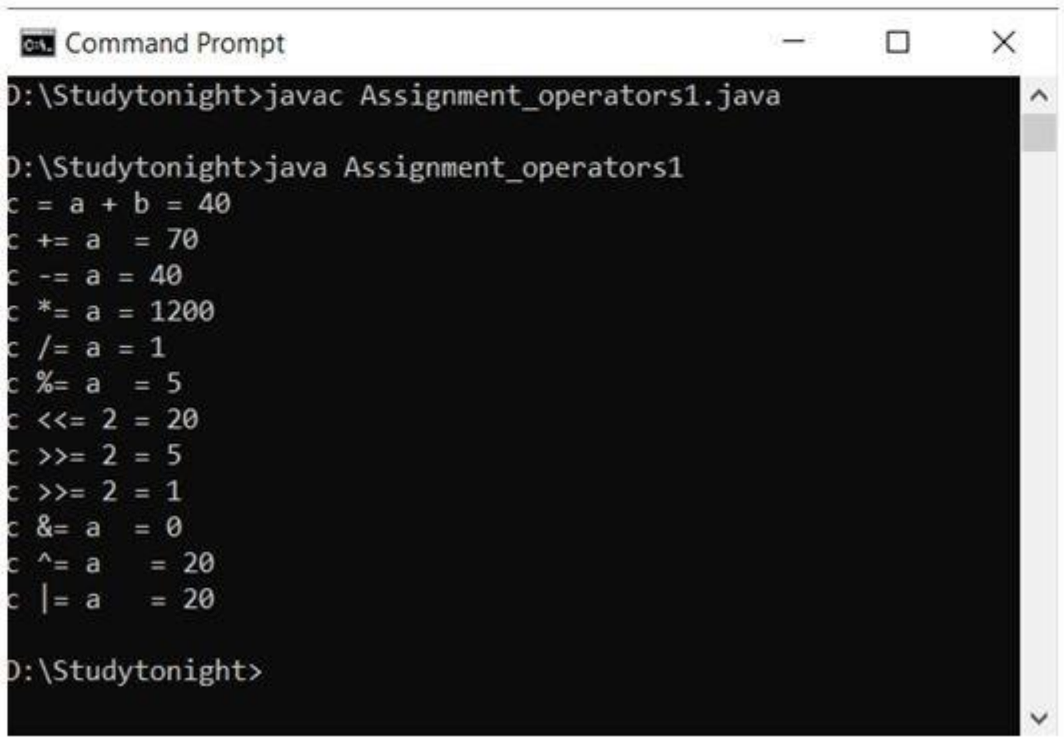
Example:

Lets create an example to understand use of assignment operators. All assignment operators have right to left associativity.

```
class Assignment_operators1{

public static void main(String as[])

{

    int a = 30;

    int b = 10;

    int c = 0;


      c = a + b;

System.out.println("c = a + b = " + c );


      c += a ;

System.out.println("c += a  = " + c );


      c -= a ;

System.out.println("c -= a = " + c );


      c *= a ;

System.out.println("c *= a = " + c );
```

```
      a = 20;

      c = 25;

      c /= a ;

System.out.println("c /= a = " + c );


      a = 20;

      c = 25;

      c %= a ;

System.out.println("c %= a  = " + c );


      c <<= 2 ;

System.out.println("c <<= 2 = " + c );


      c >>= 2 ;

System.out.println("c >>= 2 = " + c );


      c >>= 2 ;

System.out.println("c >>= 2 = " + c );


      c &= a ;

System.out.println("c &= a  = " + c );


      c ^= a ;

System.out.println("c ^= a   = " + c );


      c |= a ;

System.out.println("c |= a   = " + c );

    }

}
```

Copy



```
D:\Studytonight>javac Assignment_operators1.java

D:\Studytonight>java Assignment_operators1
c = a + b = 40
c += a  = 70
c -= a = 40
c *= a = 1200
c /= a = 1
c %= a  = 5
c <<= 2 = 20
c >>= 2 = 5
c >>= 2 = 1
c &= a  = 0
c ^= a   = 20
c |= a   = 20

D:\Studytonight>
```

Misc. operator

There are few other operator supported by java language.

Conditional operator

It is also known as ternary operator because it works with **three operands. It is short alternate of if-else statement.** It can be used to evaluate Boolean expression and **return either true or false** value

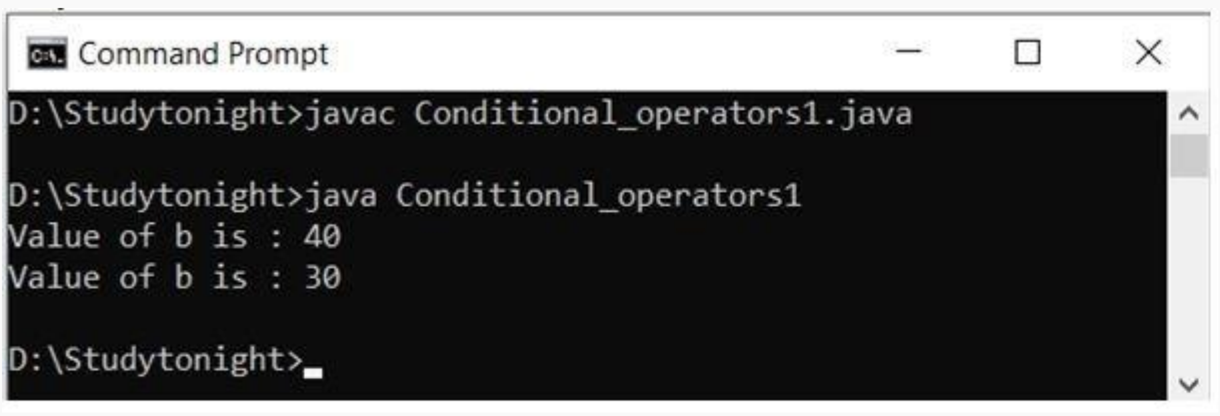
```
epr1 ? expr2 : expr3
```

Copy
Example:

In ternary operator, if **epr1** is true then expression evaluates after **question mark (?)** else evaluates **after colon (:)**. See the below example.

```
class Conditional_operators1{

public static void main(String as[])

{

int a, b;

      a = 20;

      b = (a == 1) ? 30: 40;

System.out.println( "Value of b is : " +  b );



      b = (a == 20) ? 30: 40;
System.out.println( "Value of b is : " + b );

}

}
```

Copy



```
D:\Studytonight>javac Conditional_operators1.java

D:\Studytonight>java Conditional_operators1
Value of b is : 40
Value of b is : 30

D:\Studytonight>_
```

instanceOf operator

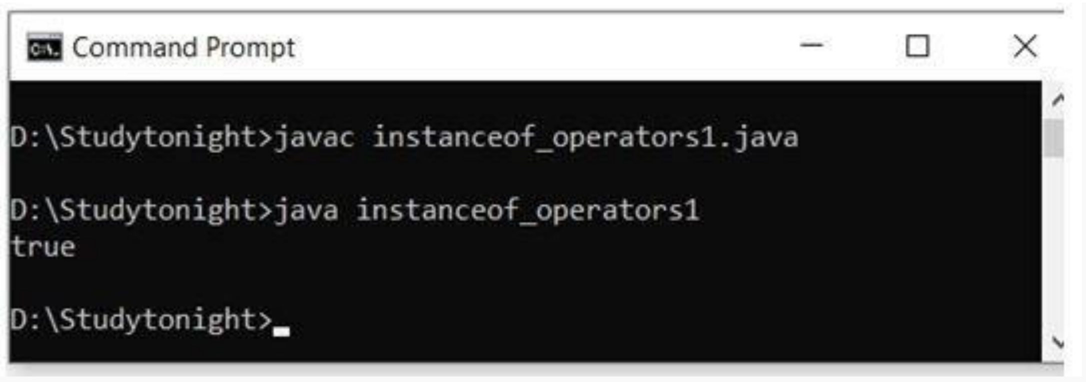
It is a java **keyword** and used to test whether the given **reference belongs to provided type** or not. Type can be a class or interface. **It returns either true or false**.

Example:

Here, we created a string reference variable that stores "studytonight". Since it stores string value so we test it using isinstance operator to check whether it belongs to string class or not. See the below example.

```
class instanceof_operators1{

public static void main(String as[])

{

      String a = "Studytonight";

boolean b = a instanceof String;

System.out.println( b );

}

}
```

Copy



```
D:\Studytonight>javac instanceof_operators1.java

D:\Studytonight>java instanceof_operators1
true

D:\Studytonight>_
```

Java Arrays

An array is a collection of similar data types. Array is a container object that hold values of **homogeneous** type. It is also known as static data structure because size of an array must be specified at the time of its declaration.

Array starts from **zero** index and goes to **n-1** where **n** is **length** of the array.

In Java, array is treated as an object and **stores into heap memory**. It allows to store primitive values or reference values.

Array can be single dimensional or multidimensional in Java.

Features of Array

- It is always indexed. Index begins from 0.
- It is a collection of similar data types.
- It occupies a contiguous memory location.
- It allows to access elements randomly.

Single Dimensional Array

Single dimensional array use single index to store elements. You can get all the elements of array by just increment its index by one.

Array Declaration

Syntax :

```
datatype[] arrayName;;

or

datatype arrayName[];
```

Copy

Java allows to declare array by using both declaration syntax, both are valid.

The **arrayName** can be any valid array name and datatype can be any like: **int, float, byte** etc.

Example :

```
int[ ] arr;
```

```
char[ ] arr;

short[ ] arr;

long[ ] arr;

int[ ][ ] arr;    // two dimensional array.
```

Copy

Initialization of Array

Initialization is a process of allocating memory to an array. At the time of initialization, we specify the size of array to reserve memory area.

Initialization Syntax

```
 arrayName = new datatype[size]
```

Copy

`new` operator is used to initialize an array.

The **arrayName** is the name of array, new is a keyword used to allocate memory and size is length of array.

We can combine both declaration and initialization in a single statement.

```
Datatype[] arrayName = new datatype[size]
```

Copy

Example : Create An Array

Lets create a single dimensional array.

```


class Demo

{

public static void main(String[] args)

  {

    int[] arr = new int[5];

        for(int x : arr)

        {

                System.out.println(x);

        }

   }
}
```

Copy

```
0
0
0
0
0
```

In the above example, we created an array **arr** of **int type** and can store **5** elements. We iterate the array to access its elements and it prints five times zero to the console. It prints zero because we did not set values to array, so all the elements of the **array initialized to 0 by default.**

Set Array Elements

We can set array elements either at the time of initialization or by assigning direct to its index.

```
int[] arr = {10,20,30,40,50};
```

Copy

Here, we are assigning values at the time of array creation. It is useful when we want to store static data into the array.

or

```
arr[1] = 105
```

Copy

Here, we are assigning a value to array's 1 index. It is useful when we want to store dynamic data into the array.

Array Example

Here, we are assigning values to array by using both the way discussed above.

```
class Demo

{

public static void main(String[] args)

  {

     int[] arr = {10,20,30,40,50};

        for(int x : arr)

        {

                System.out.println(x);

        }
     // assigning a value


        arr[1] = 105;

        System.out.println("element at first index: " +arr[1]);

    }

}
```

Copy

```
10

20

30

40

50

element at first index: 105
```

Accessing array element

we can access array elements by its index value. Either by using loop or direct index value. We can use loop like: for, for-each or while to traverse the array elements.

Example to access elements

```
class Demo

{

public static void main(String[] args)

  {

     int[] arr = {10,20,30,40,50};

        for(int i=0;i<arr.length;i++)

        {

                System.out.println(arr[i]);

        }


        System.out.println("element at first index: " +arr[1]);

    }

}
```

```
```
Copy

```
10
20
30
40
50
element at first index: 20
```

Here, we are traversing array elements using loop and accessed an element randomly.

Note:-To find the length of an array, we can use length property of array object like: `array_name.length`.

Multi-Dimensional Array

A multi-dimensional array is very much similar to a single dimensional array. It can have multiple rows and multiple columns unlike single dimensional array, which can have only one row index.

It represent data into tabular form in which data is stored into row and columns.

Multi-Dimensional Array Declaration

```
datatype[ ][ ] arrayName;
```
Copy

Initialization of Array

```
datatype[ ][ ] arrayName = new int[no_of_rows][no_of_columns];
```
Copy

The **arrayName** is the name of array, **new** is a keyword used to allocate memory and no_of_rows and no_of_columns both are used to **set size of rows and columns** elements.

Like single dimensional array, we can statically initialize multi-dimensional array as well.

```
int[ ][ ] arr = {{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};
```
Copy

Example:

```
class Demo
{
public static void main(String[] args)
  {
     int arr[ ][ ] = {{1,2,3,4,5},{6,7,8,9,10},{11,12,13,14,15}};

        for(int i=0;i<3;i++)

        {

          for (int j = 0; j < 5; j++) {


               System.out.print(arr[i][j]+" ");

               }

          System.out.println();



        }



     // assigning a value

        System.out.println("element at first row and second column: "
+arr[0][1]);

   }

}
```

Copy

```
1 2 3 4 5

6 7 8 9 10

11 12 13 14 15

element at first row and second column: 2
```

Jagged Array

Jagged array is an array that has different numbers of columns elements. In java, a jagged array means to have a multi-dimensional array with uneven size of columns in it.

Initialization of Jagged Array

Jagged array initialization is different little different. We have to set columns size for each row independently.

```
int[ ][ ] arr = new int[3][ ];

arr[0] = new int[3];

arr[1] = new int[4];
```

```
arr[2] = new int[5];
```

Copy

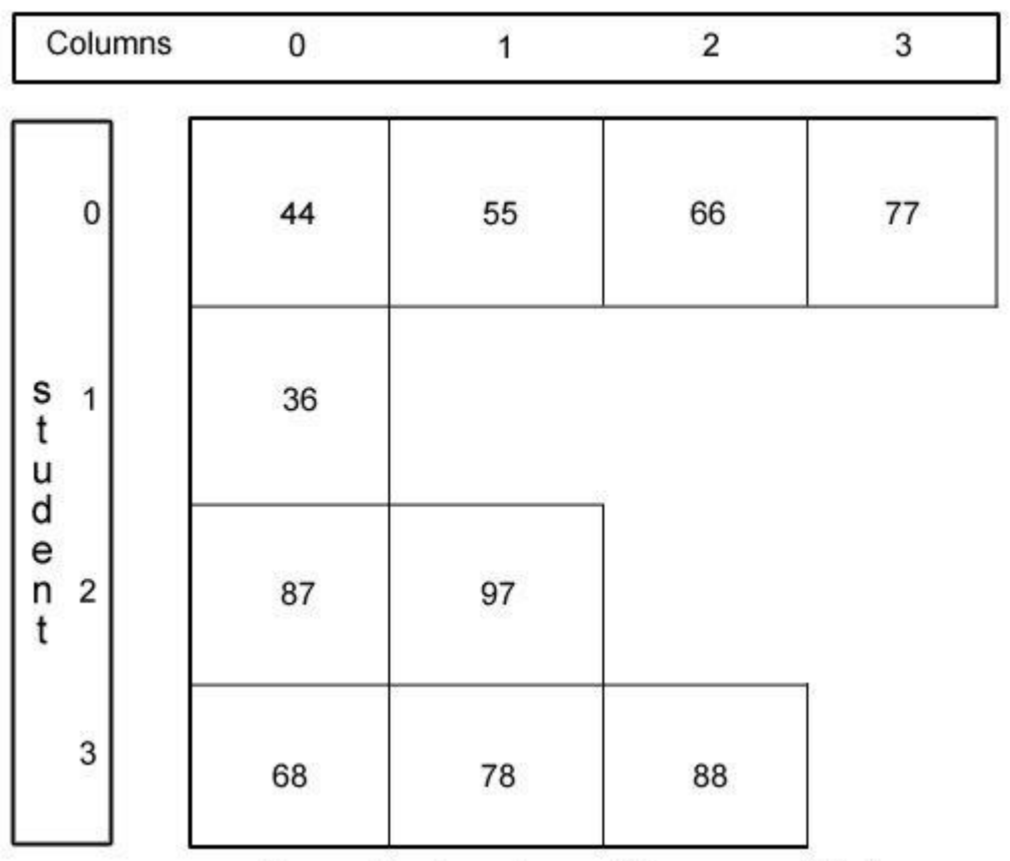


FIgure : Varying columns 2D array - matrix form

Example : Jagged Array

```
     class Demo

{

public static void main(String[] args)

  {

     int arr[ ][ ] = {{1,2,3},{4,5},{6,7,8,9}};

        for(int i=0;i<3;i++)

        {

          for (int j = 0; j < arr[i].length; j++) {


               System.out.print(arr[i][j]+" ");

               }

          System.out.println();



        }
```

```
    }

}
```

Copy

```
1 2 3

4 5

6 7 8 9
```

Here, we can see number of **rows are 3** and **columns are different for each row**. This type of array is called **jagged array**.

Different ways to create objects in Java

Java is an object-oriented language, everything revolve around the object. An object **represents runtime entity** of a class and is **essential to call variables and methods of the class**.

To create an object Java provides various ways that we are going to discuss in this topic.

1. New keyword
2. New instance
3. Clone method
4. Deserialization
5. NewInstance() method

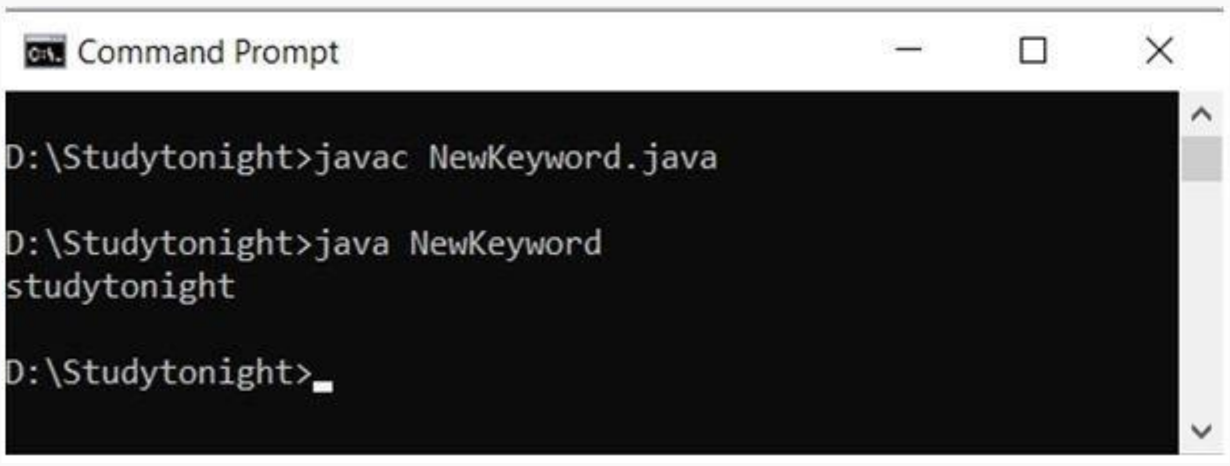
1) new Keyword

In Java, creating objects using **new keyword** is very popular and common. Using this method user or system defined default constructor is called that initialize instance variables. And new keyword creates a memory area in heap to store created object.

Example:

In this example, we are creating an object by using new keyword.

```
public class NewKeyword

{

    String s = "studytonight";

    public static void main(String as[])

    {

NewKeyword a = new NewKeyword();

System.out.println(a.s);

    }
}
```
Copy


```
D:\Studytonight>javac NewKeyword.java

D:\Studytonight>java NewKeyword
studytonight

D:\Studytonight>_
```

2) New Instance

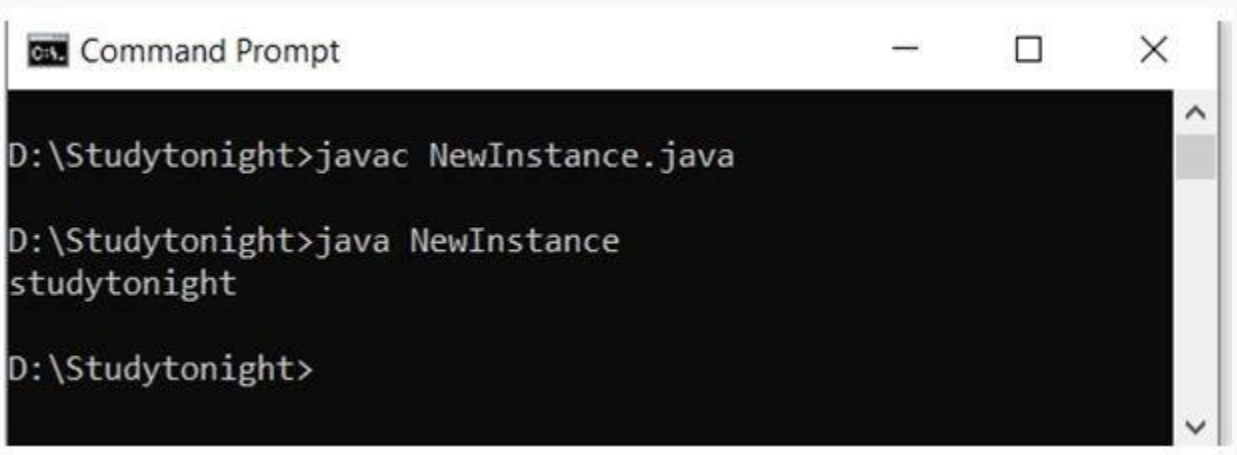
In this case, we use a static method `forName()` of **Class** class. This method loads the class and returns an object of type Class. That further we cast in our required type to get required type of object. This is what we do in this case.

Example:

You can see the example and understand that we loaded our class **NewInstance** by using **Class.forName()** method that returns the type Class object.

```
public class NewInstance

{

    String a = "studytonight";

    public static void main(String[] args)

    {

        try

        {

            Class b = Class.forName("NewInstance");

NewInstance c = (NewInstance) b.newInstance();

System.out.println(c.a);

        }

        catch (ClassNotFoundException e)

        {

e.printStackTrace();

        }

        catch (InstantiationException e)
        {
e.printStackTrace();
        }
        catch (IllegalAccessException e)
        {
e.printStackTrace();
        }
    }
}
```

Copy



```
D:\Studytonight>javac NewInstance.java

D:\Studytonight>java NewInstance
studytonight

D:\Studytonight>
```

3) Clone() method

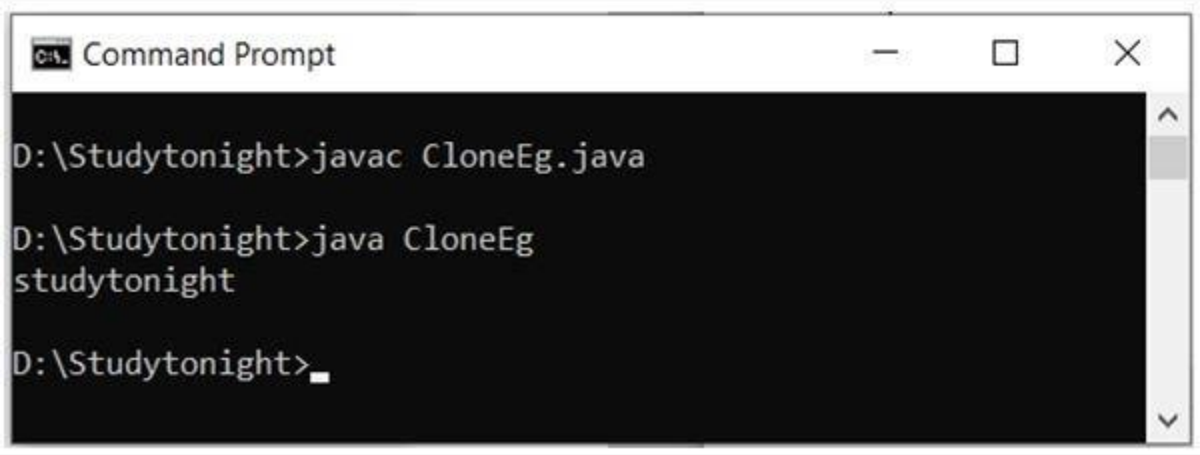
In Java, clone() is called on an object. When a clone() method is called JVM creates a new object and then copy all the content of the old object into it. When an object is created using the clone() method, a constructor is not invoked. To use the clone() method in a program the class implements the cloneable and then define the clone() method.

Example:

In this example, we are creating an object using clone() method.

```
public class CloneEg implements Cloneable
{
    @Override
    protected Object clone() throws CloneNotSupportedException
    {
        return super.clone();
    }
    String s = "studytonight";


    public static void main(String[] args)
    {
CloneEg a= new CloneEg();
        try
        {
CloneEg b = (CloneEg) a.clone();
System.out.println(b.s);
        }
        catch (CloneNotSupportedException e)
        {
e.printStackTrace();
        }
    }
}
```

Copy



```
D:\Studytonight>javac CloneEg.java

D:\Studytonight>java CloneEg
studytonight

D:\Studytonight>_
```

4) deserialization

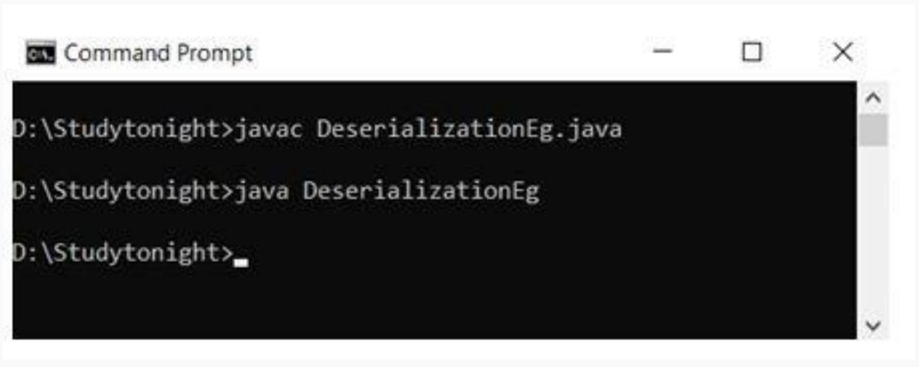
In Java, when an object is serialized and then deserialized, JVM create another separate object. When deserialization is performed JVM does not use any constructor for creating an object.

Example:

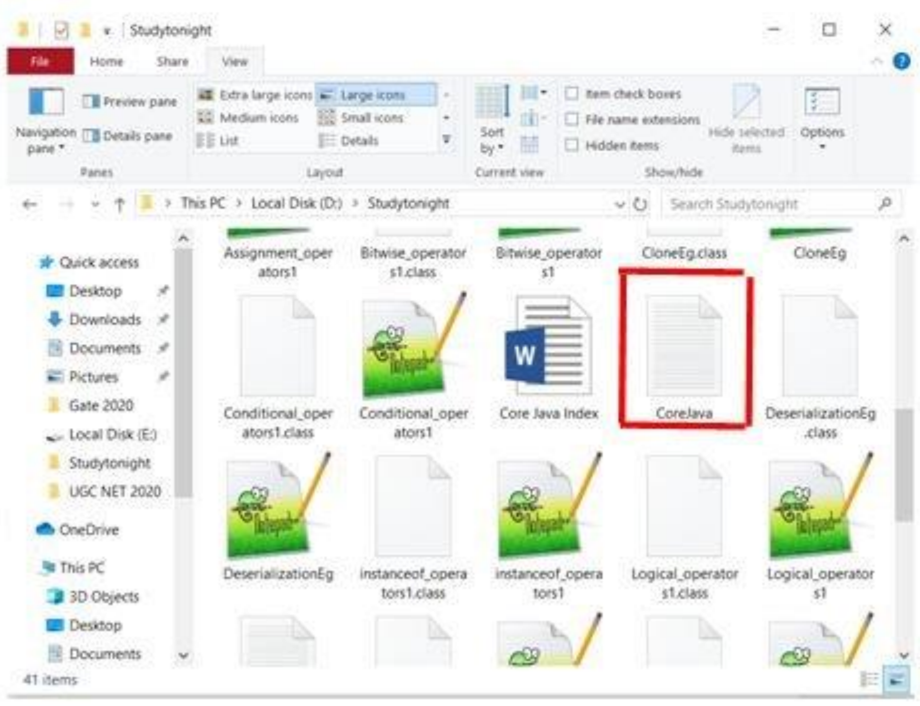
Lets see an example of creating object by deserialization concept.

```
import java.io.*;

class DeserializationEg implements Serializable
{
    private String a;
DeserializationEg(String name)
    {
this.a = a;
    }

    public static void main(String[] args)
    {
        try
        {
DeserializationEg b = new DeserializationEg("studytonight");
FileOutputStream c = new FileOutputStream("CoreJava.txt");
ObjectOutputStream  d = new ObjectOutputStream(c);
d.writeObject(b);
d.close();
d.close();
        }
        catch (Exception e)
        {
e.printStackTrace();
        }
    }
}
```

Copy



```
D:\Studytonight>javac DeserializationEg.java

D:\Studytonight>java DeserializationEg

D:\Studytonight>_
```



newInstance() method of Constructor class

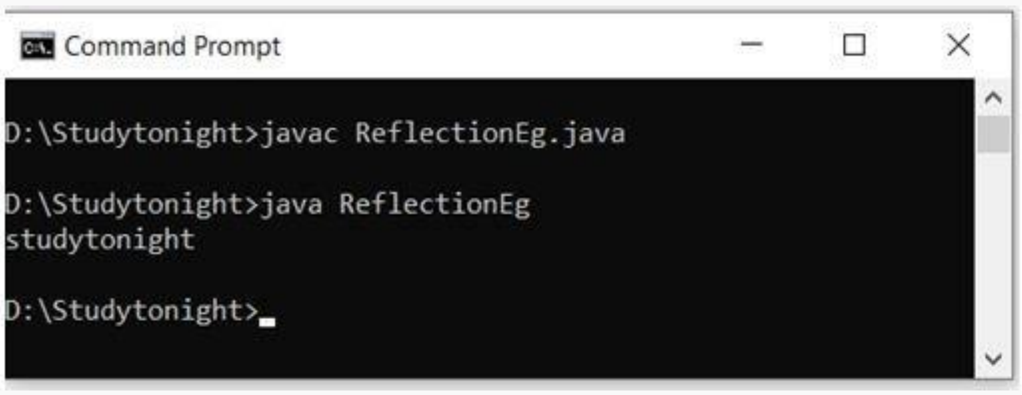
In Java, Under **java.lang.reflect package** Constructor class is located. We can use it to create object. The Constructor class provides a method newInstance() that can be used for creating an object. This method is also **called a parameterized constructor.**

Example:

Lets create an example to create an object using newInstance() method of Constructor class.

```
import java.lang.reflect.*;


public class ReflectionEg

{

private String s;

ReflectionEg()

{

}

public void setName(String s)
{

this.s = s;

}

public static void main(String[] args)

{

try

{

Constructorconstructor = ReflectionEg.class.getDeclaredConstructor();

ReflectionEg r = constructor.newInstance();

r.setName("studytonight");

System.out.println(r.s);

}

catch (Exception e)

{

e.printStackTrace();

}

}

}
```

Copy



```
D:\Studytonight>javac ReflectionEg.java

D:\Studytonight>java ReflectionEg
studytonight

D:\Studytonight>_
```

Java Command line argument

The command line argument is the argument that passed to a program during runtime. It is the way to pass argument to the main method in Java. These arguments store into the String type **args** parameter which is main method parameter.

To access these arguments, you can simply traverse the args parameter in the loop or use direct index value because args is an array of type String.

For example, if we run a HelloWorld class that contains main method and we provide argument to it during runtime, then the syntax would be like.

```
java HelloWorld arg1 arg2 ...
```

Copy

We can pass **any number of arguments** because argument type is an array. Lets see an example.

Example

In this example, we created a class HelloWorld and during running the program we are providing command-line argument.

```
class cmd

{

    public static void main(String[] args)

    {

        for(int i=0;i< args.length;i++)

        {

            System.out.println(args[i]);

        }

    }

}
```

Copy

Execute this program as java cmd 10 20 30

```
10

20

30
```

To terminate the program in between based on some condition or program logic, Java provides **exit()** that can be used to terminate the program at any point. Here we are discussing about the **exit()** method with the example.

Java System.exit() Method

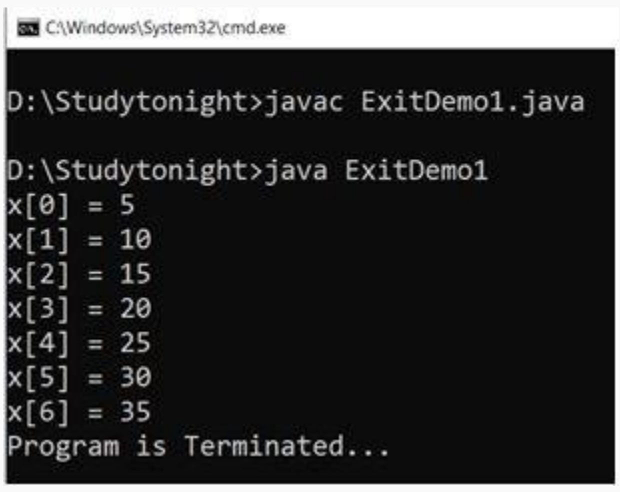
In Java, exit() method is in the java.lang.System class. This method is used to take an exit or terminating from a running program. It can take either zero or non-zero value. exit(0) is used for successful termination and exit(1) or exit(-1) is used for unsuccessful termination. The exit() method does not return any value.

Example:

In this program, we are terminating the program based on a condition and using exit() method.

```
import java.util.*;

import java.lang.*;


class ExitDemo1

{

    public static void main(String[] args)

    {

        intx[] = {5, 10, 15, 20, 25, 30, 35, 40, 45, 50};


        for (inti = 0; i<x.length; i++)

        {

            if (x[i] >= 40)

            {

                System.out.println("Program is Terminated...");

                System.exit(0);

            }

            else

                System.out.println("x["+i+"] = " + x[i]);

        }

    }

}
```

Copy



```
D:\Studytonight>javac ExitDemo1.java

D:\Studytonight>java ExitDemo1
x[0] = 5
x[1] = 10
x[2] = 15
x[3] = 20
x[4] = 25
x[5] = 30
x[6] = 35
Program is Terminated...
```