JAVA:

—-----

1. Core Java
2. Adv Java
3. J2EE

JAVA:

—------

1. J2SE / JAVA SE
2. J2EE / JAVA EE
3. J2ME / JAVA ME

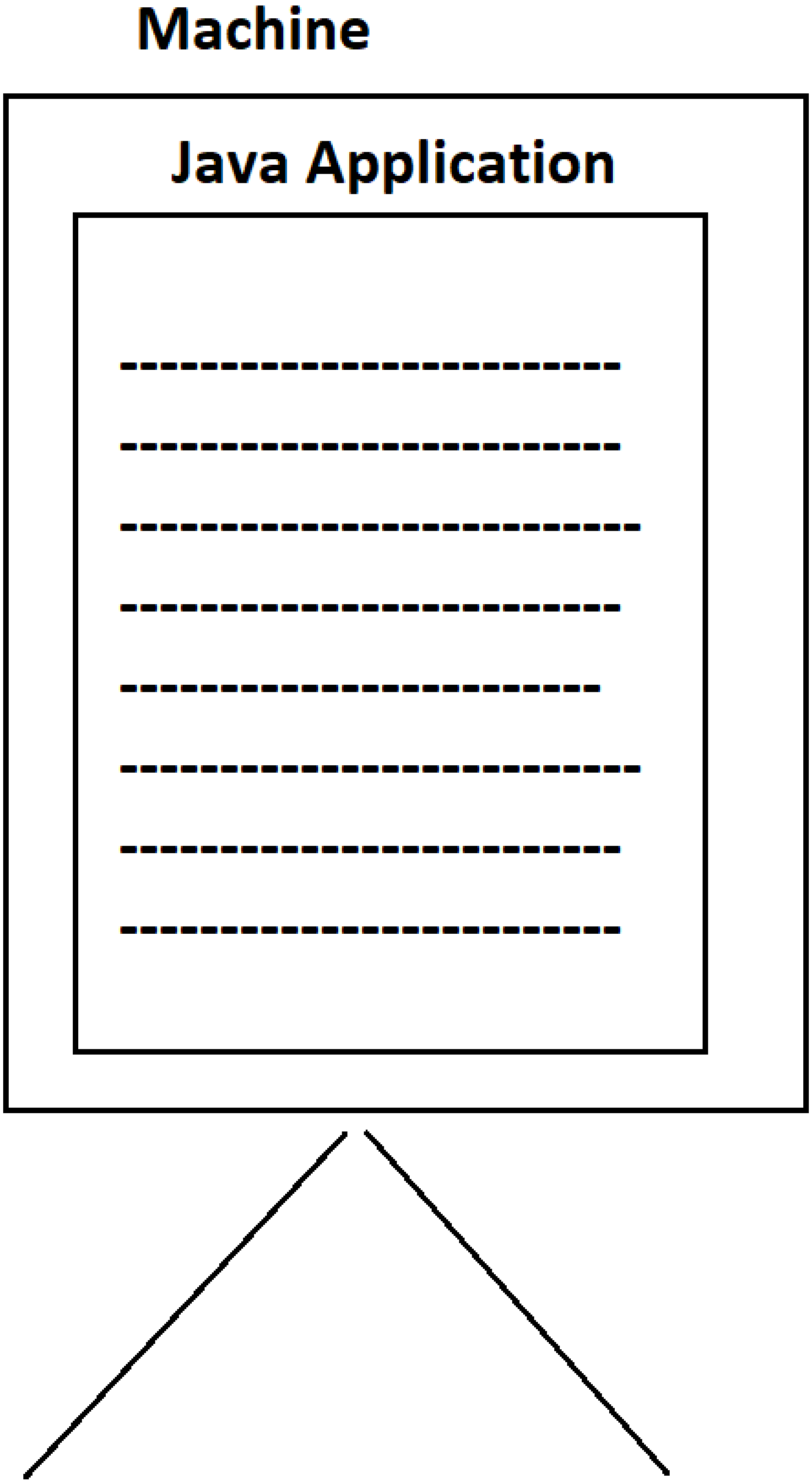
J2SE / JAVA SE

—-------------------

1. Java 2 Standard Edition
2. It will cover only the fundamentals of the Java programming language.
3. Standalone Applications.
4. If we design and execute any application without using client-server arch or without distributing application logic over multiple machines then that application is called as “Standalone Application”.
5. EX: Calculator, paint, notepad, editplus, notepad++
6. 5% of the applications are standalone applications
7. Its dependency level is 1000%.
8. Java is a dependent Programming language for
    1. J2EE
    2. Android
    3. Selenium

—-

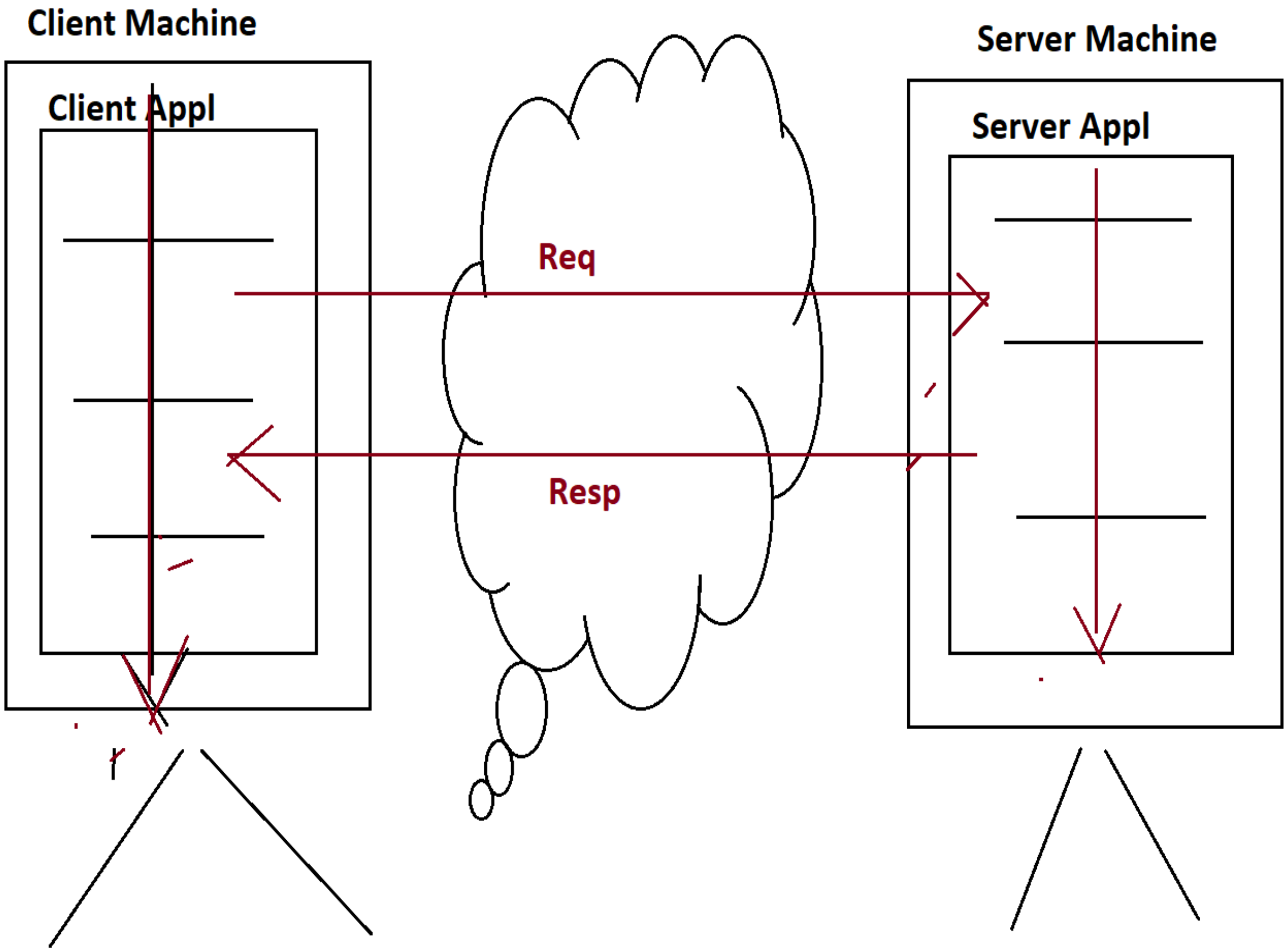
—-


Machine
Java Application

2. J2EE/JAVA EE

----

1. Java 2 Enterprise Edition
2. It will cover Server side programming
3. Distributed Applications / Enterprise Applications
4. If we design and execute any application on the basis of Client-Server arch or by distributing application logic over multiple machines then that application is called a Distributed application or Enterprise application.



5. 95% of the applications are distributed applications.

3. J2ME / JAVA ME

1. Java 2 Micro Edition
2. Micro programming
3. Mobile Based Applications
4. If we design and execute any application on the basis of Mobile H/W system then that application is called as Mobile Based Application.
5. Calculator in Mobile phones, Calender mobile phones,...
6. Mobile application are having very good demand in out side market, but, no demand is available for J2ME.

Content of Core Java:

1. Introduction
2. Steps to Prepare First Java Application
3. Language Fundamentals
4. OOPs ************** 100
5. Wrapper Classes
6. String Manipulations ***** 10
7. Inner Classes
8. Exception Handling ***********50
9. Multi Threading *********** 50
10. IOStreams ****5
11. Networking
12. RMI

13. Arrays
14. Collection Framework*************80
15. Generics
16. Reflection API
17. Annotations ******10
18. JVM Arch
19. Garbage Collection
20. I18N[Internationalization]
21. GUI
22. Regular Expressions
23. Java 8 Features**********50
24. JAVA 9 Features **** 5
25. —--

—--

—-

35. Java 19 Version Features

Introduction:
—---------------

1. Java History
2. Differences between Java and Others[C and C++]
3. Java Features
4. Java Naming Conventions
5. Java Programming Format

Java History:
—-------------

1991, SUN Microsystems had a requirement to prepare new PL , by using new PL SUN Microsystems wants prepare softwares for electronic consumer goods like Remote Controllers, Cable TV Switch Boxes,....

30 members Team → James Gosling and Patrick Naughtan

Green

```
    PL ---->  1. Simple PL
              2. Tight Coded PL
              3. Arch Neutral PL
```

Simple PL:
→ Less execution time --> More Performance
→ Less Memory Consumption -→ Less cost for end product
→ Less Power consumption → Less Maintenance cost

Tight Coded:
→ In C PL --> Stack

```
  1.PUSH ----> 20 LOC
  2.POP ------> 20 LOC
  3.PEEK -----> 20 LOC
    ---------------------------
    Total ----> 60 LOC
```

IN JAVA:

```
    Stack s = new Stack();
    s.push("AAA");
    s.pop();
       s.peek();
-----------------------------
Total ----> 4 LOC
```

Arch Neutral PL:
It must execute its functionalities in all the H/W systems.

End of 1992, Prepared new Programming language.

OAK

JAVA

First Product   *7 , it is a Remote Controller.
1993, up to half of 1994

Mid of 1994, www as part of internet,

Browser

Musaik → Mark Anderson → UG student → Netscafe communications

Id of 1994, New Browser → Janathan Paine and Patric Naughtan
End of 1995 → new Browser → Hot Java Browser

1996, JAN, 27th, --> JAVA 1.0 -> JDK1.0

Differences between Java and others[C and C++]
----

1. C and C++ are Static Programming Languages but Java is a dynamic Programming language:

If any PL allows memory allocation for primitive data types at compilation time then that programming language is called as “Static Programming Language”.

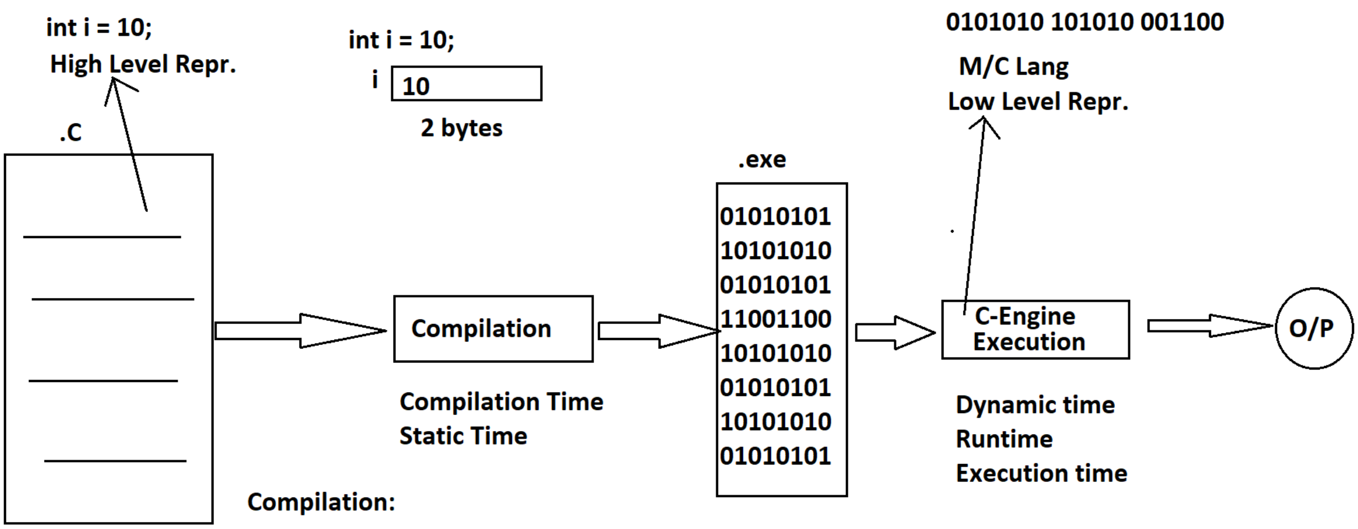
EX: C and C++

In C and C++ applications, memory will be allocated for the primitive data types at compilation time only.

If any PL allows memory allocation for the primitive data types at runtime then that PL is called as Dynamic Programming Language.

EX: JAVA

In Java applications memory will be allocated for the primitive data types at runtime only.



Compilation:

1. To tranhslate program from HL repr to LL repr.
2. To check developers mistakes in applications.

2.Pre-Processor is required in C and C++, but, Pre-Processor is not required in JAVA:

Preprocessor is the first phase in Compilation.

In C and C++, the complete predefined library was provided in the form of header files.

```
EX: stdio.h
    conio.h
    math.h
```

If we want to use the predefined library in C and C++ applications , first, we have to include header files into C and C++ applications.

To include header files in C and C++ applications we have to use #include<>.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
```

When we compile C and C++ program Preprocessor will perform the following actions.

1. Preprocessor will recognize all #include<> statement and Preprocessor will take the provided header files names.
2. Preprocessor will check whether these header files exist or not in C and C++ software.
3. If the specified header files do not exist in C and C++ softwares then errors will be raised.
4. If the specified header files exist in C and C++ softwares then the preprocessor will load these header files content to the memory.

   Loading a predefined library at compilation time is called “static Loading”.

In C and C++ applications, Preprocessor is required to recognize #include<> statements in order to load header files to the memory.

In Java, the complete predefined library was provided in the form of classes and interfaces in packages.

```
EX: java.io
    java.util
    java.sql
```

In Java applications, if we want to use a predefined library then we have to import the required packages, to import the packages we have to use the “import” statement.

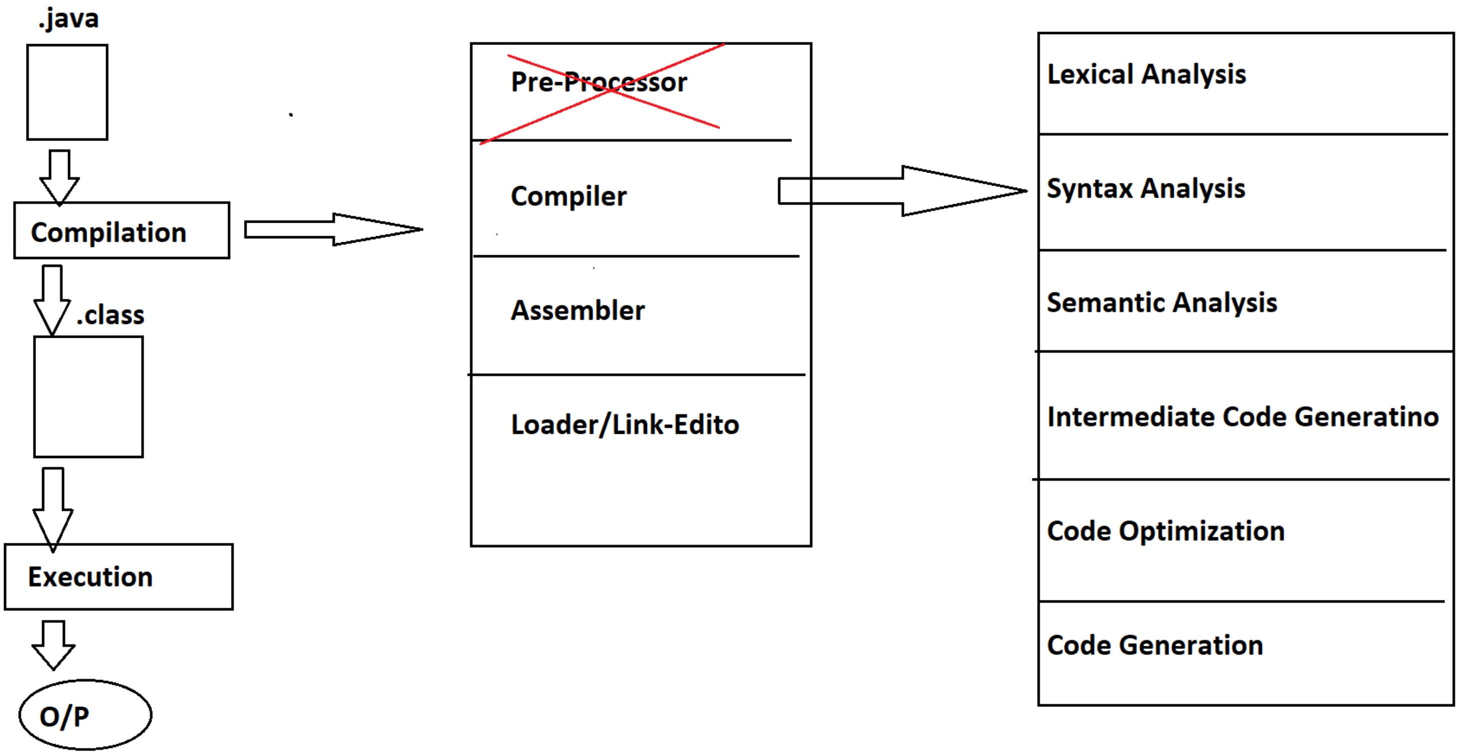
```
EX: import java.io.*;
    import java.util.*;
    import java.sql.*;
```

When we compile java application , compiler will perform the following actions.

1. Compiler will recognize all the import statements, compiler will take the provided package names.
2. Compiler will check whether these packages exist or not in java software.
3. If the specified packages are not existed then compiler will raise an error.
4. If the specified packages are existed then compiler will not raise any error and at the same time compiler will not load any package content to the memory.

    Note: In Java, at runtime , JVM will load the required library to the memory , this type of loading is called as “Dynamic Loading”.

IN Java, preprocessor is not required , because, in java #include<> statement s and header files are not existed.



Q)What are the differences between #include<> statement and import statement?

Ans:

---

1. #include<> statements are available in C and C++.
   Import statements are available in JAVA.

2. #include<> statements are able to include predefined libraries existing in header files.

   Import statements are able to include the predefined library existing in packages.

3. #include<> statements are evaluated by Preprocessor.

   Import statements are evaluated by both compiler and JVM.

4. #include<> statements supporting “Static Loading”.

   Import statements are supporting “Dynamic Loading”.

5. If we want to include more than one header file in C and C++ applications then we have to use more than one #include<> statement.

   ```
   EX: #include<stdio.h,conio.h,math.h> —-> Invalid
   EX:
   #include<stdio.h>
   #include<conio.h>
   #include<math.h>
   Status : Valid
   ```

   In Java, by using a single import statement we are able to import more than one class or more than one interface of the same package.

   EX: import java.io.*;

3. C and C++ are platform Dependent Programming languages, but, Java is a platform Independent programming Language:

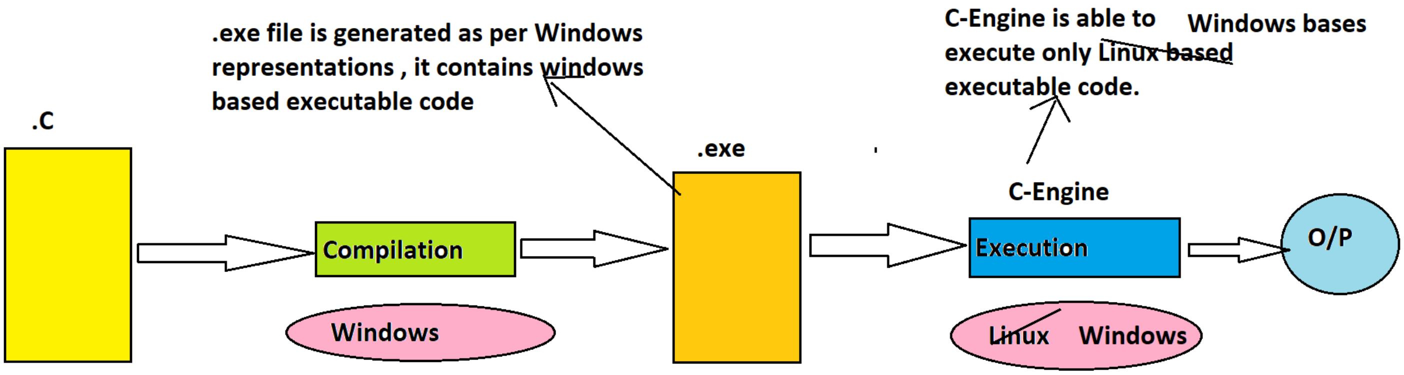
———

If any Programming language allows its applications to perform compilation and execution on the same operating system then that programming language is called a Platform Dependent Programming language.

EX: C and C++

If we compile C programming on windows operating systems then Compiler will generate windows represented .exe file, here the windows represented .exe file will be executed on only windows operating system, not possible to execute on other Operating Systems.

Due to the above reason , C and C++ are Platform Dependent programming languages.



.exe file is generated as per Windows representations , it contains windows based executable code

C-Engine is able to execute only ~~Linux based~~ executable code. Windows bases

If any Programming language allows its applications to perform compilation on one Operating system and execution is on another operating system then that Programming Language Platform Independent   Programming Language.

EX: JAVA

If we compile a Java application on Windows Operating System then the compiler will generate .class file, it contains bytecode, it is not directly executable code, it is an intermediate code and it does not have any Operating System representations.

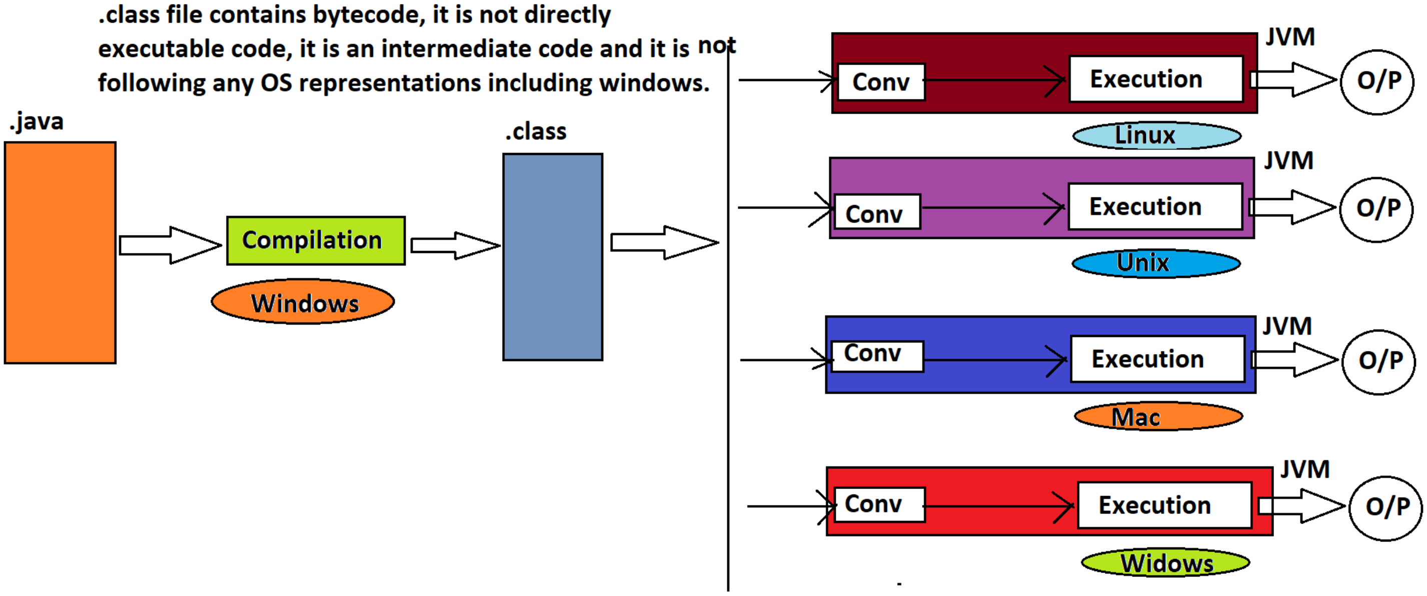
If we want to execute bytecode on any of the Operating Systems then that neutral bytecode must be converted to the respective Operating System representations.

In the above context, SUN Microsystems has provided the required converters in order to execute java applications in multiple Operating systems that is to make java as platform independent Programming language.

In the above context, SUN Microsystems has provided all converters along with the Execution process in a single component inside Java software that is “JVM”.

Java is a platform independent Programming language because of JVMs , but, JVM is Platform Dependent.

Note: Not only JVM, the complete Java Software is platform Dependent.



.class file contains bytecode, it is not directly executable code, it is an intermediate code and it is not following any OS representations including windows.

Q)What are the differences between .exe file and .class file?

--

Ans:

----

1. .exe files exist in C and C++.

   .class files exist in Java.

2. .exe file contains directly executable code.

   .class file contains bytecode, it is not directly executable code, it is an intermediate code.
3. .exe file is platform dependent file.

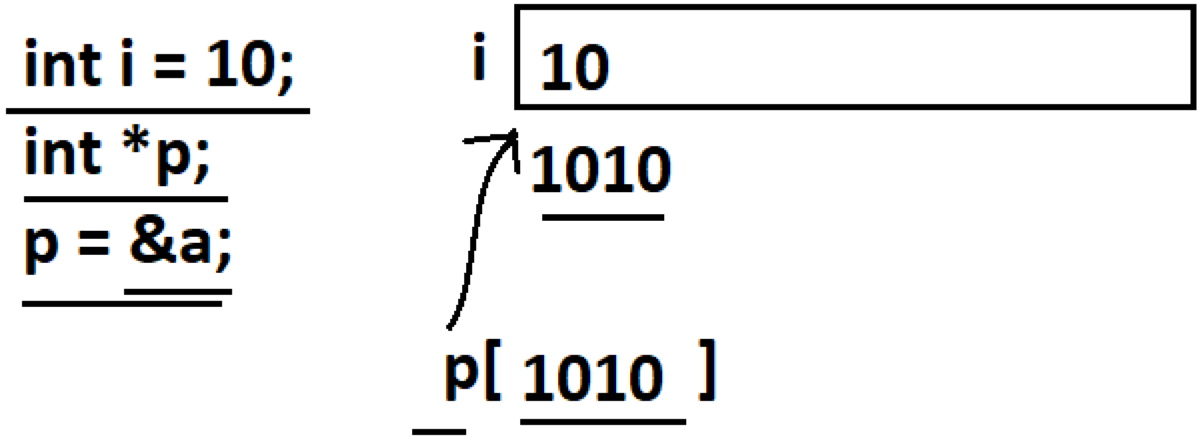
   .class file is platform independent file.

4..exe files are providing less security.

.class files are providing more security.

Pointers are existed in C and C++, but, Pointers are not existed in Java:

Pointer is a variable, it is able to store address locations of the data structures, where data structures may be a variable, an array, a struct, another pointer variable.



```
int i = 10;
int *p;
p = &a;
```

Q)Why pointer variables are suitable in C and C++ and why pointer variables are not suitable in Java?

Ans:

----

1. Pointer variables required static memory allocation, but Java is following Dynamic memory allocation.
2. Pointers are supported by Static Programming languages, but Java is a dynamic programming language.

3. Pointer variables are suitable in Platform Dependent programming Languages, but Java is platform independent Programming language.
4. Pointers is a bit confusion oriented feature, but java is a simple programming language, it should not allow any confusion oriented feature.

Q)What are the differences between pointer variables and Reference variables?
—--

Ans:
—---

1. Pointer variables exist in C and C++.
   Reference variables exist in Java.

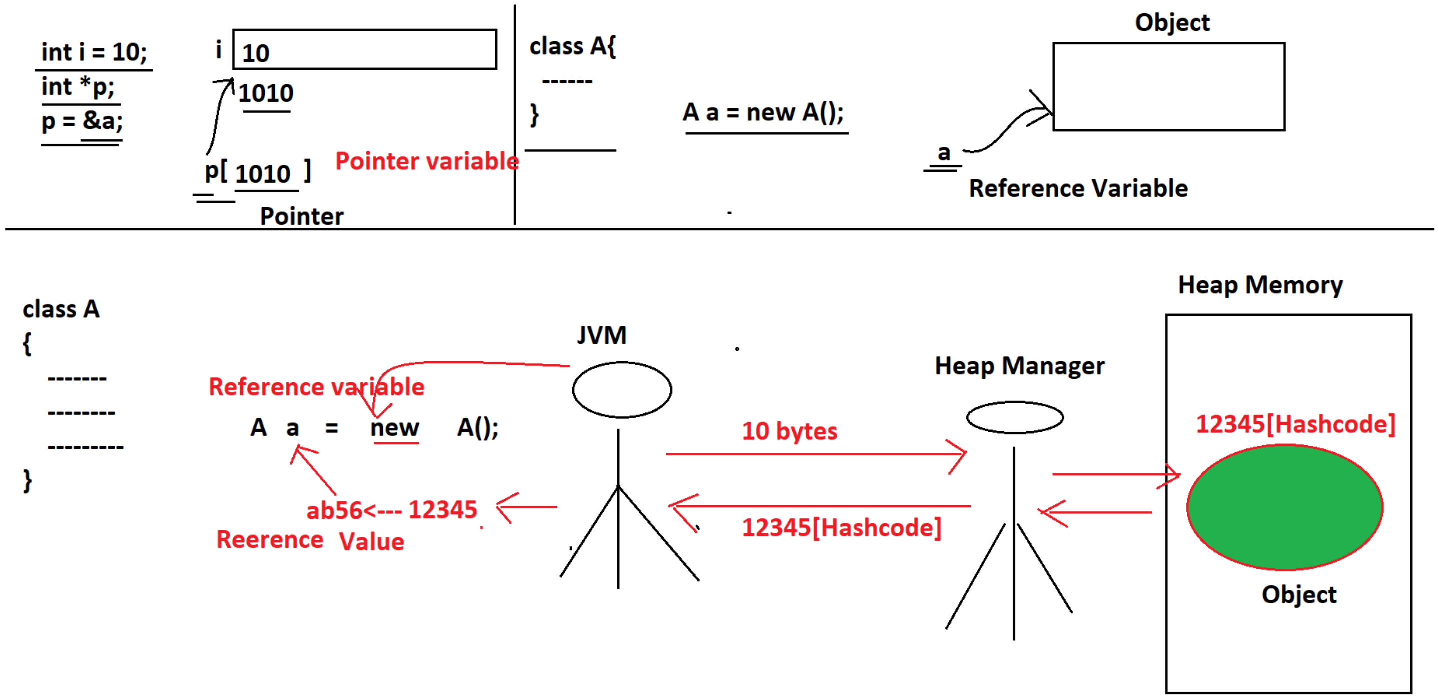
2. Pointer variables are able to refer to a block of memory by storing address locations.

   Reference variables are able to refer to a block of memory[Object] by storing Object reference value.

   Note: Object reference value is the hexa decimal form of hashcode, where Hashcode is an unique identity for each and every Object provided by heap manager in the form of an integer value.

3. Pointer variables are very much suitable in static programming languages and Platform Dependent Programming languages.

Reference variables are very much suitable in Dynamic Programming Languages and Platform independent Programming languages.



5. Multiple Inheritance is not possible in Java:

If any PL represents data in the form of Objects as per Object Oriented Features then that PL is called as an Object Oriented Programming language.

There are seven no of Object oriented Features to represent data in Object Oriented Programming languages.

1. Class
2. Object
3. Encapsulation
4. Abstraction
5. Inheritance
6. Polymorphism
7. Message Passing

In Object orientation , all the real world entities are declared as classes.

EX:

```
class Student{
}
class Product{
}
class Customer{
}
```

Inheritance: It is a relation between classes , it will provide variables and methods from one class[Super class] to another class[Sub class] in order to improve Code Reusability.

The main intention of Inheritance is to provide Code Reusability.

Initially there are two types of Inheritances.

1. Single Inheritance
2. Multiple Inheritance

On the basis of the above inheritances, there are three more inheritances.

1. Multi Level Inheritance.
2. Hierarchical Inheritance.
3. Hybrid Inheritance

Single Inheritance: It is a relation between classes, it will provide variables and methods from only one super class to one or more no of sub classes.

Java does support Single INheritance.
2. Multiple Inheritance: It is a relation between classes, it will provide variables and methods from more than one superclass to one or more no of sub classes.

Java does not support Multiple Inheritance.

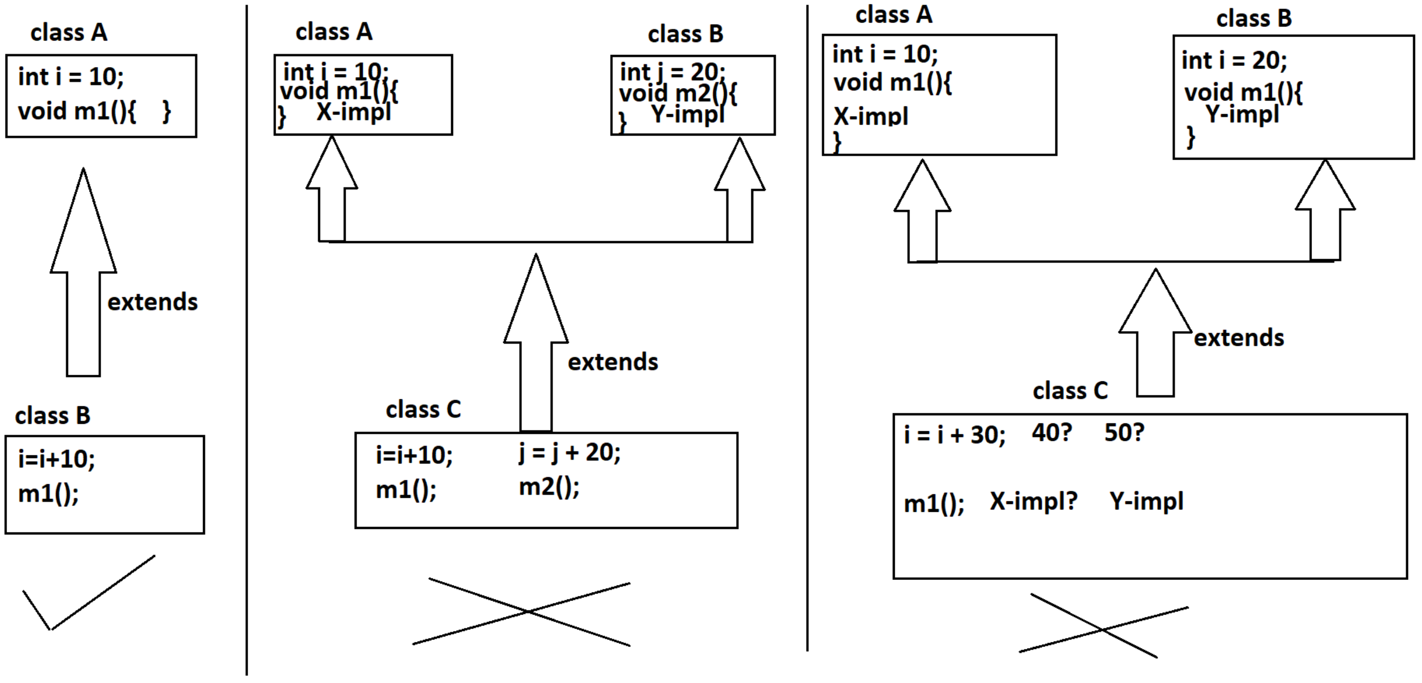
In case of Multiple inheritance, if we declare same variables and same methods with different values and with different implementations in more than one superclass and if we access that variables and methods in the respective sub class then it will be a confusion state for compiler and JVM that to get variable values and methods implementations from which super class.

Java is a simple programming language, it must not allow the confusion oriented features like Multiple Inheritance.

To restrict Multiple inheritance, JAVA has defined ‘extends’ keyword in such a way that to allow only one super class name , not to allow more than one superclass name in subclass declaration.

EX:

```
class A{
}
class B extends A{
}
Status : Valid

EX:
class A{
}
class B{
}
class C extends A,B{
}
Status : Invalid
```



```
Note: In Python Multiple Inheritance is possible
class A:
    i = 10
```

```
class B:
       i = 20

Case#1
class C(A):
    Void m1():
        print(i) // 10



case#2
class C(B):
    Void m1():
        print(i)//20

case#3
class C(A,B):
    void m1():
        print(i)//  10
Case#4
class C(B,A):
    void m1():
        print(i)//20
```

6. Destructors are existed in C++, but, Destructors are not existed in Java:

—--

In Object oriented programming languages, it is a minimum convention to represent data in the form of Objects.

In Object oriented programming languages, to manage the data in the form of objects, first we have to create the objects , atleast at the end of the applications we have to destroy the objects.

Due to the above reasons, we must perform the following two operations in Object oriented programming languages.

1. Creating Objects
2. Destroying Objects

To create objects , all the Object oriented programming languages are providing a separate feature called “Constructors”.

To destroy objects, almost all the Object oriented programming languages are providing a separate feature called as “Destructors”.

In C++, developers are responsible for creating objects and destroying objects, so developers must use “Destructors” explicitly to destroy objects.

In Java, developers are responsible for only creating objects, not responsible for destroying objects, because, in Java, there is an internal component to destroy objects automatically , here the component which is provided by JAVA to destroy objects is called as “Garbage Collector”.

7. Operator Overloading is not possible in Java:
——

If any PL allows to represent data in the form of Objects then that PL is called an Object oriented programming language.

To represent data in the form of Objects we have to follow Object oriented Features.

There are 7 no of Object oriented Features.

1. Class
2. Object
3. Encapsulation
4. Abstraction
5. Inheritance
6. Polymorphism
7. Message passing

Polymorphism: Polymorphism is a Greek word, where Poly means many and Morphism means forms or Structures.

If one thing exists in more than one form then it is called Polymorphism.

The main advantage of Polymorphism is “Flexibility” to design/ Develop applications.

There are two types of Polymorphisms.

1. Static polymorphism
2. Dynamic Polymorphism

If the polymorphism exists at compilation time then it is called Static polymorphism.
EX: Overloading

If the polymorphism exists at runtime then that polymorphism is called Dynamic Polymorphism.

EX: Overriding

There are two types of Overloadings.

1. Method Overloading
2. Operator Overloading

If we declare more than one method with the same name and with the different parameter list then it is called Method Overloading.

```
class Math{
    Void add(int i , int j){
        int result = i + j;
    }
    Void add(float f1, float f2){
        float f = f1 + f2
    }
    Void add(double d1, double d2){
        double result = d1 + d2;
    }
}
```

Java does support method overloading.

If we define more than one operation or functionality for any single operator then it is called “Operator Overloading”.

EX:

```
int i = 10;
int j = 20;
int k = i + j; → here + is for  Arithmetic Addition.
System.out.prntln(k); → 30

String str1 = “abc”;
String str2 = “def”;
String str3 = str1 + str2; → Here + is for String concatenation.
System.out.println(str3); → abcdef
```

Java does not support operator overloading, because,

1. Operator overloading is a rarely used feature in application development.
2. It is a bit of a confusion oriented feature when we define more no of operators with more no of functionalities.

Note: As per the java internal requirement, JAVA made some of the predefined operators as overloaded operators with fixed functionalities, but, JAVA has not provided any environment to define operator overloading explicitly at developers level.

8. C and C++ are following call By value and call by reference parameter passing mechanisms, but Java is following only call by value parameter passing mechanism:

—---

In all the Programming Languages, if we pass primitive data [byte, short, int, long, float, double, boolean, char] as parameters to the methods then the parameter passing mechanism is “Call by Value” Parameter passing mechanism.

In any Programming Language, if we pass address locations as parameters to the methods then the parameter passing mechanism is “Call By Reference”.

In C and C++ applications, if we pass pointer variables as parameters to the methods then the parameter passing mechanism is

“Call By Reference”, because, in C and C++ applications pointer variables are able to represent address locations.

In Java applications, if we pass reference variable as parameter to the methods then the parameter passing mechanism is “Call By Value” only, not Call By Reference, because, in Java applications, reference variables are able to store object reference values, not address locations, where Object reference value is hexa-decimal form of Hashcode, where Hashcode is an unique identity provided to the each and every abject in the form of an integer value.

9. In C and C++ , integers will take 2 bytes of memory, characters will take 1 byte of memory, but in Java, integers will take 4 bytes of memory, characters will take 2 bytes of memory:
—---

In C and C++, Memory allocation for the primitive data types is not fixed, it is variable and it is dependent on the Operating System which we used for the applications .

In the case of Java, memory allocation for the primitive data types is fixed irrespective of the operating systems which we used.
byte —----------> 1 byte
short —---------> 2 bytes
int —------------> 4 bytes
long —-----------> 8 bytes
float —----------> 4 bytes
double —---------> 8 bytes
char —-----------> 2 bytes
boolean —--------> 1 bit

Q)In C and C++, to represent characters one byte of memory is sufficient then what is the requirement to provide 2 bytes of memory for characters in Java?

—---

Ans:

—---

In C and C++, all the characters are represented in the form of ASCII values, to store any ASCII value one byte of memory is sufficient.

In the case of JAVA, all the characters are represented in the form of UNICODEs , to store any UNICODE value 2 bytes of memory is required.

Q)What is UNICODE and what is its requirement in Java?

—---

Ans:

—----

UNICODE is one of the character representations, it is able to represent all the alphabet from all the natural languages like English, Hindi, Italian, Japanese,.....With these representations , it is able to provide very good Internationalization support in Java applications.

Q)What is Internationalization?

—-----------------------------------

Ans:

—---

Preparing applications with the local conventions of the users / Customers is called internationalization.

Java is providing a very good Internationalization Service due to the UNICODE representations only.

==

Java Features:

—---------------

To describe the nature of the Java programming language, JAVA has provided a set of features called Java features.

1. Simple
2. Object Oriented
3. Platform Independent
4. Arch Neutral
5. Portable
6. Robust
7. Dynamic
8. Secure
9. Distributed
10. Multi Threaded
11. Interpretive
12. High Performance

1. Simple:

—-------------

Java is a simple Programming language, because,

1. Java applications will take less memory and less execution time.

2. Java has removed all the confusion oriented features like Pointers, Multiple Inheritance,.....
3. Java is using all the simplified syntaxes from C and C++.

2. Object Oriented :

—-----------------------

Java is an Object oriented Programming language, because java allows to represent data in the form of Objects as per the Object oriented features.

3. Platform Independent:

—---------------------------

Java is a platform independent programming language, because java allows its applications to perform compilation on one operating system and execution on another operating system.

4. Arch Neutral :

—------------------

Java is an Arch neutral Programming language, because Java allows its applications to compile on one H/W system and execute on another H/W system.

5. Portable:

—------------

Java is a portable programming language, because Java allows its applications to execute in multiple Operating systems and in multiple H/W systems without changing anything in the Java application.

Nowadays Java applications are used in many machines like Computers, Mobiles, Parking meters, sensors, signaling systems,.........

Java is used to prepare all the types of applications like standalone applications, Web applications, Distributed applications, database related applications, some other products like selenium , salesforce, SAP ,....

6. Robust:

———————————

Java is a Robust programming language, because

1. Java has a very good memory management system in the form of Heap memory management system, it is a dynamic memory management system, it allocates and deallocates memory for the objects at runtime.
2. Java has very good exception handling mechanisms, because java has a very good predefined library to represent and handle almost all the exceptions which are coming frequently in java applications.

7. Dynamic:

—————————————

Java is a dynamic programming language, because Java allows memory allocation for the data at runtime.

8. Secure:

——————————

Java is a secure programming language, because

1. Java is very good at implicit security , because Java has a Security manager to provide implicit security with the help of Byte code verifier inside JVM.

2. Java has very good websecurity, because Java has provided a separate middleware service in the form of JAAS [Java Authentication And Authorization Service] to provide web security in web applications.
3. Java has very good network security, because Java has provided very good predefined implementations for almost all the network security algorithms as part of its security module.

9. Distributed:

—-----------------

Java is used to prepare two types of applications.

1. Standalone Applications
2. Distributed Applications

If we design and execute any application without using Client-Server arch or without distributing application logic over multiple machines then that Programming language is called “Standalone Applications”. To prepare Standalone Applications J2SE libraries are sufficient.

If we design and execute any application on the basis of Client-Server arch or by distributing application logic over multiple machines then that Programming language is called as Distributed application.

To prepare Distributed applications, JAVA has provided a separate module that is J2EE module.

10. Multi Threaded:

—----------------------

Thread is a flow of execution to perform a particular task.
There are two thread models

1. Single Thread Model
2. Multi Thread Model

Single Thread Model:

1. It Allows only one thread.
2. It follows sequential execution.
3. It takes more execution time
4. Provides Less performance.

Multi Thread Model:

1. It allows more than one thread at a time.
2. It follows parallel execution.
3. It takes less execution time
4. It provides more performance

Java is following the Multi Thread Model to execute its applications. Java is able to provide a very good environment to create and execute more than one thread at a time. Due to this reason JAVA is Multithreaded Programming Language.

11. Interpretive:

Java is both Compilative and interpretive programming language. To convert a program from Source code to bytecode and to check developers mistakes we need compilation and to execute java applications we need an interpreter inside JVM.

12. High Performance:

Java is high performance programming language, because ,

1. Its rich set of features like Portable, Dynamic, Secure, Multithreaded, Distributed,......
2. Due to the components like JIT Compiler inside JVM to boost up JVM Performance.

==

Java Naming Conventions:

—---------------------------

Java is a case sensitive Programming Language, where in java applications there is a separate recognition for lower case letters and uppercase letters.

abc != ABC

To use lower case letters and upper case letters separately we have to use the following conventions.

1. All Java classes names, abstract classes names, interfaces names must be started with an uppercase letter and the subsequent symbols must also be uppercase letters.
   ```
   EX: String , StringBuffer, InputStreamReader
       WindowAdapter, GenericServlet
       Serializable, Externalizable, Comparable
   ```

2. All Java variable names must be started with lower case letters, but the subsequent symbols must be upper case letters.
   ```
   EX: in, out, err
       pageContext, bodyContent
       tempEmpAddr
   ```

3. All Java method names must be started with lower case letters, but the subsequent symbols must be upper case letters.
   EX: concat()

```
        forName()
        getInputStream()
```

4. All java package names must be provided in lower case letters.
   EX: java.io
       java.awt.event
       javax.servlet.jsp.tagext

5. All java constant variables must be provided in Upper case letters.
   EX: MIN_PRIORITY
       MAX_PRIORITY
       NORM_PRIORITY

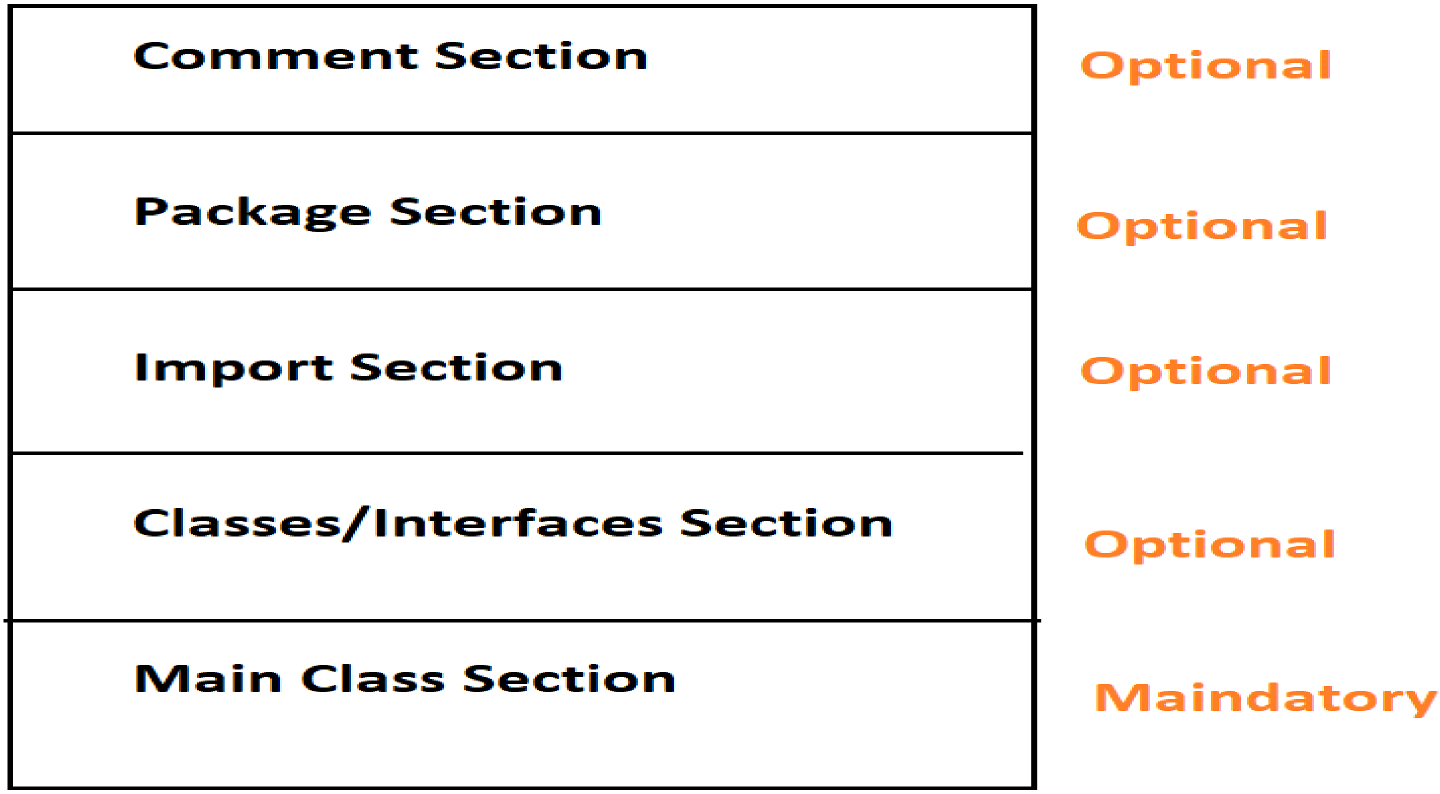
Note: All the above conventions are mandatory while writing predefined libraries, the above conventions are optional while writing user defined libraries, but suggestible.

```
String str = new String(“abc”);

class Employee{
}
```

==

Java Programming Format:

To prepare a basic java program we have to use the following phases.



Comment Section Optional

Package Section Optional

Import Section Optional

Classes/Interfaces Section Optional

Main Class Section Maindatory

Comment Section:

In general, in java application development, it is the convention to provide some description about our implementation before starting code,here the description includes Objective of the implementation, Client details, Author details, Module details,....., here to provide description we will use Comment section, where we will use comments.

Comment is a non-executable part of a Java program.

In Java, there are three types of comments.

1. Single Line comment
2. Multi Line Comment
3. Documentation Comment

Single Line Comment:

—------------------------

It allows the description within a single line.

Syntax:

```
// —---description—--
```

EX:

```
class Calculator{// It represents simple arithmetic operations.
    void add(int i, int j){// It represents addition operation.
        —-----
    }
    void sub(int i, int j){// It represents Subtraction operation
        —-----
    }
    void mul(int i, int j){// It represents Multiplication
operation
        —-----
    }

}
```

Multi Line comments:

—----------------------

It allows the description in more than one line

Syntax:

```
/*
—--
—-- Description—---
—--
*/
```

EX:
--

```
class Calculator{
    /*
        Objective: To perform Arithmetic Addition.
        Parameters: int, int
        Return Type : int
    */
    int add(int i, int j){
        --
        return result;
    }
    /*
        Objective: To perform Arithmetic subtraction.
        Parameters: int, int
        Return Type : int
    */
    int sub(int i, int j){
        --
        return result;
    }

    /*
        Objective: To perform Arithmetic Multiplication.
        Parameters: int, int
        Return Type : int
    */
    int mul(int i, int j){
        --
        return result;
    }
}
```

Documentation Comment:
—---------------------------

It allows the description in more than one page.
It can be used to include the comments in Java documentation of our program.

Syntax:

```
/**
*----
*-----
—
—-
*----
*/
```

In general, we will use documentation comments to prepare API Documentation.

API Documentation[Application Programming Interface]:
It is a document in the form of an html file or a text file or a pdf file or doc file ….it will provide description about the programming elements like variables, methods, classes,.... Which we have used in java applications.

EX: Java has its own API documentation about to give description of all the classes, abstract classes, interfaces, packages,.... Which are provided by JAVA.

If we want to prepare API documentation by using documentation comments then it may take a lot of time, not suggestible, to simplify API documentation creation JAVA has provided a separate tool internally that is “javadoc” tool.

In Java , javadoc is able to provide API documentation in the form of Html files.

```
EX:
—-
Employee.java
—---------------
public class Employee
{
    public int eno;
    public String ename;
    public float esal;
    public String eaddr;

    public Employee(){
    }
    public Employee(int eno){
    }
    public Employee(int eno, String ename){
    }
    public Employee(int eno, String ename, float esal){
    }
    public Employee(int eno, String ename, float esal, String 
eaddr){
    }
    public void add(int eno, String ename, float esal, String
eaddr){
    }
    public void search(int eno){
    }
    public void update(int eno, String ename, float esal, String
eaddr){
    }
    public void delete(int eno){
    }

}
```

On COmmand Prompt:

```
C:\abc>javadoc Employee.java
Loading source file Employee.java...
Constructing Javadoc information...
Building index for all the packages and classes...
Standard Doclet version 17.0.4+11-LTS-179
Building tree for all the packages and classes...
Generating .\Employee.html...
Employee.java:1: warning: no comment
public class Employee
       ^
Employee.java:6: warning: no comment
        public String eaddr;
                      ^
Employee.java:4: warning: no comment
        public String ename;
                      ^
Employee.java:3: warning: no comment
```

```
        public int eno;
                   ^
Employee.java:5: warning: no comment
        public float esal;
                     ^
Employee.java:8: warning: no comment
        public Employee(){
               ^
Employee.java:10: warning: no comment
        public Employee(int eno){
               ^
Employee.java:12: warning: no comment
        public Employee(int eno, String ename){
               ^
Employee.java:14: warning: no comment
        public Employee(int eno, String ename, float esal){
               ^
Employee.java:16: warning: no comment
        public Employee(int eno, String ename, float esal, String
eaddr){
               ^
Employee.java:19: warning: no comment
        public void add(int eno, String ename, float esal, String
eaddr){
                    ^
Employee.java:25: warning: no comment
        public void delete(int eno){
                    ^
Employee.java:21: warning: no comment
        public void search(int eno){
                    ^
Employee.java:23: warning: no comment
```

```
        public void update(int eno, String ename, float esal,
String eaddr){
                    ^
Generating .\package-summary.html...
Generating .\package-tree.html...
Generating .\overview-tree.html...
Building index for all classes...
Generating .\allclasses-index.html...
Generating .\allpackages-index.html...
Generating .\index-all.html...
Generating .\index.html...
Generating .\help-doc.html...
14 warnings
```

In general, single line comments and multi line comments are not included in API documentation which is generated from javadoc tool, Only Documentation Comments allows to represent data in API Documentation which is generated from javadoc tool.

In Java applications , to represent metadata[Data About the data] or description , JDK1.5 version has provided an alternative that is “Annotations”.

Q)IN Java applications, to provide description we have already comments then what the requirement is to use Annotations?
—--

Ans:
—--

If we provide description along with comments in java applications then comments and its description exist up to compilation of the program , not possible up to .class file and up to RUNTIME of the application.

As per the requirements like to simplify debugging process , to simplify Testing process , if the application is server side application , if the application is database related application or any frameworks related application then we have to provide description up to .java file, up to .class file and up to RUNTIME of our application.

In the above context, to bring description up to .java file, up to .class file and up to RUNTIME of our applications we have to use “Annotations”.

Q)To bring description up to RUNTIME we already have XML documents then what is the requirement to use Annotations?
—---
Ans:
—--
To provide description up to Runtime if we use XML documents then we have to get the following problems.

1. Java developers must learn XML tech .
2. We must check every time whether the provided XML document is in a well formed format or not.
3. Every time, developers must check whether the provided XML document must be available at the right locations or not.
4. Every time we have to check whether we are using the right parsing mechanism or not to read data from an XML document.

To overcome all the above problems we need a Java alternative that is “Annotations”.

Note: In JAVA/J2EE technologies annotations are the best alternative for XML documents.

XML Based Tech		Annotation Based Tech
1)Up to JDK1.4	--------------------->	JDK1.5
2)JDBC3.x	--------------------->	JDBC4.x
3)Servlets2.5	--------------------->	Servlets3.x
4)EJBs2.x	--------------------->	EJBs3.x
5)Struts1.x	--------------------->	Struts2.x
6)JSF1.x	--------------------->	JSF2.x
7) Spring2.4	--------------------->	Spring2.5

Package Section:

Def:

1. Package is the collection of related classes and interfaces as a single unit.
2. Package is a Folder containing .class files representing related classes and interfaces.

java.lang

1. String
2. StringBuffer

----

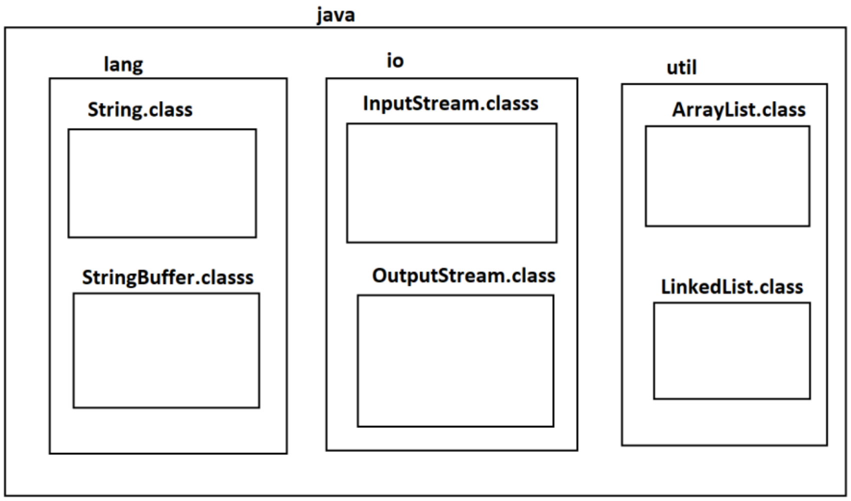
java.io

1. InputStream
2. OutputStream

-----

java.util

1. ArrayList
2. LinkedList



Advantages:

1. Modularity
2. Abstraction
3. Security
4. Shareability
5. Reusability

Types:

1. Predefined packages
2. User Defined packages

Predefined packages are the packages defined by JAVA programming language and provided along with Java software.

User defined Packages are the packages which are defined by the developers as per their application requirements.

To declare a user defined package in java applications we will use the “package” statement.

Syntax: package packageName;

Where packageName may be a single name or the combination of parent package name and child package name with . seperator.

EX:

```
package p1;
package p1.p2.p3;
```

Conditions:

If we want to use packages in Java applications we have to use the following conditions.

1. Package Statement must be the first statement in the Java file after the comment section.
2. Package name must be unique, it must not be shareable and it must not be duplicated.

Q)Is it possible to provide more than one package declaration statement in a single java file?

—---

Ans:

—---

No, it is not possible to provide more than one package declaration statement in a single java file, because, in Java files, a package statement must be the first statement.

EX: abc.java

—------------

```
package p1; —---> Valid
package p2; —---> Invalid
Package p3; —---> Invalid
```

To provide uniqueness in package names JAVA has provided a suggestion to the developers to include our company domain name in reverse in package name.

EX:

```
package com.durgasoft.icici.transactions.deposit;
com.durgasoft —---> Company Domain name in reverse.
icici —---------------> Client name / project name
transactions —------> Module name
deposit —------------> Sub module name
```

Import Section:

The main purpose of the import statement is to make available all classes and interfaces of a particular package in to the present java file.

Syntax:

1. import packageName.*;

   It is able to import all classes and interfaces from the specified package in to the present java file.

   ```
   EX: import java.io.*;
       import java.util.*;
       import java.sql.*;
   ```

2. import packageName.memberName;

   It is able to import only the specified class or interface from the specified package into the present java file.

   ```
   EX  import java.io.BufferedReader;
       import java.util.ArrayList;
       import java.sql.Connection;
   ```

   Note: In java files, we are able to provide at most one package declaration statement , but we are able to provide any number of import statements.

   ```
   EX: abc.java
   —-------------
   package p1; —---> Valid
   package p2;-----> Invalid
   package p3;-----> Invalid
   import java.sql.*; —--> Valid
   import java.util.*; —--> Valid
   ```

```
    import java.io.*; ––---> Valid
```

Q)Is it possible to use classes and interfaces of a particular package without importing the respective package?
––

Ans:
––--

Yes, it is possible to use classes and interfaces of a particular package without importing the respective package, but, we must use the “Fully Qualified Name” of the respective class or interface.

Note: If we provide classes and interfaces along with their respective package name is called a “Fully Qualified Name”.

```
EX: java.io.BufferedReader
    java.util.ArrayList
    java.sql.Connection
```

Note: In Java applications it is always suggestible to use import statements instead of fully qualified names, here in import statements it is always suggestible to use import statements with out * notation, that is, individual classes or interfaces importing is suggestible.

EX: A Java program without import statement:

```
java.io.BufferedReader br = new java.io.BufferedReader(new
java.io.InputStreamReader(System.in));
```

EX: A Java program with import statement with * notation:

```
import java.io.*;
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

EX: A Java program with import statement with out * notation:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
BufferedReader br = new BufferedReader(new
InputStreamReader(System.in));
```

Classes / interfaces Section:

The main intention of class or interface is to represent all real world entities in Java applications.
EX: Student, Employe, Product, Customer, Account,....

In Java applications we are able to write any no of classes and interfaces as per the application requirement.

Main Class Section:

If any class includes the main() method then that class is called “Main Class”.

The main intention of the main() method is,

1. To manage the application logic which is executed by JVM automatically we need the main() method.
2. To define starting point and ending point to the application execution we need the main() method.

Syntax:

```
class Test{
    public static void main(String[] args){
        —-Application Logic—----
    }
}
```

===

Steps to prepare First Java Application:
======================================

1. Download and install Java Software.
2. Download and install Java Editor
3. Open Java editor and Write Java program
4. Save Java File
5. Compile Java file
6. Execute Java Application

Download and install Java Software:
After installation of the Java software we have to set the “path” env variable to “C:\Java\jdkVersion\bin” in order to use java software through command prompt.

There are two ways to set the path environment.

1. Temporary Setup

2.Permanent Setup

Temporary Setup:
Use the following command on command prompt.

```
set path=C:\Java\jdk1.8.0_202\bin;
```

If we provide a temporary setup for the path environment variable then this setup is valid up to the present command prompt, it not applicable to the other command prompts.

If we want to apply path environment variable setup to all the command prompts then we have to set path environment variable in Permanent setup.

Q)Is it possible to install more than one Java software in a single computer?
—--
--
Ans:
—--

Yes, we can install more than one Java software in our computer, but, Which java software we set up in the path environment variable that java software is available to use.
EX:

```
C:\Users\Administrator>set path=C:\Java\jdk1.6.0_45\bin;

C:\Users\Administrator>javac -version
javac 1.6.0_45

C:\Users\Administrator>set path=C:\Java\jdk1.7.0_80\bin;
```

```
C:\Users\Administrator>javac -version
javac 1.7.0_80

C:\Users\Administrator>set path=C:\Java\jdk1.8.0_202\bin;

C:\Users\Administrator>javac -version
javac 1.8.0_202

C:\Users\Administrator>
```

Q)If we set all three versions of java softwares to a single path environment variable then which java software version is available for us to use in command prompt?
—---

Ans:
—--

In the “path” environment variable which java version we set as first one in the order that java version is available for us to use in command prompt.

EX:

```
C:\Users\Administrator>set
path=C:\Java\jdk1.6.0_45\bin;C:\Java\jdk1.7.0_80\bin;C:\Java\jdk1.8
.0_202\bin;

C:\Users\Administrator>javac -version
javac 1.6.0_45

C:\Users\Administrator>set
path=C:\Java\jdk1.7.0_80\bin;C:\Java\jdk1.8.0_202\bin;C:\Java\jdk1.
6.0_45\bin;
```

```
C:\Users\Administrator>javac -version
javac 1.7.0_80

C:\Users\Administrator>set
path=C:\Java\jdk1.8.0_202\bin;C:\Java\jdk1.6.0_45\bin;C:\Java\jdk1.
7.0_80\bin;

C:\Users\Administrator>javac -version
javac 1.8.0_202
```

As per the requirement, if we want to execute a java application in multiple java versions then we have to use “set path=....\bin” command on command prompt every time, it will increase burden to the developers.

```
D:\java6>set path=C:\Java\jdk1.6.0_45\bin;
D:\java6>javac Test.java
D:\java6>java Test
    X-Results
D:\Java6>set path=C:\Java\jdk1.7.0_80\bin;
D:\java6>javac Test.java
D:\java6>java Test
    Y-Results
D:\java6>set path=C:\Java\jdk1.8.0_202\bin;
D:\java6>javac Test.java
D:\java6>java Test
    Z-Results
```

In Java , if we want to simplify switching java from one version to another version we will use “batch files”.

Batch file is like a text file but the file extension must be “.bat”, it will be executed by the Operating system when we use its name on command prompt.

EX:

```
java6.bat
—------------
set path=C:\Java\jdk1.6.0_45\bin;

java7.bat
—----------
set path=C:\Java\jdk1.7.0_80\bin;



java8.bat
—----------
set path=C:\Java\jdk1.8.0_202\bin;
```

On command prompt:

```
D:\Java6>java6
Java6 setup…..
D:\Java6>java7
Java7 Setup…..
D:\Java6>java8
Java8 Setup…..
```

Java Editor:

It is a software, it able to provide a very good environment to write Java programs and to save java programs in to our computer.

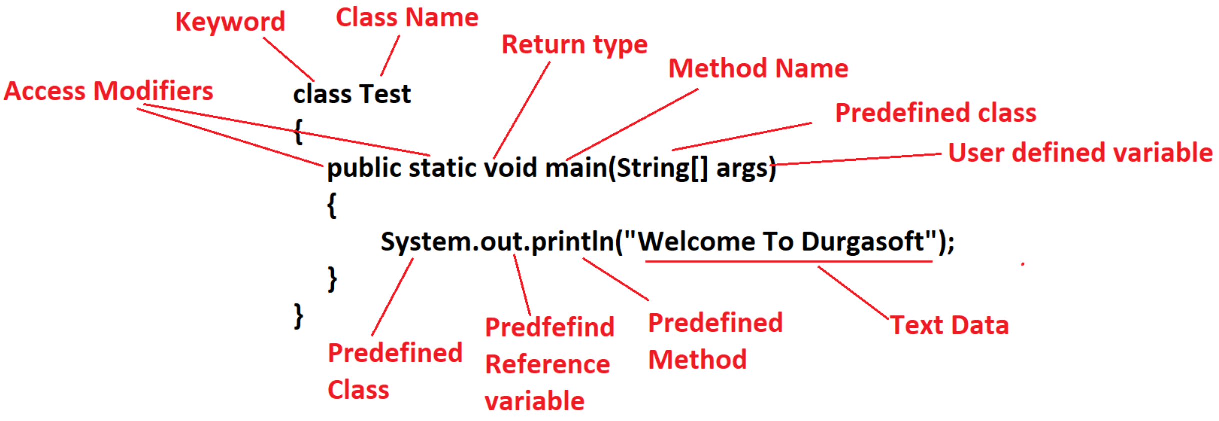
EX: Notepad, Notepad++, Editplus, ……

In Java Real time application development , it is not suggestible to use Editors, Editors are suggestible up to learning process only, but in Real time application development we must use IDEs.

IDE: Integrated Development Environment.
EX: Eclipse, Netbeans, Intellij Idea,...

Write Java program:
To write java applications at the basic level we have to use the following programming elements.

1. Declare a class
2. Declare main() method
3. Inside the main() method write application logic.



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

Save Java File:

To save Java files , we have to use the following rules and regulations.

1. If the present java file contains public class or public abstract class or public interface then the respective java file name must be the public class name or public abstract class name or public interface name, if we violate this rule then the compiler will raise an error.
2. If no element[Class / abstract class / interface] is public in the present java file then it is possible to save a java file with any name like abc.java or xyz.java, in this context, java has given a suggestion like to save java file with main class name.

Q)Is it possible to provide more than one public class in a Single Java file?

—--

Ans:

—----

No, it is not possible to provide more than one public class in single java file, as per the java rules and regulations if any class is public in java file then we must save that java file with the public class name , in the case if we provide more than one class as public then we must save that java file with more than one name , it is not possible in all the operating systems.

Compile Java File:

The main intention to perform compilation is

1. To translate a java program from Source code to bytecode.
2. To check developers' mistakes before execution.

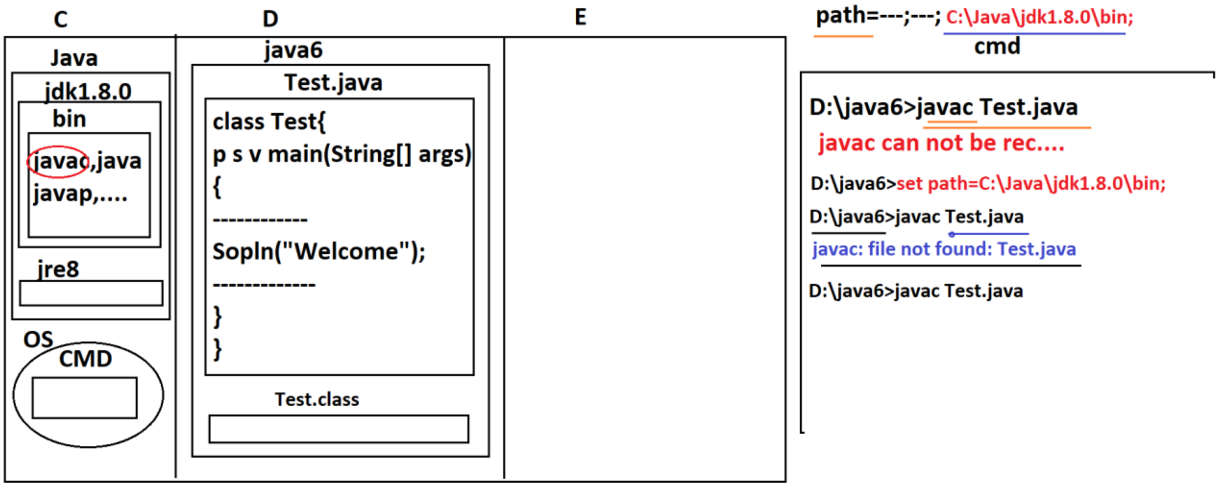
To perform compilation , Java software has provided a command that is “javac”.

```
D:\Java6>javac Test.java
```

When we use the above command on command prompt, the Operating System will perform the following actions.

1. The Operating System will take a javac command from the command prompt.
2. The Operating System will search for javac command in its internal commands list and at the locations referred by “path” environment.

3. If javac is not identified at both the locations then the Operating system will provide the following message on command.
   ```
   'javac' is not recognized as an internal or external command,
   operable program or batch file.
   ```
4. To make available all java commands like javac, java,javap,... to the operating system we have to set the “path” environment variable to “C:\Java\jdk1.8.0_202\bin”.
   ```
   D:\java6pm>set path=C:\Java\jdk1.8.0_202\bin;
   ```
5. When the Operating System identifies javac command at path environment variable referred locations , Operating System will execute javac command, with this, Java compiler software will be activated and Java compiler software will perform the following actions.
   a. Java Compiler will take the provided java file from the command prompt.
   b. Java Compiler will search for the java file at the current location.
   c. If the required java file is not identified at the current location then compiler will raise an error like below.
      ```
      javac: file not found: Test.java
      ```
   d. If the required java file is identified at the current location then compiler will start compilation right from starting point of the file to ending point of the file.
   e. If any error is identified in the java file then compiler will provide error message on command prompt.
   f. If no error is identified in the java file then the compiler will finishes compilation and generate .class files.



```
class Test{
p s v main(String[] args)
{
------------
Sopln("Welcome");
-------------
}
}
```

```
D:\java6>javac Test.java
 javac can not be rec....
D:\java6>set path=C:\Java\jdk1.8.0\bin;
D:\java6>javac Test.java
javac: file not found: Test.java
D:\java6>javac Test.java
```

Generating the number of .class files is not depending on the number of java files which we compiled, it is completely dependending on the following elements which we provided in java file.

1. Number classes
2. Number of abstract classes
3. Number of interfaces
4. Numbers of enums
5. Number of inner classes

EX:

```
Test.java
enum E{// E.class
}
interface I{// I.class
}
abstract class A{// A.class
}
class B{// B.class
     class C{// B$C.class
     }
}
class Test{// Test.class
     public static void main(String[] args){
          System.out.println("Welcome To Java");
     }
}
```

On COmmand Prompt:

```
D:\java6pm>javac Test.java
D:\java6pm>dir
 Volume in drive D is Data
 Volume Serial Number is 68E9-29EB

 Directory of D:\java6pm

13-12-2022  18:49    <DIR>          .
13-12-2022  18:49    <DIR>          ..
13-12-2022  18:49               179 A.class
13-12-2022  18:49               270 B$C.class
13-12-2022  18:49               223 B.class
13-12-2022  18:44            65,929 Compilation.png
13-12-2022  18:49               611 E.class
13-12-2022  18:49                86 I.class
10-12-2022  18:43                35 java6.bat
10-12-2022  18:44                35 java7.bat
10-12-2022  18:45                36 java8.bat
13-12-2022  18:49               417 Test.class
13-12-2022  18:48               246 Test.java
13-12-2022  18:47               181 Test.java.bak
              12 File(s)         68,248 bytes
               2 Dir(s)  351,419,473,920 bytes free
```

If we want to compile a java file from the current location and if we want to send the generated .class files to some other location then we have to use ‘-d’ option along with javac command.

```
javac -d targetLocation FileName.java
```

EX:

```
D:\java6>javac -d E:\abc Test.java
```

EX:

```
D:\java6pm\Test.java
—------------------------
enum E{// E.class
}
interface I{// I.class
}
abstract class A{// A.class
}
class B{// B.class
     class C{// B$C.class
     }
}
class Test{// Test.class
     public static void main(String[] args){
          System.out.println("Welcome To Java");
     }
}
```

On COmmand Prompt:

```
D:\java6pm>javac -d E:\abc Test.java

D:\java6pm>dir
 Volume in drive D is Data
 Volume Serial Number is 68E9-29EB

 Directory of D:\java6pm

13-12-2022  18:59    <DIR>          .
13-12-2022  18:59    <DIR>          ..
13-12-2022  18:44            65,929 Compilation.png
10-12-2022  18:43                35 java6.bat
10-12-2022  18:44                35 java7.bat
10-12-2022  18:45                36 java8.bat
13-12-2022  19:00               246 Test.java
13-12-2022  18:47               181 Test.java.bak
               6 File(s)         66,462 bytes
               2 Dir(s)  351,419,473,920 bytes free

D:\java6pm>e:

E:\>cd abc

E:\abc>dir
 Volume in drive E is New Volume
 Volume Serial Number is 7E96-6D63

 Directory of E:\abc

13-12-2022  19:01    <DIR>          .
13-12-2022  19:01    <DIR>          ..
13-12-2022  19:01               179 A.class
13-12-2022  19:01               270 B$C.class
13-12-2022  19:01               223 B.class
13-12-2022  19:01               611 E.class
13-12-2022  19:01                86 I.class
13-12-2022  19:01               417 Test.class
               6 File(s)          1,786 bytes
               2 Dir(s)  304,148,496,384 bytes free
```

In Java file, if we provide a package declaration statement then we have to compile that java file with '-d' option in order to create directory structure w.r.t the package name at target location.

EX:

```
D:\java6pm\Test.java
package com.durgasoft.core;
enum E{// E.class
}
interface I{// I.class
}
abstract class A{// A.class
}
class B{// B.class
     class C{// B$C.class
     }
}
class Test{// Test.class
     public static void main(String[] args){
          System.out.println("Welcome To Java");
     }
}
```

On COmmand Prompt:

```
D:\java6pm>javac -d E:\abc Test.java

D:\java6pm>e:

E:\abc>dir
 Volume in drive E is New Volume
 Volume Serial Number is 7E96-6D63

 Directory of E:\abc

13-12-2022  19:12    <DIR>          .
13-12-2022  19:12    <DIR>          ..
13-12-2022  19:12    <DIR>          com
               0 File(s)              0 bytes
               3 Dir(s)  304,148,492,288 bytes free

E:\abc>cd com

E:\abc\com>dir
 Volume in drive E is New Volume
 Volume Serial Number is 7E96-6D63

 Directory of E:\abc\com

13-12-2022  19:12    <DIR>          .
13-12-2022  19:12    <DIR>          ..
13-12-2022  19:12    <DIR>          durgasoft
```

```
               0 File(s)              0 bytes
               3 Dir(s)  304,148,492,288 bytes free

E:\abc\com>cd durgasoft

E:\abc\com\durgasoft>dir
 Volume in drive E is New Volume
 Volume Serial Number is 7E96-6D63

 Directory of E:\abc\com\durgasoft

13-12-2022  19:12    <DIR>          .
13-12-2022  19:12    <DIR>          ..
13-12-2022  19:12    <DIR>          core
               0 File(s)              0 bytes
               3 Dir(s)  304,148,492,288 bytes free

E:\abc\com\durgasoft>cd core

E:\abc\com\durgasoft\core>dir
 Volume in drive E is New Volume
 Volume Serial Number is 7E96-6D63

 Directory of E:\abc\com\durgasoft\core

13-12-2022  19:12    <DIR>          .
13-12-2022  19:12    <DIR>          ..
13-12-2022  19:12               198 A.class
13-12-2022  19:12               346 B$C.class
13-12-2022  19:12               261 B.class
13-12-2022  19:12               706 E.class
13-12-2022  19:12               105 I.class
13-12-2022  19:12               436 Test.class
               6 File(s)          2,052 bytes
               2 Dir(s)  304,148,492,288 bytes free
```

If we want to generate folder structure w.r.t the package name at the current location from where we are compiling the java file then we have to use ‘.’ in place of target location.

EX:

```
D:\java6pm\Test.java
package com.durgasoft.core;
enum E{// E.class
}
interface I{// I.class
}
abstract class A{// A.class
}
class B{// B.class
     class C{// B$C.class
     }
}
class Test{// Test.class
     public static void main(String[] args){
          System.out.println("Welcome To Java");
     }
}
```

On COmmand Prompt:

```
D:\java6pm>javac -d  . Test.java

D:\java6pm>cd com

D:\java6pm\com>cd durgasoft

D:\java6pm\com\durgasoft>cd core

D:\java6pm\com\durgasoft\core>dir
 Volume in drive D is Data
 Volume Serial Number is 68E9-29EB

 Directory of D:\java6pm\com\durgasoft\core
```

```
13-12-2022  19:18    <DIR>          .
13-12-2022  19:18    <DIR>          ..
13-12-2022  19:18               198 A.class
13-12-2022  19:18               346 B$C.class
13-12-2022  19:18               261 B.class
13-12-2022  19:18               706 E.class
13-12-2022  19:18               105 I.class
13-12-2022  19:18               436 Test.class
               6 File(s)          2,052 bytes
               2 Dir(s)  351,419,473,920 bytes free
```

Q)Is it possible to compile more than one java file with a single java command at a time?

—---

Ans:

—---

Yes, It is possible to compile more than one java file by using single javac command, but we have to use ‘*’ notation in java file names.

Case#1: To compile all java files existed at current location

```
D:\java6>javac *.java
```

Case#2: To compile all java files which are prefixed with Employee

```
D:\java6>javac Employee*.java
```

Case#3: To compile all java files which are suffixed with Names

```
D:\java6>javac *Names.java
```

Case#4: To compile all java files which includes Employee

```
D:\java6>javac *Employee*.java
```

Execute Java Application:

—---------------------------

To execute a Java program , Java has provided a predefined command in the form of “java”.

```
java MainClassName
```

EX:

```
D:\java6>java Test
```

If we use the above command on command prompt then the Operating System will perform the following actions.

1. Operating SYstem will take java command from command prompt and search for it in its predefined commands list and at the locations referred by “path” environment variable.
2. When the Operating system identifies a Java command at “C:\Java\jdk1.8.0\bin” location through the path environment variable then the Operating System will execute the Java command.
3. When the Operating System executed java command , automatically JVM software will be activated and JVM will perform the following actions.
    a. JVM will take main class name from the command prompt which we provided along with java command.
    b. JVM will search for the main class .class file at the following three locations.
        1. At current location
        2. At Java predefined library
        3. At the locations referred by “classpath” environment variable.

    c. If the required main class .class file is not available at all the above locations then JVM will provide the following exception or error.

    ```
    Java6: java.lang.NoClassDefFoundError: Test
    Java7: Error: Could not find or load main class Test
    ```

    Note: If the main class .class file is available at a different location , not at all the above locations then we have to make available that .class file to the

JVM, for this, we have to set “classpath” environment variable to the location where main class .class file is existed.

```
set classpath=E:\abc;
```

D. If JVM identifies main class .class file at either of the above locations then JVM will load the main class bytecode to the memory by using “Class Loaders” Internally.

E. After loading main class bytecode to the memory , JVM will search for the main() method in the loaded main class.

f. If main method is not identified in the loaded main class then JVM will provide the following Exception /Error

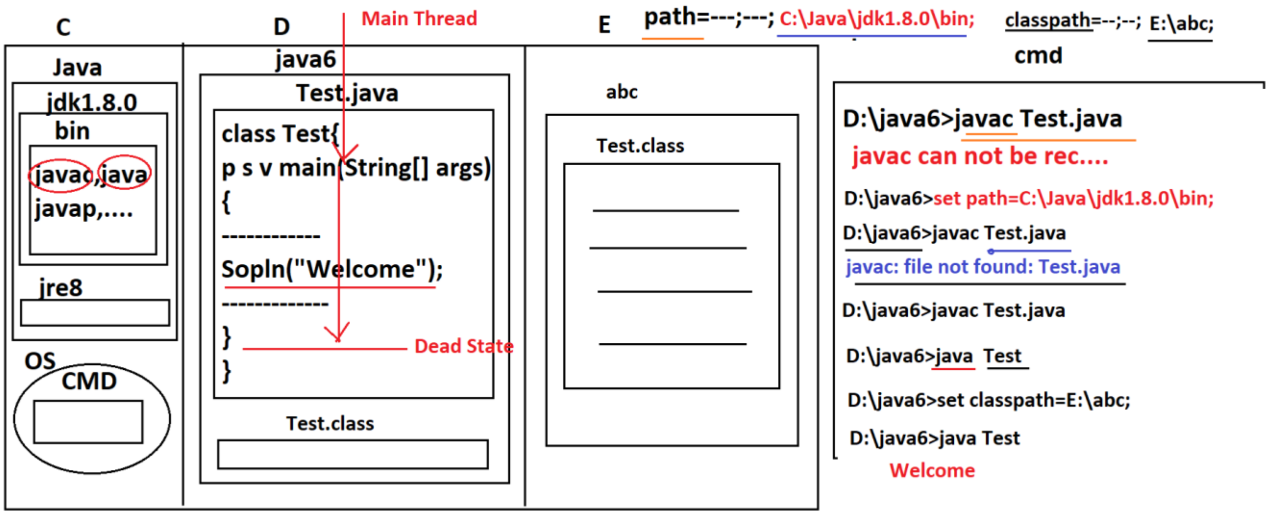
```
Java6: java.lang.NoSuchMethodError: main
Java7: Error: Main method not found in class Test, please define the main
    method as:   public static void main(String[] args)
```

G. If the main() method is existed in the loaded main class then JVM will create a thread to execute main() method named “Main Thread”.

H. Main Thread will start application execution right from starting point of the main() method to the ending point of the main() method.

I. When Main Thread reaches the ending point of main() method , Main Thread will come to Dead State.

J. When Main Thread is in Dead state, JVM will stop all the internal services which are assigned to the Java program execution and JVM will go to Shut down mode.



```
class Test{
p s v main(String[] args)
{
------------
Sopln("Welcome");
-------------
}
}
```

```
D:\java6>javac Test.java
 javac can not be rec....
D:\java6>set path=C:\Java\jdk1.8.0\bin;
D:\java6>javac Test.java
javac: file not found: Test.java
D:\java6>javac Test.java
D:\java6>java Test
D:\java6>set classpath=E:\abc;
D:\java6>java Test
        Welcome
```

Q)What is the difference between “path” environment variable and “classpath” environment variable?

—---

Ans:

—---

“path” environment variable can be used to provide all java commands locations information to the Operating System.

```
     set path=C:\Java\jdk1.8.0\bin;
```

“classpath” environment variable can be used to provide .class files location information to the compiler and JVM.

```
     set classpath=E:\abc;
```

Note: In general, we will set “path” environment variable permanently, we will not set “classpath” environment variable permanently, because “path” environment variable location is fixed, by “classpath” environment variable location is not fixed, it varies from application to application.

Notepad++:
—------------

1. Download and Install Notepad++ Software.
2. Open File and save with .java extension.
3. Write Java program
4. Compile Java file
5. Execute Java Application

Sublime Text:
—-------------

1. Download and Install Sublimetext
2. Open File and save with .java extension.
3. Write Java program
4. Compile java file
5. Execute Java Application

Language Fundamentals:
—--------------------------

If we want to design or develop any Java application we need some resources or building blocks which must be provided by the Programming language called “Language Fundamentals”.

To prepare Java applications, JAVA has provided the following building Blocks.

1. Tokens
2. Data Types
3. Type Casting
4. Java Statements
5. Arrays

Tokens:
—--------

Lexeme: The individual logical unit in Java programming is called “Lexeme”.

Token: The collection of Lexemes comes under a particular group called
“Token”.

EX:

```
int a = b + c * d;
No of Lexemes: int, a, =, b, +, c, *, d, ;     :  9 Lexemes

int —-----> Data Type
a, b, c , d —----> Variables / Identifiers
=, +, * —------> Operators
; —-----------> Special Symbol / Terminator
—-----------------------------
No of Tokens  : 4 types of Tokens
```

To prepare Java applications, JAVA has provided the following Tokens.

1. Identifiers
2. Literals
3. Keywords / Reserved Words
4. Operators

Identifiers:
—------------

Identifier is a name assigned to the programming elements like variables , methods, constructors, classes, interfaces, packages,.....

Java has provided the following rules and regulations to give names to the programming elements.

1. All Java identifiers must not be started with a number, identifiers may start with an alphabet, _ symbol , $ symbol, but the subsequent symbols may be a number, an alphabet, _ symbol, $ symbol.

```
EX:
int eno = 111; ---------------------> Valid
String 9ename = “Durga”; -------> Invalid
float _esal = 50000.0f; ---------> Valid
float $esal = 500.0; -------------> Valid
String emp_Id = “E-111”; -------> Valid
float esalIn$ = 500.0f;  --------> Valid
```

2. Identifiers do not have spaces in the middle.

```
EX: int emp No = 111; -------> Invalid
    int empNo = 111;  ------> Valid
    String empAddr = “Hyd”; ---> Valid
    String emp Addr = “Hyd”; ---> Invalid
    String tempEmpAddr = “Hyd”; -> Valid
    String temp Emp Addr = “Hyd”;----> Invalid
```

3. In Java applications, all operators are not allowed in the identifiers.

```
EX:  int empNo = 111; --------------> Valid
     int emp+No = 111; ---------------> Invalid
     String emp-Name = “Durga”; ---> Invalid
     float emp_Sal = 50000.0f; ------> Valid
     String emp*Addr = “Hyd”; ------> INvalid
```

4. All Java identifiers are not allowing all special symbols except _ symbol and $ symbol.

```
EX:  int emp.No = 111; ------------------> Invalid.
     String emp@Hyd = “Durga”; -----> Invalid
```

```
        float empSal$ = 5000.0; ---------> Valid
        String #eaddr = “Hyd”; ----------> Invalid
        String emp-Qual = “BTech”;  -----> Invalid
```

5. All Java identifiers are not allowing keywords

```
    EX: int a = 10; -----------------> Valid
        int break = 20; -----------> Invalid
        float esal = 50000.0f; ----> Valid
        float for = 50000.0f; ------> Invalid
```

6. All Java identifiers are not allowing primitive data types.

```
    EX: int a = 10; -------------------> Valid
        int long = 10; ----------------> Invalid
        String int = “abc”; ----------> Invalid
        double float = 22.22; -------> Invalid
        float f = 22.22f; ------------> Valid
```

7. All Java identifiers are allowing predefined classes , abstract classes and interfaces names.

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

OP:
10

EX:
class Test{
    public static void main(String[] args){
        String String = "String";
        System.out.println(String);

    }
}
```

OP:

```
String
```

EX:

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

Status: Compilation Error.

Reason: If we declare any predefined class name as an integer variable then in the remaining part of the program we must use that predefined class name as integer variable only , not possible to use that predefined class name as like its original class name, if we use that name as like its original class name then compiler will raise an error.

In the above context, if we want to use the predefined class as like its original class name then we must use “Fully Qualified Name”.

EX:

——

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

To provide identifiers in Java applications , Java has provided the following Suggestions.

1. In Java, it is suggestible to provide all identifiers with a particular meaning.
   String xxx = “abc123”; —---> Not Suggestible
   String accNo = “abc123”; —> Suggestible
2. In Java, there is no length restriction for the identifiers , we can write identifiers with any length , but, it is suggestible to manage length of the identifiers around 10 symbols.
   EX:
   String temporaryemployeeaddress = “Hyd”; —-> Not Suggestible
   String tempEmpAddr = “Hytd”; —--------------> Suggestible

3. In Java identifiers, if we have multiple words then it is suggestible to separate multiple words with the special notations like _ symbol
   EX:
   String tempEmpAddr = “Hyd”; —-----> Not Suggestible
   String temp_Emp_Addr = “Hyd”;-----> Suggestible

Literals:
—--------

Literal is a constant assigned to the variables.

int a = 10;

int —---> Data Type

a —-----> Variable/Identifier

= —-----> Operator

10 —----> Constant[Literal]

; —-------> Terminator / Special Symbol

To prepare Java applications, JAVA has provided the following types of Literals.

1. Integral / Integer Literals

   byte, short, int, long —--> 10, 20, 34,......

   char —-> ‘a’, ‘b’, ‘\t’, ‘\n’

   Note: All Long literals must be provided with the suffix values with ‘l’ or ‘L’.

2. Floating point Literals

   float —----> 22.22f, 456.234f, 876.234F,....

   Double —--> 222.22, 345.345d, 567.345D,...

   Note: ALl float literals must be suffixed with either ‘f’ or ‘F’.

   Note: All the Double literals must be provided with the following suffixes.

   1. No Suffix
   2. ‘D’ suffix
   3. ‘d’ suffix

3. Boolean Literals:

   boolean —-> true, false

4. String Literals:

String —--> “abc”, “xyz”, “12334”, “true”, “22.22f”, “234.345”,

Note: Only String data type is able to represent all the types of data including numbers, characters,......

Note: To improve Readability in literals JAVA 1.7 version has provided a flexibility to include ‘_’ symbols in the literals.

EX:

```
Up to JAVA6: double d = 123456789.2345;
From JAVA7: double d = 12_34_56_789.2345;
```

When we compile the above instruction, Compiler will remove all the ’_’ symbols from the number, Compiler and JVM will process that number without ‘_’ symbols and JVM will provide output without ‘_’ symbols only.

Number Systems in Java:

—---------------------------

In all programming languaGES , to represent numbers we have to follow some standards they are “Number Systems”.

In general, there are four types of Number Systems.

1. Binary Number System
2. Octal Number System
3. Decimal Number System
4. Hexa-Decimal Number System

All the number Systems are allowed in Java, but the default number system is “Decimal Number System”.

1. Binary Number System:

   Its base value is ‘2’, that is, 2 input alphabet.

This number system allows only 0’s, 1’s symbols.
To represent numbers in a binary number system we must use either ‘0b’ or ‘0B’ prefix for the number.

EX:
–––

```
    int a = 10; –––––––––––> It is not a binary number, it is a
decimal number.
    int b = 0b1010; ––––> Valid
    int c = 0B1100; ––––> Valid
    int d = 0b1020; ––––> Invalid
```

Note: Binary Number System was introduced in JAVA in its JDk1.7 version.

EX:
––

```
class Test{
 public static void main(String[] args){
    int a = 0b10;// 0*2p0 + 1*2p1 ==> 0 + 2 = 2
    System.out.println(a);// 2
 }
}
```

On Command PRompt:

```
D:\java6pm>javac -version
javac 1.7.0_80

D:\java6pm>javac Test.java

D:\java6pm>java Test
2

D:\java6pm>java6.bat
```

```
D:\java6pm>set path=C:\Java\jdk1.6.0_45\bin;

D:\java6pm>javac Test.java
Test.java:3: ';' expected
        int a = 0b10;// 0*2p0 + 1*2p1 ==> 0 + 2 = 2
                 ^
1 error
```

2.Octal Number System:

Its Base value is 8, that is 8 numbers of input alphabet.

It allows the numbers like 0,1,2,3,4,5,6 and 7 to represent numbers in the Octal number system.

To represent Octal numbers in Java applications we have to use ‘0’ [zero] as a prefix.

EX:

int a = 10; ---------> It is not an Octal number, it is a decimal number

int b = 012345; --> Valid

int c =O34567; --> INvalid, number must be prefixed with 0 but not O.

int d = 0678; ----> Invalid, 8 not allowed in octal number system.

EX:

```
class Test{
 public static void main(String[] args){
    int a = 010;// 0*8p0 + 1*8p1 ==> 0 + 8 ==> 8
    System.out.println(a);
 }
}
```

Q)Font the number system from the following instruction.

```
int a = 0;
```

Ans: Decimal Number System

Q)Font the number system from the following instruction.

```
int a = 00;
```

Ans: Octal Number System

3. Decimal Number System:

Its Base value is 10, that is 10 symbols are allowed to represent numbers.

It uses the numbers like 0,1,2,3,4,5,6,7,8 and 9.

No prefix value exists for the decimal numbers.

4. Hexa-Decimal Number System:

Its Base value is 16, that is , it is using 16 input alphabet.

It is using the numbers like 0,1,2,3,4,5,6,7,8,9,a,b,c,d,e and f.

It uses either ‘0x’ or ‘0X’ as a prefix value for the numbers.

EX:

```
int a = 10; —-------> It is  not a Hexa-decimal number, it is a
Decimal num.
int b = 0x123456;--> Valid
int c = 0X789abc; → valid
int d = 0xdefg; —---> Invalid
```

EX:

```
class Test{
 public static void main(String[] args){
    int a = 0x10;// 0*16p0 + 1*16p1 ==> 0 + 16 ==> 16
    System.out.println(a);// 16
```

```
     }
    }
```

Note: if we provide any number in any number system in Java applications then Compiler will convert the provided number from the provided number system to decimal number system , where compiler and JVM will process that number as like decimal number only and JVM will provide the outputs in the form of Decimal numbers only.

EX:
—-

```
class Test{
 public static void main(String[] args){
    int a = 0b1010;// 10
    int b = 0B0010;// 2
    System.out.println(a+b);// 12
    System.out.println(a-b);// 8
    System.out.println(a*b);// 20
    System.out.println(a/b);// 5
 }
}
```

OP:
—-

```
12
8
20
5
```

EX:
—-

```
class Test{
 public static void main(String[] args){
    int a = 0b1010;
    int b = 03;
    System.out.println(a+b);
 }
}
```

OP:

```
13
```

EX:
—-

```
class Test{
 public static void main(String[] args){
    int a = 02;
    int b = 0xa;
    System.out.println(a+b);
 }
}
```

OP:

```
12
```

Keywords and Reserved Words:
—----------------------------------

If any predefined word has word recognition and internal functionality then that predefined word is called “Keyword”.

If any predefined word has only word recognition but does not have internal functionality then that predefined word is called “Reserved Word”.
EX: goto, const

Note: In Java applications , to use any predefined word it must have both word recognition and internal functionality, only word recognition is not sufficient to use in java applications.

Conclusion: In java applications, only keywords are possible to use, Reserved words are not possible to use .

To prepare Java applications, Java has provided the following keywords.

1. Data Types and Return Types:
   void, byte, short, int, long, float, double, boolean, char
2. Access Modifiers:
   public, protected, private, static, final, abstract, native, volatile, synchronized, strictfp,transient.....
3. Flow Controllers:
   if, else, switch, case, default,for, do, while, break, continue,.....
4. Class/Object Related Keywords:
   class, extends, interface, implements, enum, new, this, super, package, import,......
5. Exception Handling Related Keywords:
   throws, throw, try, catch, finally,....

Operators:
—-----------

Operator is a symbol, it is able to perform a particular operation over the provided operands.

Java has provided the following list of operators to perform operations .

1. Arithmetic Operators:

   +, -, *, /, %, ++,--

2. Assignment Operators:
   =, +=, -=, *=,/=,%=
3. Comparison Operators:
   ==, !=, <, <=, >, >=
4. Logical Boolean Operators:
   &, |, ^,....
5. Logical bitwise Operators:
   &, |, ^, <<, >>,...
6. Ternary Operator:
   Expr1?Expr2:Expr3;
7. Short-Circuit Operators:
   &&, ||

EX:

—-

```
class Test{
    public static void main(String[] args){
        int a = 10;
        int b = 3;
        System.out.println(a+b);// 13
        System.out.println(a-b);// 7
        System.out.println(a*b);// 30
        System.out.println(a/b);// 3
        System.out.println(a%b);// 1
    }
}
```

IN Java , there are two types of increment and decrement operators
.

1. Increment Operators: They will increment the value by 1.
   a. Pre Increment Operator : ++var

b.PostIncrement Operator : var++

2. Decrement Operators: They will decrement the value by 1.

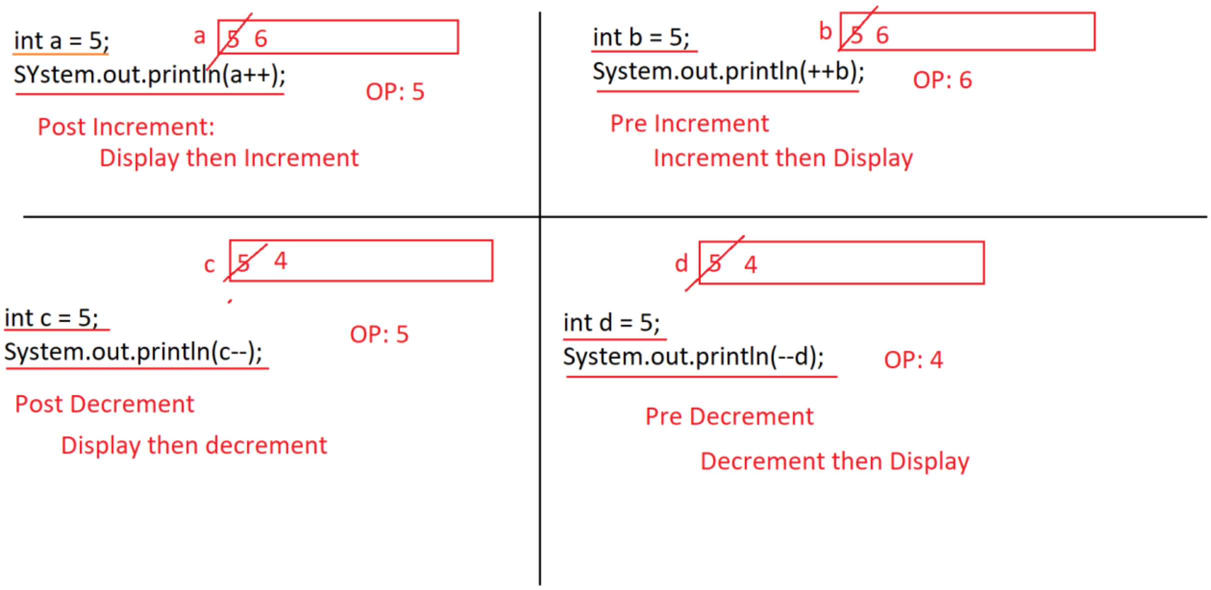
   a.Pre decrement operator : —-var

   b.Post decrement Operator: var—-

EX:

—-

```
class Test{
    public static void main(String[] args){
        int a = 5;
        System.out.println(a++);//5
        System.out.println(a);// 6
        System.out.println();

        int b = 5;
        System.out.println(++b);// 6
        System.out.println(b);// 6
        System.out.println();

        int c = 5;
        System.out.println(c--);// 5
        System.out.println(c);// 4
        System.out.println();

        int d = 5;
        System.out.println(--d);// 4
        System.out.println(d);// 4
    }
}
```



int a = 5;
SYstem.out.println(a++);

a ~~5~~ 6

OP: 5

Post Increment:
Display then Increment

int b = 5;
System.out.println(++b);

b ~~5~~ 6

OP: 6

Pre Increment
Increment then Display

c ~~5~~ 4

int c = 5;
System.out.println(c--);

OP: 5

Post Decrement
Display then decrement

d ~~5~~ 4

int d = 5;
System.out.println(--d);

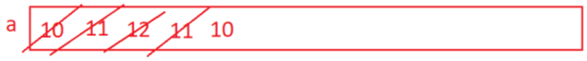
OP: 4

Pre Decrement
Decrement then Display

EX:

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

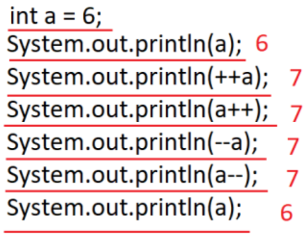
```
int a = 10;
System.out.println(a);    10
System.out.println(a++);  10
System.out.println(++a);  12
System.out.println(a--);  12
System.out.println(--a);  10
System.out.println(a);    10
```



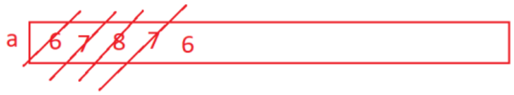
```
 EX:
class Test{
    public static void main(String[] args){
        int a = 6;
        System.out.println(a);
        System.out.println(++a);
        System.out.println(a++);
        System.out.println(--a);
        System.out.println(a--);
        System.out.println(a);
    }
}
```

```
OP:
6
7
7
7
```

```
7
6
```



```
int a = 6;
System.out.println(a);    6
System.out.println(++a);  7
System.out.println(a++);  7
System.out.println(--a);  7
System.out.println(a--);  7
System.out.println(a);    6
```



EX:

```
class Test{
    public static void main(String[] args){
        int a = 5;
        System.out.println(++a+--a);
    }
}
OP: 11
```

int a = 5;

System.out.println(++a+--a);

a ~~5~~ ~~6~~ 5

++a + --a

6 + 5 = 11

EX:

--

```
class Test{
    public static void main(String[] args){
        int a = 7;
        System.out.println(--a+a--+--a);
    }
}
```

OP:

```
16
```

```
int a = 7;
System.out.println(--a+a--+--a);
```

a $\boxed{\cancel{7}\ \cancel{6}\ \cancel{5}\ \ 4}$

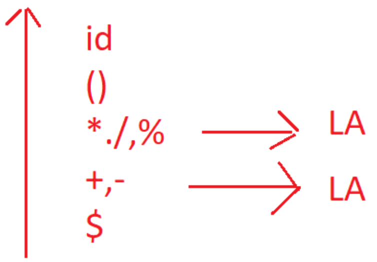
--a + a-- + --a

6 + 6 + 4

12 + 4

16

Operators Priorities And Associativity Rules



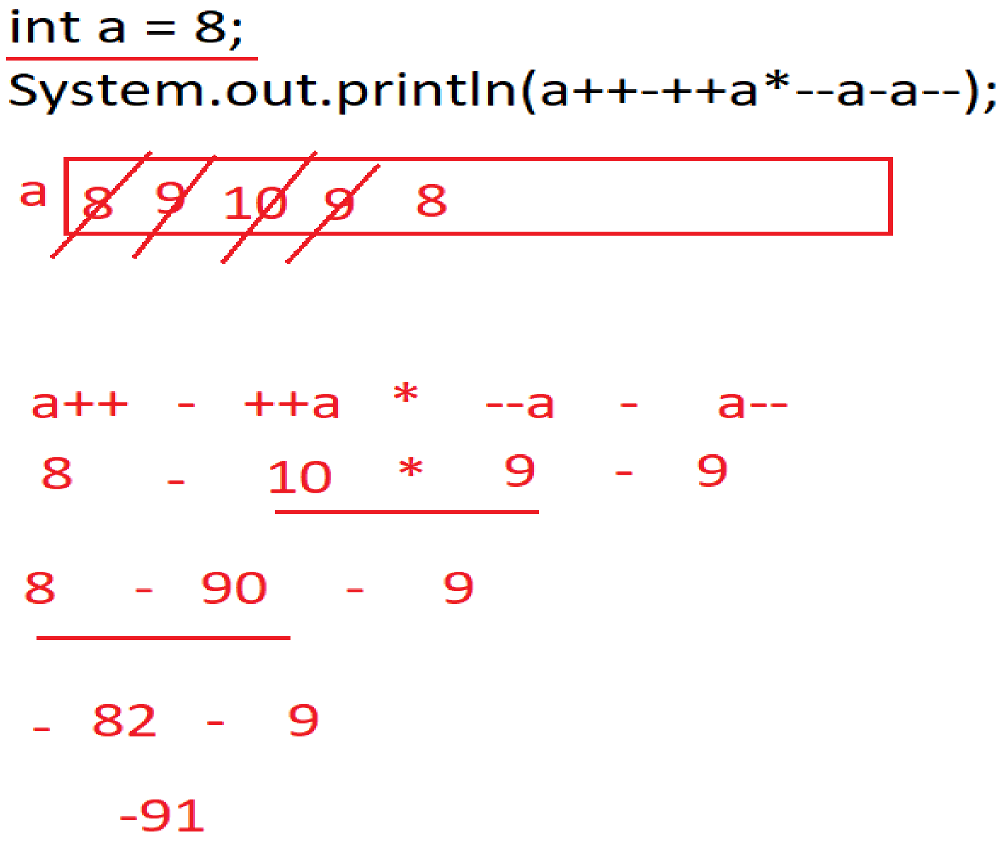
id
()
*./,% → LA
+,- → LA
$

EX:

```
class Test{
    public static void main(String[] args){
        int a = 8;
        System.out.println(a++-++a*--a-a--);
    }
}
```

OP:

```
-91
```



```
int a = 8;
System.out.println(a++-++a*--a-a--);
```

a ~~8~~ ~~9~~ ~~10~~ ~~9~~ 8

a++ - ++a * --a - a--

8 - 10 * 9 - 9

8 - 90 - 9

- 82 - 9

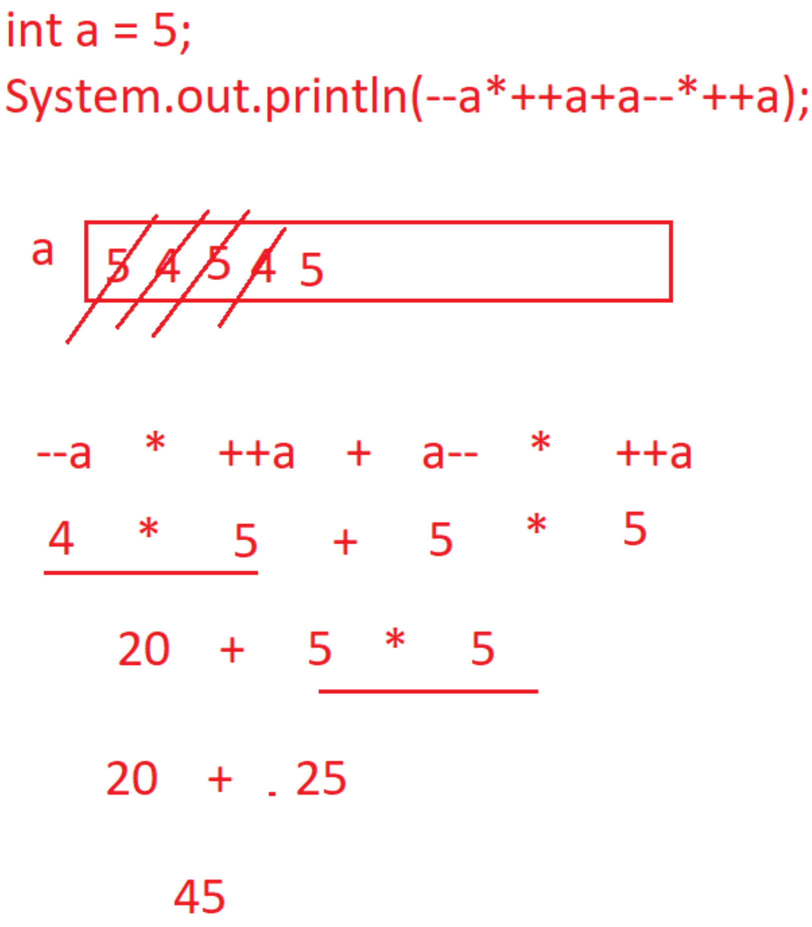
-91

EX:

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

OP:

```
45
```



int a = 5;
System.out.println(--a*++a+a--*++a);

a ~~5~~ ~~4~~ ~~5~~ ~~4~~ 5

--a * ++a + a-- * ++a

4 * 5 + 5 * 5

20 + 5 * 5

20 + . 25

45

EX:
--

```
class Test{
    public static void main(String[] args){
        int a = 7;
        System.out.println((--a+--a)-(++a-a++)*(++a-++a)+(++a+--
a));
    }
}
```

OP:
30

int a = 7

(--a + --a) - (++a - a++) * (++a - ++a) + (++a + --a)

(6 + 5) - (6 - 6) * (8 - 9) + (10 + 9)

$11 - \underline{0 * -1} + 19$

$\underline{11 - 0} + 19$

11 + 19

30

a ~~7~~ ~~6~~ ~~5~~ ~~6~~ ~~7~~ ~~8~~ ~~9~~ ~~10~~ 9

```
EX:
class Test{
    public static void main(String[] args){
        int a = 5;
        System.out.println(++a*--a/2-++a);
    }
}

OP:
9
```

int a = 5;

++a * --a / 2 - ++a

$\underline{6 * 5} / 2 - 6$

$\underline{30 / 2} - 6$

15 - 6 = 9

a ~~5~~ ~~6~~ ~~5~~ 6

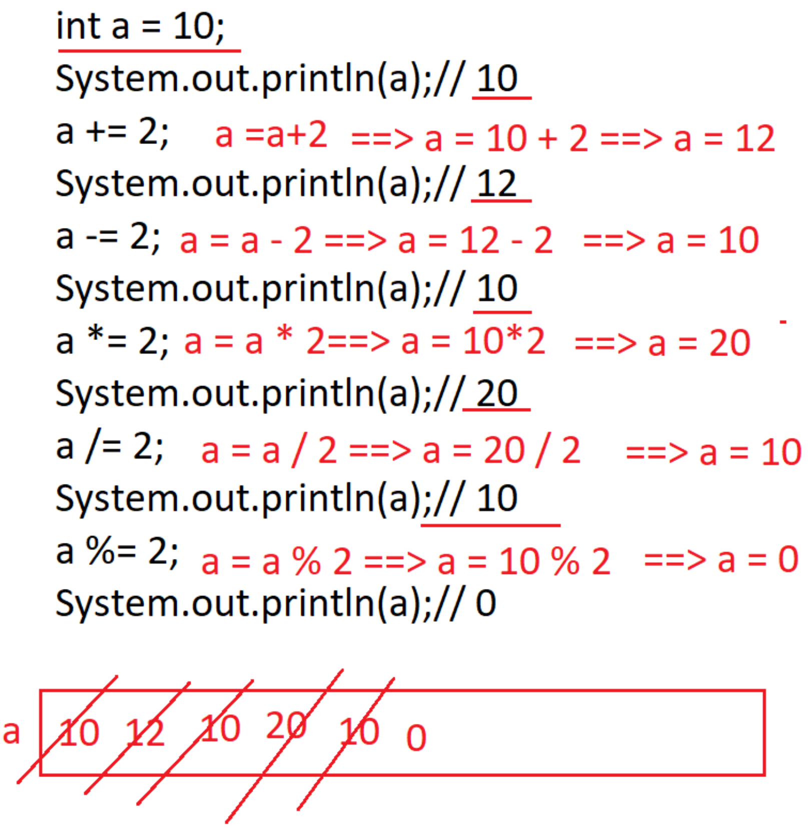
EX:

--

```
class Test{
    public static void main(String[] args){
        int a = 10;
        System.out.println(a);// 10
        a += 2;
        System.out.println(a);// 12
        a -= 2;
        System.out.println(a);// 10
        a *= 2;
        System.out.println(a);// 20
        a /= 2;
        System.out.println(a);// 10
        a %= 2;
        System.out.println(a);// 0
    }
}
```

OP:
--

```
10
12
10
20
10
0
```



```
int a = 10;
System.out.println(a);// 10
a += 2;    a =a+2  ==> a = 10 + 2 ==> a = 12
System.out.println(a);// 12
a -= 2;  a = a - 2 ==> a = 12 - 2   ==> a = 10
System.out.println(a);// 10
a *= 2; a = a * 2==> a = 10*2   ==> a = 20
System.out.println(a);// 20
a /= 2;   a = a / 2 ==> a = 20 / 2    ==> a = 10
System.out.println(a);// 10
a %= 2;  a = a % 2 ==> a = 10 % 2   ==> a = 0
System.out.println(a);// 0
```

EX:
--

```
class Test{
    public static void main(String[] args){
        int a = 12;
        System.out.println(a);// 12
        a += 3;// a = a + 3 ==> a = 12+3 ==> a = 15
        System.out.println(a); // 15
        a -= 4;// a = a - 4 ==> a = 15-4 ==> a = 11
        System.out.println(a);// 11
        a *= 2; // a = a * 2 ==> a = 11 * 2 ==> a = 22
        System.out.println(a);// 22
        a /= 3;// a = a / 3 ==> a = 22 / 3 ==> a = 7
        System.out.println(a);// 7
        a %= 3; // a = a % 3 ==> a = 7 % 3 ==> a = 1
        System.out.println(a);// 1
    }
}
```

OP:

```
12
15
11
22
7
1
```

A	B	A&B	A|B	A^B
T	T	T	T	F
T	F	F	T	T
F	T	F	T	T
F	F	F	F	F

EX:

—--

```
class Test{
    public static void main(String[] args){
        boolean b1 = true;
        boolean b2 = false;

        System.out.println(b1&b1);// true
        System.out.println(b1&b2);// false
        System.out.println(b2&b1);// false
        System.out.println(b2&b2);// false

        System.out.println(b1|b1);// true
        System.out.println(b1|b2);// true
        System.out.println(b2|b1);// true
        System.out.println(b2|b2);// false

        System.out.println(b1^b1);// false
        System.out.println(b1^b2);// true
        System.out.println(b2^b1);// true
        System.out.println(b2^b2);// false
    }
}
```

```
OP:
—-
true
false
false
false
true
true
true
false
false
true
```

```
true
false
```

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

EX:
—--

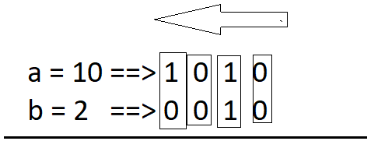
```
class Test{
    public static void main(String[] args){

        int a = 10;// 1010
        int b = 2;//  0010

        System.out.println(a&b);//
        System.out.println(a|b);//
        System.out.println(a^b);//
        System.out.println(a<<b);//
        System.out.println(a>>b);//
    }
}
```

OP:
—--

```
2
10
8
40
```

2

```
int a = 10;
int b = 2;
System.out.println(a&b);
System.out.println(a|b);
System.out.println(a^b);
System.out.println(a<<b);
System.out.println(a>>b);
```



```
a = 10 ==> 1 0 1 0
b = 2  ==> 0 0 1 0
```

```
a&b  ==> 0 0 1 0   ==> 2

a|b  ==> 1 0 1 0   ==> 10

a^b  ==>  1 0 0 0   ==> 8

a<<b ==> 10 << 2==>   00001010
                         00101000 ===> 32 + 8 ==> 40

a>>b==>10>>2 ==> 00001010
                 00 000010    ==> 2
```

EX:

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

        int a = 10;
        int b = 3;

        System.out.println(a&b);
        System.out.println(a|b);
        System.out.println(a^b);
        System.out.println(a<<b);
        System.out.println(a>>b);
    }
}
```

```
OP:
—-
2
11
9
80
1
```

Ternary Operator:
—--------------------

Syntax:
Expr1?Expr2:Expr3;

Where Expr1 is a conditional Expression[Boolean Expression], its result may be true or false.
Where Expr2 and Expr3 are normal Expressions.

If the Expr1 result is true then JVM will execute Expr2.
If the Expr1 result is false then JVM will execute Expr3.

EX:
—---

```
class Test{
    public static void main(String[] args){
        int a = 10;
        int b = 20;

        int bigNum = a > b ? a : b ;

        System.out.println(bigNum);

    }
}
```

EX:

—--

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

        int a = 10;
        int b = 20;

        int smallNum = a < b ? a : b ;

        System.out.println(smallNum);

    }
}
```

Q)Write a Java program to check whether a number is an even number or odd number by using Ternary Operator?

—--

```
class Test{
    public static void main(String[] args){
        int a = 5;
        String evenOrOdd = (a % 2 == 0) ? "Even Number" : "ODD 
Number" ;
        System.out.println(evenOrOdd);
    }
}
```

```
OP: ODD NUmber
```

EX:

```
class Test{
    public static void main(String[] args){
        int a = 10;
        String evenOrOdd = (a % 2 == 0) ? "Even Number" : "ODD 
Number" ;
        System.out.println(evenOrOdd);
    }
}
```

OP

```
Even Number
```

In Java applications, we are able to write nested Ternary Operators, that is we are able to write ternary operators inside another ternary operator.

Syntax:

```
Expr1?(Ex2?Ex3:Ex4):(Ex5?Ex6:Ex7);
```

Where we can write Ternary operators in Expr2, Expr3.

Q)Write a Java program to find the biggest among three numbers by using Ternary operators?

—--

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

        int a = 10;
        int b = 20;
        int c = 30;

        int bigNum = (a>b)?(a>c?a:c):(b>c?b:c);
        System.out.println("Biggest Num : "+bigNum);

    }
}
```

Short-Circuit Operators:
—---------------------------
The main intention of the Short-Circuit operator is to improve Java applications performance.
EX: &&, ||

Q)What is the difference between |[Single OR] and ||[Double OR] ?
—--

Ans:
—---
In the case of a Logical-OR operator , If the first operand value is true then it is not required to check the second operand value , directly we can get the result of the overall OR operator is true.

In the case of | [Single OR], if the first operand value is true even JVM will not go for the shortcut, JVM will evaluate second operand value and JVM will find the overall OR operator result on the basis of both the operand values.

In the above case, JVM is evaluating second operand unnecessarily, it will increase application execution time, it will reduce application performance.

In the case of ||[Double OR], if the first operand value is true then JVM will use Shortcut of OR operator and JVM will find the overall result of OR operator true without evaluating Second operand.

In the above case, JVM is not evaluating second operand, so it will reduce application execution time, and it will improve application performance.

EX:
—--

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

        int a = 10;
        int b = 10;

        if(a++ == 10 | b++ == 10){
            System.out.println(a+"    "+b);// 11    11
        }

        int c = 10;
        int d = 10;

        if(c++ == 10 || d++ == 10){
            System.out.println(c+"    "+d); // 11   10
        }
    }
}
```

Q)What is the difference between &[Single AND] and &&[Double AND] ?
—--

Ans:
—---
In the case of a Logical-AND operator , If the first operand value is false then it is not required to check the second operand value , directly we can get the result of the overall AND operator is false.

In the case of & [Single AND], if the first operand value is false even JVM will not go for the shortcut, JVM will evaluate second operand value and JVM will find the overall AND operator result on the basis of both the operand values.

In the above case, JVM is evaluating second operand unnecessarily, it will increase application execution time, it will reduce application performance.

In the case of &&[Double AND], if the first operand value is false then JVM will use Shortcut of AND operator and JVM will find the overall result of AND operator false without evaluating Second operand.

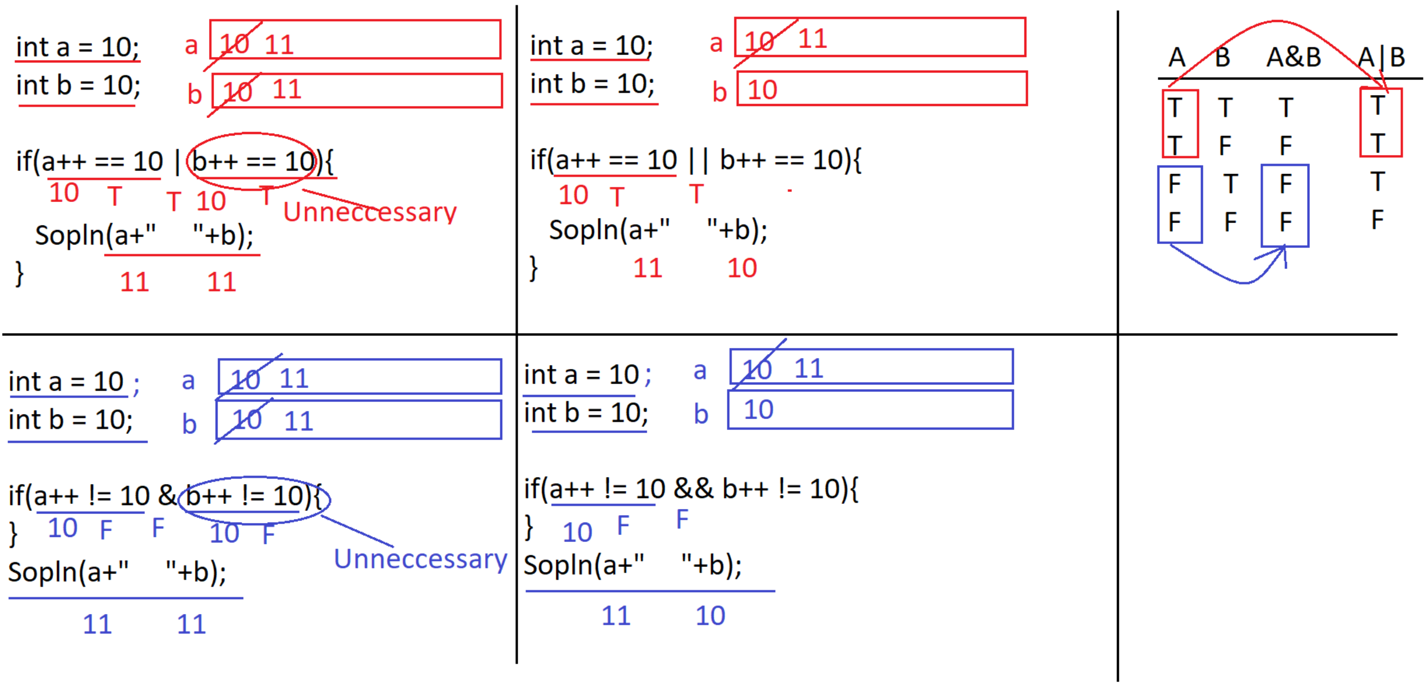
In the above case, JVM is not evaluating second operand, so it will reduce application execution time, and it will improve application performance.

EX:
—--

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

        int a = 10;
        int b = 10;

        if(a++ != 10 & b++ != 10){
        }
        System.out.println(a+"    "+b);

        int c = 10;
        int d = 10;

        if(c++ != 10 && d++ != 10){
        }
        System.out.println(c+"    "+d);
    }
}
```

OP:
—--

```
11    11
11    10
```



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

if(a++ == 10 | b++ == 10){
  Sopln(a+"    "+b);
}
```

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

if(a++ == 10 || b++ == 10){
  Sopln(a+"    "+b);
}
```

A	B	A&B	A|B
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F

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

if(a++ != 10 & b++ != 10){
}
Sopln(a+"    "+b);
```

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

if(a++ != 10 && b++ != 10){
}
Sopln(a+"    "+b);
```

Data Types:

There are two types of Programming languages .

1. Typed Programming Languages
2. Typeless Programming Languages

If any programming language is able to represent data without having types then that programming language is Typeless Programming language.

EX: All First Generation Programming languages are typeless programming languages.

EX: ASL, BASIC

Typed Programming Languages:

If any programming language is able to represent data on the basis of data types then that programming language is called Typed Programming language.

There are two types of Typed Programming Languages .

1. Statically Typed programming Languages.
2. Dynamically Typed Programming Languages.

Dynamically Typed Programming Languages:In Any programming language, if we represent data without specifying data type , identifying the data type after representing data then the programming language is called “Dynamically Typed Programming Language”.
Conclusion: First Data then Data Type
EX: Python

```
a = 10
print(type(a)); —> int
```

Statically Typed programming Languages:
In any Programming Language, if we represent data after specifying data type then the programming language is “Statically Typed Programming Language”.
Conclusion: First Data Type then Data
EX: C, C++, Java
EX:

```
int a = 10;
```

EX:

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

In statically Typed Programming languages, first we have to specify the type of data before representing data , here to specify type of data we have to use “Data Types”.

Q)What are the advantages we are able to get from data Types in Java applications?

—--

Ans:

—---

1. On the basis of data types we are able to identify the memory sizes for the data.

EX: int a = 10; → ‘a’ variable will have 4 bytes of memory on the basis of int

EX: byte b = 22;--> ‘b’ variable will have 1 byte of memory on the basis of byte

2. On the basis of data types only we are able define range values for the data for the variables.
   EX: byte variables are able to take the values from -128 to +127.

To prepare Java applications, JAVA has provided the following Data Types.

1. Primitive Data types / Primary Data Types:
   I. Numeric Data Types:

a. Integral / Integer Data Types:

byte ------------> 1 byte ----> 0
short ----------> 2 bytes----> 0
int -------------> 4 bytes----> 0
long------------> 8 bytes----> 0L

b. Non Integral Data Types:

float------------> 4 bytes------> 0.0f
double ---------> 8 bytes-------> 0.0

II. Non Numeric Data Types:

char------------------> 2 bytes ------> ‘ ‘[Single space]
boolean--------------> 1 bit ----------> false

2. User defined Data Types / Secondary Data Types:

EX: All Arrays, All classes, abstract classes, interfaces,.....

Note: No fixed memory sizes for the user defined data types, memory sizes are depending on the volume of data which we represented.

Note: The default value for all the user defined variables is ‘null’.

EX: String str; // where str value is null

In Java applications, to find range values of the data types we have to use the following formula.

$$-2^{n-1} \quad \text{to} \quad 2^{n-1} - 1$$

Where ‘n’ is the number of bits of the data type.

EX: byte
Size: 1 byte = 8 bits

-2^{8-1} to $2^{8-1} - 1$

-2^{7} to $2^{7} - 1$

-128 to 128 - 1

-128 to 127

The above formula is applicable for Integral data Types only, not for all the remaining data types.

To represent Min Value and Max Value of each and every data type , JAVA has provided the following constant variables from all the wrapper classes.

MIN_VALUE
MAX_VALUE

Note: Java has provided a separate class representation for each and every primitive data type , where the class which is representing the primitive data type is called “Wrapper Classes”.

byte –--------------> java.lang.Byte
short–--------------> java.lang.Short

```
int—-----------------> java.lang.Integer
long—---------------> java.lang.Long
float—--------------> java.lang.Float
double—------------> java.lang.Double
char—---------------> java.lang.Character
boolean—------------> java.lang.Boolean
```

Note: MIN_VALUE and MAX_VALUE constants are available in almost all the wrapper classes except Boolean, these constants exist in Character class but they are able to provide space is min value and ? is max value.

EX:
—-

```
class Test{
    public static void main(String[] args){
        System.out.println(Byte.MIN_VALUE+" ---->
"+Byte.MAX_VALUE);
        System.out.println(Short.MIN_VALUE+" --->
"+Short.MAX_VALUE);
        System.out.println(Integer.MIN_VALUE+" –>
"+Integer.MAX_VALUE);
        System.out.println(Long.MIN_VALUE+" ---->
"+Long.MAX_VALUE);
        System.out.println(Float.MIN_VALUE+" ---->
"+Float.MAX_VALUE);
        System.out.println(Double.MIN_VALUE+" ---->
"+Double.MAX_VALUE);
        System.out.println(Character.MIN_VALUE+" ---->
"+Character.MAX_VALUE);
        //System.out.println(Boolean.MIN_VALUE+" ---->
"+Boolean.MAX_VALUE);
```

```
    }
}
```

Output:

```
-128 ----> 127
-32768 ----> 32767
-2147483648 ----> 2147483647
-9223372036854775808 ----> 9223372036854775807
1.4E-45 ----> 3.4028235E38
4.9E-324 ----> 1.7976931348623157E308
  ----> ?
```

Type Casting:

The process of converting data from one data type to another data type is called “Type Casting”.
The main advantage of Type Casting is “Flexibility” To represent Data.

There are two types of Type Castings.

1.Primitive Data Types Type Casting
2.User Defined Data Types Type casting

User Defined Data Types Type casting: The process of converting data from One user defined data type to another data type is called “User defined Data Types Type Casting”.

To perform User defined data types type casting we have to provide either extends[Inheritance] or implements[Polymorphism] relation between two user defined data types.

Primitive Data Types Type Casting:

–--------------------------------------

The process of converting data from One primitive data type to another primitive data type is called “Primitive Data Types Type Casting”.

There are two types of Primitive Data Types Type castings.

1. Implicit Type casting / Widening
2. Explicit Type Casting / Narrowing

Implicit Type Casting:
The process of converting data from lower data type to higher data type is called “Implicit Type casting”.

To cover all possibilities of Implicit Type casting JAVA has provided a predefined chart.

```
  1            2          4       8        4             8
   byte—>short—>int—>long—>float—>double
                             ^
                             |
                          char
                           2
```

IN Java applications, to perform implicit type casting we have to assign lower data type variables to higher data type variables as per the above chart.
EX:

```
byte b = 10;
int i = b;
```

When we compile the above code[Assignment statement], the compiler will perform the following actions.

1. Compiler will recognize right side variable data type and left side variable data type.
2. Compiler will check whether the right side variable data type is compatible with the left side variable data type or not.
3. If the right side variable data type is not compatible with left side variable data type then compiler will raise any error like
   JAVA7: Possible loss of Precision.
   JAVA8: Incompatible Types: possible lossy conversion from xxx data type to yyy data type.
4. If the right side variable data type is compatible with the left side variable data type then Compiler will not raise any error, but at the same time Compiler will not perform any type casting.

   Note: In Java, all lower data types are compatible with higher data types, so we can assign lower data type variables to higher data type variables, all higher data types are not compatible with lower data types, so we are unable to assign higher data type variables to lower data type variables.

   Note: In Java, compiler is responsible for only Type checking, not responsible for Type casting, IN Java, only JVM is responsible for Type casting.

When we execute the above code[Assignment Statement] JVM will perform the following actions.

1. JVM will convert the right side variable data type to left side variable data type internally or implicitly[Implicit Type Casting].

2. JVM will copy the value from the right variable to the left side variable.

EX:
–--

```
class Test{
    public static void main(String[] args){
        int i = 10;
        byte b = i;
        System.out.println(i+"    "+b);
    }
}
```

Status: Compilation Error

EX:

```
class Test{
    public static void main(String[] args){
        byte b = 10;
        int i = b;
        System.out.println(b+"   "+i);
    }
}
```

Status: No Compilation Error

```
OP: 10     10
```

EX:
–--

```
class Test{
    public static void main(String[] args){
        byte b = 65;
        char c = b;
        System.out.println(b+"    "+c);
    }
}
```

Status: Compilation Error.

EX:

```
class Test{
    public static void main(String[] args){
        char c = 'A';
        short s = c;
        System.out.println(c+"    "+s);
    }
}
```

Status: Compilation Error

Reason: IN Implicit Type Casting, no conversions exist between char and byte, short and char due to the difference in internal data representations.

EX:

```
class Test{
    public static void main(String[] args){
        char c = 'A';
        int i = c;
        System.out.println(c+"    "+i);
    }
}
```

Status: No COmpilation Error

```
OP: A      65
```

EX:

```
class Test{
    public static void main(String[] args){
        long l = 10;
        float f = l;
        System.out.println(l+"    "+f);
    }
}
```

Status: No Compilation Error

OP: 10   10.0

EX:

```
class Test{
    public static void main(String[] args){
        float f = 22.22f;
        long l = f;
        System.out.println(l+"    "+f);
    }
}
```

Status: Compilation Error.

EX:

```
class Test{
    public static void main(String[] args){
        float f = 22.22f;
        double d = f;
        System.out.println(f+"    "+d);
    }
}
```

Status: No Compilation Error

OP: 22.22   22.2199999

EX:

—--

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

Status: Compilation Error.

Reason: 128 is more than the max value of byte, so 128 will come under integer.

EX:

```
class Test{
    public static void main(String[] args){
        byte b1 = 60;
        byte b2 = 70;
        byte b = b1 + b2;
        System.out.println(b);
    }
}
```

Status: Compilation Error.

EX:

—--

```
class Test{
    public static void main(String[] args){
        byte b1 = 30;
        byte b2 = 30;
        byte b = b1 + b2;
        System.out.println(b);
    }
}
```

Status: Compilation Error

Reason:
If we add two data type variables then the resultant value may not be of the same data type, it may be of the next higher data type.

X,Y and Z are three primitive data types.
X + Y = Z
1. If X and Y belong to {byte, short, int} then Z should be int.
2. If either X or Y or both X and Y belong to {long, float, double} then Z should
    be higher(X,Y).

EX:

```
byte + int = int
short + byte = int
short + int = int
int + int = int
byte + long =  long
long + float = float
float + double = double
```

EX:
—--

```
class Test{
    public static void main(String[] args){
        byte b1 = 60;
        byte b2 = 70;
        int i = b1 + b2;
        System.out.println(i);
    }
}
```

Status: No Compilation Error
OP: 130

EX:

```
class Test{
    public static void main(String[] args){
        float f = 22.22f;
        long l = 10;
        long l1 = f + l;
        System.out.println(l1);
    }
}
```

Status: Compilation Error

EX:

```
class Test{
    public static void main(String[] args){
        long l = 10;
        double d = 23.345;
        float f = l + d;
        System.out.println(f);
    }
}
```

Status: Compilation Error.

Explicit Type Casting/ Narrowing:
—-------------------------------------

The process of converting data from higher data type to lower data type is called “Explicit Type Casting”.

To perform Explicit Type casting we have to use the following pattern.

```
P    a   =     (Q)    b;
```

Where ‘b’ variable is of the higher data type than ‘a’ variable data type.
Where Q is either same as P or lower than P.
EX:

```
int i = 10;
byte b = (byte)i;
```

If we compile the above assignment statement then the compiler will perform the following actions.

1. Compiler will recognize the cast operator provided data type and left side variable data type .
2. Compiler will check whether the cast operator provided data type is compatible with the left side variable data type or not.
3. If the cast operator provided data type is not compatible with the left side variable data type then the compiler will raise an error.
4. If the cast operator provided data type is compatible with the left side variable data type then the compiler will not raise any error and the compiler will not perform any type casting.

Note: IN Java applications compiler is responsible for only Type checking , compiler is not responsible for type casting.

When we execute the above assignment statement then JVM will perform the following actions.

1. JVM will convert the data from right side variable data type to cast operator provided data type explicitly[Explicit Type Casting].
2. JVM will copy the value from the right side variable to the left side variable.

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        byte b = (byte)i;
        System.out.println(i+"    "+b);
    }
}
```

Status: No Compilation Error

```
OP: 10    10
```

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        short s = (short)i;
        System.out.println(i+"    "+s);
    }
}
```

Status: No Compilation Error

```
OP: 10    10
```

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        short s = (byte)i;
        System.out.println(i+"    "+s);
    }
}
```

Status: No COmpilation Error

OP: 10   10

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        short s = (char)i;
        System.out.println(i+"    "+s);
    }
}
```

Status: Compilation Error

EX:

Consider the following program

```
class Test{
    public static void main(String[] args){
        int i = 10;
        short s = (X)i;
        System.out.println(i+"    "+s);
    }
}
```

What are the possible values for X.

a.byte, char
b.byte, short
c.short, char
d.None

Ans: b

EX:

```
class Test{
    public static void main(String[] args){
        byte b = 65;
        char c = (char)b;
        System.out.println(b+"     "+c);
    }
}
```

Status: NoCompilation Error.

OP: 65   A

EX:

```
class Test{
    public static void main(String[] args){
        char c = 'A';
        short s = (short)c;
        System.out.println(c+"    "+s);
    }
}
```

Status: No Compilation Error

OP: A   65

EX:

```
class Test{
    public static void main(String[] args){
        char c = 'A';
        short s = (byte)c;
        System.out.println(c+"    "+s);
    }
}
```

Status: No Compilation Error

OP: A   65

Reason: Conversions are not possible between byte and char, short and char in implicit type casting, but these conversions are possible in Explicit Type Casting, because Explicit Type casting is forcible type casting.

EX:

```
class Test{
    public static void main(String[] args){
        int i = 140;
        byte b = (byte)i;
        System.out.println(i+"    "+b);
    }
}
```

Status: No Compilation Error

```
OP: 140    -116
```

Reason:

If the given value ‘n’ is exceeding max value of the byte then the resultant value of the byte is ((n-127) - 1) -128

EX:

```
class Test{
    public static void main(String[] args){
        byte b1 = 60;
        byte b2 = 70;
        byte b = (byte)b1 + b2;
        System.out.println(b);
    }
}
```

Status: Compilation Error

EX:

```
class Test{
    public static void main(String[] args){
        byte b1 = 60;
        byte b2 = 70;
        byte b = (byte)(b1 + b2);
        System.out.println(b);
    }
}
```

Status: No Compilation Error

OP: -126

EX:

```
class Test{
    public static void main(String[] args){
        float f = 22.22f;
        long l = 10;
        long l1 = l + (long)f;
        System.out.println(l1);
    }
}
```

Status: No Compilation Error

OP: 32

EX:

```
class Test{
    public static void main(String[] args){
        float f = 22.22f;
        long l = 10;
        long l1 = (float)l + (long)f;
        System.out.println(l1);
    }
}
Status: Compilation Error
```

EX:

```
class Test{
    public static void main(String[] args){
        double d = 22.234;
        float f = (X)d;
        System.out.println(d+"   "+f);
    }
}
```

Q)What are the possible values for X ?

Ans:

byte, short, int, long, float

In the above program if we provide byte, short, int and long inplace of X then we are able to get the following output.

```
22.234   22.0
```

In the above program if we provide a float inplace of X then we are able to get the following output.

```
22.234   22.234
```

EX:

```
class Test{
    public static void main(String[] args){
        double d = 22.222;
        byte b = (byte)(short)(int)(long)(float)d;
        System.out.println(b);
    }
}
OP: 22
```

EX:

```
class Test{
    public static void main(String[] args){
        double d = 22.222;
        byte b = (byte)(int)(char)(short)(float)(long)d;
        System.out.println(b);
    }
}
Status: No COmpilation Error
op: 22
```

Java Statements:

–------------------

Statement is the collection of Expressions.

To prepare java applications , JAVA has provided the following statements.

1. General Purpose Statements

    These statements are very much common and very much frequent in java applications.

    EX:

    a. Declaring classes, abstract classes, interfaces,....
    b. Declaring variables, methods
    c. Creating Objects for the classes
    d. Accessing variables, methods,....

2. Conditional Statements:

    These statements are able to execute a block of instructions under a
    a particular condition.

    EX: if, switch

3. Iterative Statements:

These Statements are able to execute a block of instructions repeatedly on the basis of a particular condition.
EX: for, for-Each, while, do-while

4. Transfer Statements:
   These statements are able to bypass the flow of execution from one instruction to another instruction .
   EX: break, continue

5. Exception Handling Statements:
   EX: throw, try-catch-finally
6. Synchronized Statements:
   Synchronized methods, Synchronized blocks

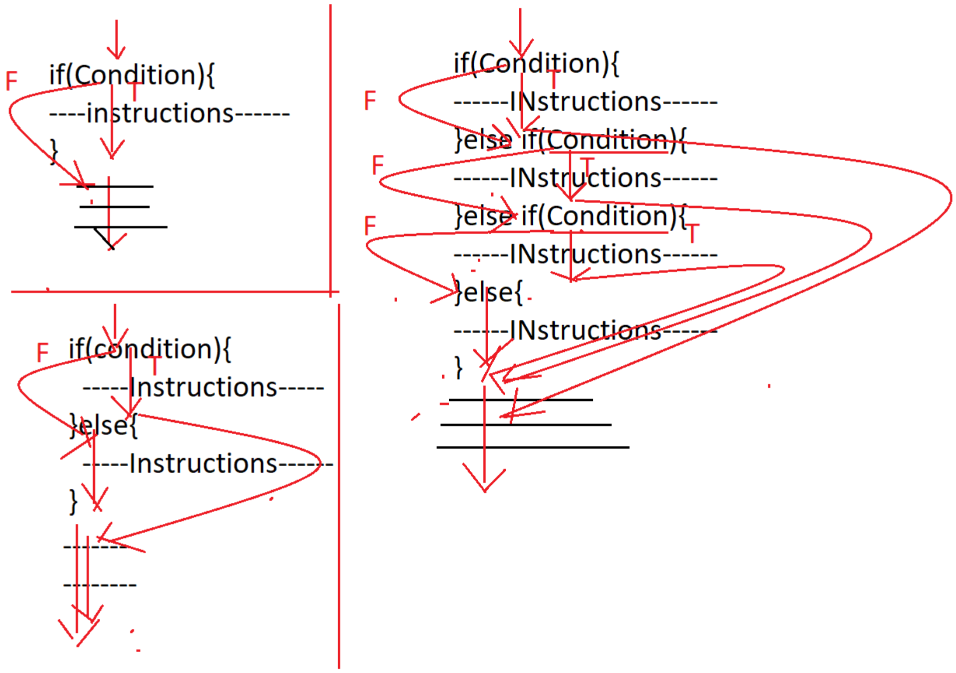
Conditional Statements:
—-------------------------
These statements are able to execute a block of instructions on the basis of a particular condition.
EX: if, switch

if:
—--
Syntax-1:

```
if(condition){
—---instructions—---
}
```

Syntax-2:

```
if(condition){
—--instructions—---
}else{
—---instructions—-----
}
Syntax-3
if(condition){
—--instructions—---
}else if(condition){
—---instructions—---
}else if(condition){
—-
}
—---
—---
else{
—---instructions—---
}
```



```
if(Condition){
----instructions------
}
```

```
if(condition){
  -----Instructions-----
}else{
  -----Instructions------
}
---------
---------
```

```
if(Condition){
------INstructions------
}else if(Condition){
------INstructions------
}else if(Condition){
-------INstructions------
}else{
------INstructions------
}
```

EX:

—--

```
class Test{
    public static void main(String[] args){
        int a = 10;
        if(a%2 == 0){
            System.out.println(a+" Is Even Number");
        }
    }
}
```

```
Status: No Compilation Error
OP: 10 Is Even Number
```

EX:

```
class Test{
    public static void main(String[] args){
        int a = 5;
        if(a%2 == 0){
            System.out.println(a+" Is Even Number");
        }else{
            System.out.println(a+" Is ODD NUmber");
        }
    }
}
```

Q)Write a Java program to display student Result on the basis of the following rules and regulations.

a. If the student marks are less than 35 and greater or equal to 0 then the status is “FAIL”.

b. If the student marks are greater than or equal to 35 and less than 50 then the status is “THIRD CLASS”.

c. If the student marks are greater than or equal to 50 and less than 60 then the status is “SECOND CLASS”.

d. If the student marks are greater than or equal to 60 and less than 70 then the status is “FIRST CLASS”.

e. If the student marks are greater than or equal to 70 and less than or equal to 100 then the status is “DISTINCTION”.

f. None of the above cases is “INVALID MARKS”.

EX:

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

        int smarks = 67;
        String status = "";

        if(smarks < 35 && smarks >= 0){
            status = "FAIL";
        }else if(smarks >= 35 && smarks < 50){
            status = "THIRD CLASS";
        }else if(smarks >= 50 && smarks < 60){
            status = "SECOND CLASS";
        }else if(smarks >= 60 && smarks < 70){
            status = "FIRST CLASS";
        }else if(smarks >= 70 && smarks <= 100){
            status = "DISTINCTION";
        }else{
            status = "INVALID MARKS";
        }
        System.out.println("STudent Marks   : "+smarks);
        System.out.println("Student Status  : "+status);
    }
}
```

Q)Write a Java Program to calculate the tax payment on the basis of the following conditions.

a. If the annual Income is less than 10L and greater than or equal to 5L then the tax pay will be 12%.
b. If the annual Income is less than 25L and greater than or equal to 10L then the tax pay will be 17%.
c. If the annual Income is less than 35L and greater than or equal to 25L then the tax pay will be 22%.
d. If the annual Income is less than 50L and greater than or equal to 35 L then the tax pay will be 28%.
e. If the annual Income is greater than or equal to 50 L then the tax pay will be 35%.
f. If the annual Income is less %5L then it will be tax Free.

EX:

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

        long annualIncome = 3300000L;
        float taxPay = 0.0f;

        if(annualIncome < 500000){
            taxPay = 0.0f;
        }else if(annualIncome >= 500000 && annualIncome <
1000000){
            taxPay = annualIncome*12/100;
        }else if(annualIncome >= 1000000 && annualIncome <
2500000){
```

```
            taxPay = annualIncome*17/100;
        }else if(annualIncome >= 2500000 && annualIncome <
3500000){
            taxPay = annualIncome*22/100;
        }else if(annualIncome >= 3500000 && annualIncome <
5000000){
            taxPay = annualIncome*28/100;
        }else{
            taxPay = annualIncome*35/100;
        }

        System.out.println("Annual Income    : "+annualIncome);
        System.out.println("Tax Pay          : "+taxPay);

    }
}
```

Q)Write a Java program to find the biggest number among two numbers?

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

        int num1 = 20;
        int num2 = 20;

        System.out.println("First Number   : "+num1);
        System.out.println("Second Number  : "+num2);

        if(num1 < num2){
            System.out.println("Biggest Number : "+num2);
        }else if(num1 > num2){
            System.out.println("Biggest Number : "+num1);
        }else{
            System.out.println("Both the Numbers are Same");
        }
    }
}
```

Q)Write a Java program to find the biggest number among three numbers?

—---

— Pending—---

In Java , only class level variables will have default values, local variables will not have default values. If we declare any variable as a local variable then we must provide initialization for that local variable explicitly either at the same declarative statement or at least before accessing that local variable, if we violate this condition then the compiler will raise an error like “Variable xxx might not have been initialized”.

```
class A{
int i; // Class Level variable,  its default value is 0.
void m1(){
   int j;// Local Variable, No default value is existed
   System.out.println(i); // 0
   System.out.println(j); // Compilation Error
   j = 20;
   System.out.println(j); // No Compilation Error, OP: 20
}
}
```

In Java, there are two types of conditional expressions.

1. Constant Conditional Expression
2. Variable Conditional Expression

Constant Conditional Expression:
It includes only constants or final variables, it will be evaluated by the Compiler at compilation time, not by JVM.
EX:

```
  1.if(10 == 10){     }
  2.if(10 == 20){      }
  3.final int i = 10;
    if(i == 10){    }
  4.int i = 10;
    if(i == 10){    } ----> Not constant conditional Expression.
```

Variable Conditional Expression:
It includes at least one non-final variable, it will be evaluated by JVM at runtime , not by compiler at compilation time.
EX:

```
  1.int i = 10;
    int j = 20;
    if(i == j){      }
  2.int  i = 10;
    if(i == 10){     }
```

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        int j;
        if(i == 10){
            j = 20;
        }
        System.out.println(j);
    }
}
```

Status: Compilation Error

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        int j;
        if(i == 10){
            j = 20;
        }else{
            j = 30;
        }
        System.out.println(j);
    }
}
```

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        int j;
        if(i == 10){
            j = 20;
        }else if(i == 20){
            j = 30;
        }
        System.out.println(j);
    }
}
```

Status: Compilation Error

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        int j;
        if(i == 10){
            j = 20;
        }else if(i == 20){
            j = 30;
        }else {
            j = 40;
        }
        System.out.println(j);
    }
}
```

Status: No Compilation Error

OP: 20

EX:

```
class Test{
    public static void main(String[] args){
        final int i = 10;
        int j;
        if(i == 10){
            j = 20;
        }
        System.out.println(j);
    }
}
```

Status: No Compilation Error
OP: 20

EX:

```
class Test{
    public static void main(String[] args){
        int j;
        if(true){
            j = 20;
        }
        System.out.println(j);
    }
}
```

Status: No Compilation Error
OP: 20

switch:
—-------

‘if’ is able to perform single condition checking, switch is able to perform multiple conditions checking.

In general, we will use switch in menu driven applications.

1. PUSH

```
  2.POP
  3.PEEK
  4.EXIT
    Your option: 1
    Enter element to PUSH : AAA
    PUSH Operation is Success

  1.PUSH
  2.POP
  3.PEEK
  4.EXIT
    Your option: 2
    POP operation is Success

  1.PUSH
  2.POP
  3.PEEK
  4.EXIT
    Your Option : 3
    TOP of the Element isd AAA

  1.PUSH
  2.POP
  3.PEEK
  4.EXIT
    Your Option : 4
    Thank You for using Stack Operations

Syntax:
–-------
switch(var){
    case 1:
        —--instructions—---
    break;
    case 2:
        —--instructions—---
    break;
    —-----
    —----
    case n:
        —--instructions—---
    break;
    default:
        —---instructions—-
    break;
}
```

When we execute the above switch, JVM will perform the following actions.

1. JVM will compare switch parameter value with case constant, one by one.
2. If any case constant is matched with the switch parameter value then JVM will execute the respective case provided instructions.
3. After executing all the instructions of the respective case , when JVM encounters a break statement JVM will bypass the flow of execution to outside of the switch.
4. If no matched case is identified in the switch, JVM will execute the default case.

EX:
—-

```
class Test{
    public static void main(String[] args){
        int day = 100;
        switch(day){
            case 1:// day == 1
                System.out.println("MONDAY");
            break;
            case 2:// day == 2
                System.out.println("TUESDAY");
            break;
            case 3:// day == 3
                System.out.println("WENSDAY");
            break;
            case 4:// day == 4
                System.out.println("THURSDAY");
            break;
            case 5:// day == 5
                System.out.println("FRIDAY");
            break;
            case 6:// day == 6
                System.out.println("SATURDAY");
            break;
            case 7:// day == 7
                System.out.println("SUNDAY");
            break;
            default:
                System.out.println("Not a Valid number, please
provide number from 1 to 7");
            break;
        }

    }
}
```

Rules and Regulations to write switch:

–---

1. In switch, the data types like byte, short, int and char are allowed as parameters, but the data types like long, float, double and boolean are not allowed as parameters.

EX:

```
class Test{
    public static void main(String[] args){
        long l = 20L;
        switch(l){
            case 5L:
                System.out.println("FIVE");
            break;
            case 10L:
                System.out.println("TEN");
            break;
            case 15L:
                System.out.println("FIFTEEN");
            break;
            case 20L:
                System.out.println("TWENTY");
            break;

            default:
                System.out.println("Default");
            break;
        }

    }
}
```

Status: Compilation Error

EX:

```
class Test{
    public static void main(String[] args){
        char ch = 'B';
        switch(ch){
            case 'A':
                System.out.println("A");
            break;
            case 'B':
                System.out.println("B");
            break;
            case 'C':
                System.out.println("C");
            break;
            case 'D':
                System.out.println("D");
            break;

            default:
                System.out.println("Default");
            break;
        }

    }
}
```

Status: No Compilation Error

EX:

```
class Test{
    public static void main(String[] args){
        boolean b = true;
        switch(b){
            case true:
                System.out.println("true");
            break;
            case false:
                System.out.println("false");
            break;

            default:
                System.out.println("Default");
            break;
        }

    }
}
```

Status: Compilation Error.

2. In Java, switch is able to allow String data type as parameter right from JAVA 1.7 version onwards, Up to Java 1.6 version , switch is not allowing String data type as parameter.

EX:

```
class Test{
    public static void main(String[] args){
        String str = "BBB";
        switch(str){
            case "AAA":
                System.out.println("AAA");
            break;
            case "BBB":
                System.out.println("BBB");
            break;
            case "CCC":
                System.out.println("CCC");
            break;
            case "DDD":
                System.out.println("DDD");
            break;
            default:
                System.out.println("Default");
            break;
        }

    }
}
```

Java6: Compilation Error

Java7: No Compilation Error

3. In switch,
    a. all cases are optional.
    b. Default is optional
    c. Both cases and default are optional.

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        switch(i){

        }

    }
}
```

Status: No Compilation Error

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        switch(i){
            case 5:
                System.out.println("FIVE");
            break;
            case 10:
                System.out.println("TEN");
            break;
            case 15:
                System.out.println("FIFTEEN");
            break;
            case 20:
                System.out.println("TWENTY");
            break;

        }

    }
}
```

Status: No Compilation Error

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        switch(i){

            default:
                System.out.println("Default");
            break;
        }
    }
}
```

Status: No Compilation Error

4. In switch, ‘break’ statement is optional, if we write switch without break statement then compiler will not raise any error, but JVM will execute all the cases right from matched case until it gets either break statement or end of switch.

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        switch(i){
            case 5:
                System.out.println("FIVE");

            case 10:
                System.out.println("TEN");

            case 15:
                System.out.println("FIFTEEN");

            case 20:
                System.out.println("TWENTY");

            default:
                System.out.println("Default");

        }

    }
}
Status: No Compilation Error
OP:
TEN
FIFTEEN
TWENTY
Default
```

5. In switch, all case constants must be provided within the range of the data type which we passed as parameter to switch.

   EX:

```
class Test{
    public static void main(String[] args){
        byte b = 126;
        switch(b){
            case 125:
                System.out.println("125");
            break;
            case 126:
                System.out.println("126");
            break;
            case 127:
                System.out.println("127");
            break;
            case 128:
                System.out.println("128");
            break;
            default:
                System.out.println("Default");
            break;
        }
    }
}
Status: Compilation Error.
```

6. In switch, all case values must be either direct constants or final variables, we must not provide variables in place of case constants.

EX:

```
class Test{
    public static void main(String[] args){
        int i = 5, j = 10, k = 15, l = 20;
        switch(10){
            case i:
                System.out.println("FIVE");
            break;
            case j:
                System.out.println("TEN");
            break;
            case k:
                System.out.println("FIFTEEN");
            break;
            case l:
                System.out.println("TWENTY");
            break;
            default:
                System.out.println("Default");
            break;
        }
    }
}
```

Status: Compilation Error

EX:

```
class Test{
    public static void main(String[] args){
        final int i = 5, j = 10, k = 15, l = 20;
        switch(10){
            case i:
                System.out.println("FIVE");
            break;
            case j:
                System.out.println("TEN");
            break;
            case k:
                System.out.println("FIFTEEN");
            break;
            case l:
                System.out.println("TWENTY");
            break;
            default:
                System.out.println("Default");
            break;
        }
    }
}
```

Status: No Compilation Error
OP: 10

7. IN switch, duplicate cases are not allowed.

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        switch(i){
            case 5:
                System.out.println("FIVE");
            break;
            case 10:
                System.out.println("TEN");
            break;
            case 20:
                System.out.println("TWENTY");
            break;
            case 15:
                System.out.println("FIFTEEN");
            break;
            case 20:
                System.out.println("TWENTY");
            break;
            default:
                System.out.println("Default");
            break;
        }
    }
}
```

Status: Compilation Error

8. In switch, we can provide all cases and default in any order.

EX:

```
class Test{
    public static void main(String[] args){
        int i = 10;
        switch(i){
                default:
                    System.out.println("Default");
                break;
                case 15:
                    System.out.println("FIFTEEN");
                break;
                case 5:
                    System.out.println("FIVE");
                break;
                case 20:
                    System.out.println("TWENTY");
                break;
                case 10:
                    System.out.println("TEN");
                break;




            }
        }
    }
```

Status: No Compilation Error.

Iterative Statements:
—------------------------

It is able to execute a set of instructions repeatedly on the basis of a particular condition.

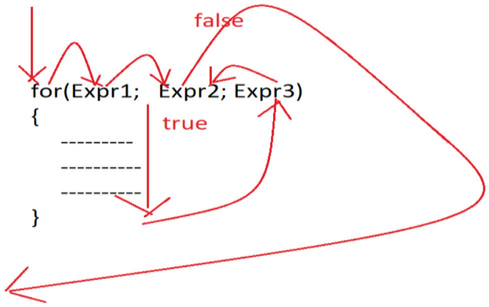
Java has provided the following iterative statements to prepare java applications.

1. for
2. while
3. do-while

for:

Syntax:

```
for(Expr1; Expr2;Expr3){
------
}
```



```
for(Expr1;   Expr2; Expr3)
{
     ---------
     ----------
     ----------
}
```

EX:

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

```
Expr1: 1 time
Expr2: 11 times
Expr3: 10 times
Body  : 10 times
```

In for loop, Expr1 is optional, it is possible to write for loop without Expr1, it is possible to write any statement like System.out.println() as Expr1, but the suggestion is to provide loop variable declaration and initialization as Expr1.

EX:
—--

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

    }
}
```

Status: No Compilation Error
OP: 0 - 9

EX:

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

    }
}
```

Status: No Compilation Error.
OP: Hello, 0-9

In the for loop, Expr1 is able to allow at most one declarative statement, if we provide more than one declarative statement either with the same data type or with the different data types then the compiler will raise an error.

EX:

```
class Test{
    public static void main(String[] args){
        for(int i = 0, float f = 0.0f; i < 10 && f < 10.0f; i++,
f++){
            System.out.println(i+"    "+f);
        }

    }
}
```

Status: Compilation Error
EX:

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

Status: Compilation Error

EX:

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

    }
}
```

Status: No Compilation Error

OP:

```
0    0
1    1
2    2
3    3
4    4
5    5
6    6
7    7
8    8
9    9
```

In for loop, Expr2 is optional, we can write for loop without expr2,in this case, by default for loop will take true value and it will make for loop as infinite loop. If we want to write any statement as Expr2 then it must be a conditional statement, it must return either true value or false value.

EX:

```
class Test{
    public static void main(String[] args){
        for(int i = 0; ; i++){
            System.out.println(i);
        }

    }
}
```

Status: No Compilation Error

OP: Infinite Output.

EX:

—--

```
class Test{
    public static void main(String[] args){
        for(int i = 0;System.out.println("Hello"); i++){
            System.out.println(i);
        }

    }
}
```

Status: Compilation Error.

EX:

```
class Test{
    public static boolean m1(){
        return true;
    }
    public static void main(String[] args){
        for(int i = 0; m1(); i++){
            System.out.println(i);
        }
    }
}
```

EX:

```
class Test{
    public static void main(String[] args){
        System.out.println("Before Loop");
        for(int i = 0; i <= 0 || i >= 0; i++){
            System.out.println("Inside Loop");
        }
        System.out.println("After Loop");
    }
}
```

Status: No Compilation Error

OP:

```
Before Loop
Inside Loop
–---
–---
Infinite times
```

EX:

```
class Test{
    public static void main(String[] args){
        System.out.println("Before Loop");
        for(int i = 0; true; i++){
            System.out.println("Inside Loop");
        }
        System.out.println("After Loop");
    }
}
```

Status: Compilation Error, Unreachable Statement

EX:

```
class Test{
    public static void main(String[] args){
        System.out.println("Before Loop");
        for(int i = 0; ; i++){
            System.out.println("Inside Loop");
        }
        System.out.println("After Loop");
    }
}
```

Status: Compilation Error, Unreachable Statement

Reason:
If we provide any statement immediately after infinite loop then that statement is “Unreachable Statement”, If the compiler knows the provided loop as an infinite loop and if compiler identifies any followed statement after the infinite loop then compiler will raise an error like Unreachable Statement, If the compiler does not know the provided loop as an infinite loop [really the loop is an infinite loop] and if compiler identifies the followed statement after the loop then compiler will not raise any error.

In general, the compiler will decide whether a loop is an infinite loop or not on the basis of the provided conditional expression, if the conditional expression is a constant conditional expression and its result is true then the compiler will decide the provided loop is an infinite loop and if it identifies any followed statement after the loop then compiler will raise an error, if the conditional expression is variable expression then compiler will not raise any error even though the provided loop is really an infinite loop.

EX:

```
class Test{
    public static void main(String[] args){
        System.out.println("Before Loop");
        for(int i = 0; false ; i++){
            System.out.println("Inside Loop");
        }
        System.out.println("After Loop");
    }
}
```

Status: Compilation Error, Unreachable Statement
Reason: Now Loop Body is unreachable.

In for loop Expr3 is optional, we can write for loop without the Expr3, in place of Expr3 we are able to write any statement like System.out.println(), always it is suggestible to provide loop variable increment or decrement statements as Expr3.

EX:

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

Status: No Compilation Error.

OP: 0….9

EX:

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

Status: No Compilation Error.

OP:

```
0
Hello
1
Hello
2
Hello
—-
—--
9
Hello
```

EX:

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

Status: Compilation Error

Reason: loop variables must not be declared as final variables, because loop variables must perform either increment or decrement over its value but final variables must not allow modifications on its value.

EX:

```
for(;;)
```

Status: Compilation Error

EX:

```
for(;;);
```

Status: No COmpilation Error

EX:

```
for(;;)
   ;
```

Status: No Compilation Error

EX:

```
for(;;){
}
```

Status: No COmpilation Error

EX:

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

    }
}
```

Status: No Compilation Error

Reason:
In for loop body if we provide single statement then curly braces are optional , if we provide no statement in for loop body then it is mandatory to provide either ; or curly braces, if violate this rule then compiler will raise an error.

Q)Write a Java program to display even numbers up to 100?
——
-
Ans:
—---

```
class Test{
    public static void main(String[] args){
        for(int i = 1; i <= 100; i++){
            if(i % 2 == 0){
                System.out.println(i);
            }
        }
    }
}
```

EX:
—-

```
class Test{
    public static void main(String[] args){
        for(int i = 1; i <= 100; i++){
            if(i % 2 != 0){
                System.out.println(i);
            }
        }
    }
}
```

Q)Write a Java program to check whether a number is a prime number or not?

—--

Ans:

—--

```
class Test{
    public static void main(String[] args){
        int num = 11;
        int count = 0;
        for(int i = 1; i <= num; i++){
            if(num % i == 0){
                count = count + 1;
            }
        }
        if(count == 2){
            System.out.println(num+" Is a Prime Number");
        }else{
            System.out.println(num+" Is not a Prime Number");
        }
    }
}
```

Q)Write a Java program to display all prime numbers up to 100?

—--

Ans:

—---

```
class Test{
    public static void main(String[] args){
        for(int num = 1; num <= 100; num++){
            int count = 0;
            for(int i = 1; i <= num; i++){
                if(num % i == 0){
                    count = count + 1;
                }
            }
            if(count == 2){
                System.out.println(num+" Is a Prime Number");
            }else{
                System.out.println(num+" Is not a Prime Number");
            }
        }
    }
}
```

Q)Write a Java program to display Fibonacci Series up to 50?

```
—------------------------------------------------------------------
--
0 1 1 2 3 5 8 13
```

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

        int num1 = 0;
        int num2 = 1;
        int num3 = 0;

        System.out.print(num1+" ");
        System.out.print(num2+" ");
        for(int i = 0; num3 < 100; i++){
            num3 = num1 + num2;
            if(num3 < 100){
                System.out.print(num3+" ");
                num1 = num2;
                num2 = num3;
            }
        }

    }
}
```

In Java applications, when we know the number of iterations over a loop body in advance before writing a loop there we will use for loop.

EX: If we want to display “Durga Software Solutions” 10 times .

```
for(int i = 1; i <= 10; i++){
    System.out.println(“Durga Software Solutions”);
}
```

EX: To read elements from an array we will use a for loop, because the number of loop iterations is the same as the number of elements existing in the array.

EX:

```
String[] str = {“AAA”,  “BBB”,  “CCC”,  “DDD”,  “EEE”,  “FFF”};
Sopln(str[0]);
Sopln(str[1]);
Sopln(str[2]);
Sopln(str[3]);
Sopln(str[4]);
Sopln(str[5]);
```

Here the number of instructions is increased, to reduce the number of instructions we have to use a for loop.

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


        String[] str = {"AAA", "BBB", "CCC", "DDD", "EEE", "FFF"};
        for(int index = 0; index < str.length; index++){
            System.out.println(str[index]);
        }
    }
}
```

To read elements from arrays if we use a for loop like above then we are able to get the following problems.

1. We have to manage a separate variable for looping purposes, we have to assign some memory.
2. At each and every iteration JVM must execute conditional expression, it is a Streangthfull expression, it may take more memory and more execution time.
3. At each and every iteration, JVM must perform either increment or decrement operation over the loop variable.
4. In this approach, we are able to read elements from the array by providing array index values explicitly, there may be a chance to get ArrayIndexoutOfBoundsException.

All the above problems may reduce Java application performance. To overcome all the above problems, JDK1.5 version has provided an enhancement in for loop that is “for-Each” Loop.

Syntax:

```
for(ArrayDataType varName: ArrayRefVar){
    –-----
}
```

EX:

```
int[] a = {10,20,30,40,50,60};
for(int i: a){
    System.out.println(i);
}
```

When JVM executes the above for-Each Loop , JVM will perform the following actions.

1. At each and every iteration, JVM will pick the value from the array.
2. JVM will assign current value to the variable which we declared in the for-Each loop.
3. JVM will execute a loop body.

JVM will repeat all the above steps up to the last element of the array.

EX:

–--

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

        String[] str = {"AAA", "BBB", "CCC", "DDD", "EEE", "FFF"};
        for(int index = 0; index < str.length; index++){
            System.out.println(str[index]);
        }
        System.out.println();

        for(String element: str){
            System.out.println(element);
        }
    }
}
```

Note: In Java applications, for-Each loop is applicable for reading elements from the arrays and from the Collections, but not for normal repetitions.

While loop:
—-----------
When we don't know the number of iterations over the loop body in advance before writing a loop there we will use while loop.
Syntax:

```
while(Condition){
    —--Instructions—---
}
```

EX:

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

        int i = 0;
        while(i < 10){
            System.out.println(i);
            i = i + 1;
        }
    }
}
```

EX:

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

        int i = 0;
        while(){
            System.out.println(i);
            i = i + 1;
        }
    }
}
```

Status: Compilation Error.

Reason: In the while loop, conditional expression is mandatory.

EX:

```
class Test{
    public static void main(String[] args){
        System.out.println("Before Loop");
        int i = 0;
        while(i <= 0 || i >= 0){
            System.out.println("Inside Loop");
            i = i + 1;
        }
        System.out.println("After Loop");
    }
}
```

Status: No Compilation Error

OP:

```
Before loop
Inside Loop
Inside Loop
—--
```

—--

EX:

—-

```
class Test{
    public static void main(String[] args){
        System.out.println("Before Loop");
        int i = 0;
        while(true){
            System.out.println("Inside Loop");
            i = i + 1;
        }
        System.out.println("After Loop");
    }
}
```

Status: Compilation Error, Unreachable Statement.

EX:

```
class Test{
    public static void main(String[] args){
        System.out.println("Before Loop");
        int i = 0;
        while(false){
            System.out.println("Inside Loop");
            i = i + 1;
        }
        System.out.println("After Loop");
    }
}
```

Status: Compilation Error, Unreachable Statement.

Q)Write a Java program to calculate factorial value for the given number?
—--

Ans:
—----

```
Fact(4) : 1*2*3*4
Fact(5) : 1*2*3*4*5

class Test{
    public static void main(String[] args){
        int num = 5;
        int result = 1;
        for(int i = 1; i <= num; i++){
            result = result * i;
        }
        System.out.println("Facrtorial of "+num+" Is : "+result);

        int result1 = 1;
        int i = 1;
        while(i <= num){
            result1 = result1 * i;
            i = i + 1;
        }
        System.out.println("Facrtorial of "+num+" Is : "+result1);

    }
}
```

do-while loop:
—---------------

Q)What are the differences between while and do-while loops?

—--

Ans:

—----

1. While loop syntax
   ```
   while(condition){
       —----Instructions—----
   }
   ```

   Do-while loop syntax:
   ```
   do{
       —----Instructions—----
   }while(Condition);
   ```

2. In the case of a while loop, first, conditional expression will be evaluated , if it returns true then body will be executed.

   In the case of a do-while loop, first, the body will be executed, after this conditional expression will be evaluated.

3. In the case of a while loop, Conditional Expression will be evaluated to perform current iteration.

   In the case of do-while loop, Conditional Expression will be evaluated to
perform the next iteration.

4. In the case of while loop there is no guarantee to execute loop body minimum one time.

In the case of do-while loop, there is a guarantee to execute loop body minimum one time.

```
EX:
----
class Test{
    public static void main(String[] args){
        int i = 0;
        do{
            System.out.println(i);
            i = i + 1;
        }while (i < 10);
    }
}
Status: No Compilation Error.
OP:
0 - 9
```

```
EX:
---
class Test{
    public static void main(String[] args){
        int i = 0;
        do{
            System.out.println(i);
            i = i + 1;
        }while();
    }
}
Status: Compilation Error.
```

Reason: Conditional Expression is mandatory in a do-while loop.

EX:
—-

```
class Test{
    public static void main(String[] args){
        int i = 0;
        do{
            System.out.println(i);
            i = i + 1;
        }while(i < 10)
    }
}
```

Status: Compilation Error.
Reason: ‘;’ is Mandatory in a do-while syntax.

EX:
—-

```
class Test{
    public static void main(String[] args){
        int i = 0;
        while(i < 10)
            System.out.println(i++);

    }
}
```

Status: No Compilation Error.
OP: 0 - 9

EX:

—--

```
int i = 0;
while(i < 10);
```

Status: No Compilation Error

EX:

—--

```
int i = 0;
while(i < 10)
    ;
```

Status: No Compilation Error

EX:

```
int i = 0;
while(i < 10)
```

Status: Compilation Error

EX:

—--

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

Status: No Compilation Error

OP: 0-9

EX:

—-

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

Status: No Compilation Error

OP: 0-9

EX:

```
class Test{
    public static void main(String[] args){
        int i = 0;
        do
        while(i < 10);
    }
}
```

Status: Compilation Error.

EX:

—-

```
class Test{
    public static void main(String[] args){
        int i = 0;
        do ;
        while(i < 10);
    }
}
```

Status: No Compilation Error

Reason: If we provide a single statement in the body then curly braces are optional, if we provide no statement in the body of the do-while loop then it is mandatory to provide either ; or curly braces.

```
EX:
—---
class Test{
    public static void main(String[] args){
        System.out.println("Before Loop");
        int i = 0;
        do{
            System.out.println("Inside Loop");
            i = i + 1;
        }while(i <= 0 || i >= 0);
        System.out.println("After Loop");
    }
}
Status: No Compilation Error
OP:
Before Loop
Inside Loop
Inside Loop
—-
—-
Infinite times

EX:
—-
class Test{
    public static void main(String[] args){
        System.out.println("Before Loop");
        int i = 0;
        do{
            System.out.println("Inside Loop");
            i = i + 1;
        }while(true);
        System.out.println("After Loop");
    }
}
```

Status: Compilation Error, Unreachable Statement.

EX:

—-

```
class Test{
    public static void main(String[] args){
        System.out.println("Before Loop");
        int i = 0;
        do{
            System.out.println("Inside Loop");
            i = i + 1;
        }while(false);
        System.out.println("After Loop");
    }
}
```

Status: No Compilation Error

OP:

```
Before Loop
Inside Loop
After Loop
```

Transfer Statements:
—-----------------------

These statements are able to bypass the flow of execution from one instruction to another instruction.

EX: break, continue, return

break:
—-----

‘break’ statement will be used inside loops and switch cases, it will terminate the loops , that is, it will bypass flow of execution to outside of the loop.

‘break’ statement will bypass flow of execution to outside of the loop by skipping remaining instructions in the current iteration and by skipping all the remaining iterations.

Syntax:

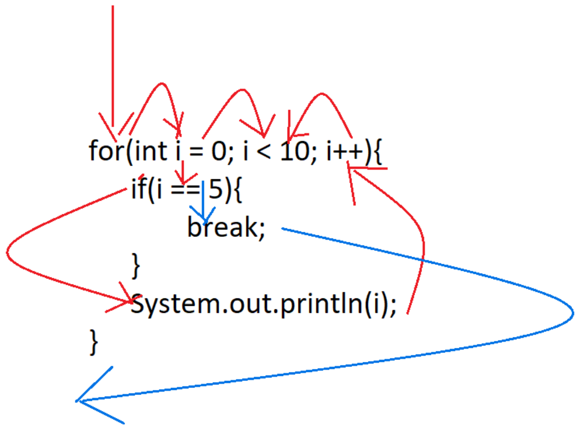
```
break;
```

EX:

```
class Test{
    public static void main(String[] args){
        for(int i = 0; i < 10; i++){
            if(i == 5){
                break;
            }
            System.out.println(i);
        }
    }
}
```

OP:

```
0
1
2
3
4
```



```
for(int i = 0; i < 10; i++){
    if(i == 5){
            break;
    }
    System.out.println(i);
}
```

```
0
1
2
3
4
```

EX:

```
class Test{
    public static void main(String[] args){
        System.out.println("Before loop");
        for(int i = 0; i < 10; i++){
            if(i == 5){
                System.out.println("Inside Loop, Before break");
                break;
                System.out.println("Inside Loop, After break");
            }
            System.out.println(i);
        }
        System.out.println("After Loop");
    }
}
```

Status: Compilation Error, Unreachable Statement.

Reason: If we provide any statement immediately after the ‘break’ statement then that statement is “Unreachable Statement”.

EX:

```
class Test{
    public static void main(String[] args){
        for(int i = 0; i < 10; i++){
            for(int j = 0; j < 10; j++){
                if(j == 5){
                    break;
                }
                System.out.println(i+"   "+j);
            }

        }
    }
}
```

Status: No Compilation Error

Reason:

Writing a loop inside another loop is called a “Nested Loop”, if we provide a ‘break’ statement in a nested loop then the provided ‘break’ statement will affect the nested loop only , it will not affect the outer loop.

In the above context, if we provide a ‘break’ statement in a nested loop and if we want to apply that break statement to a particular outer loop then we have to use a labeled break statement.

Syntax:

```
break label;
```

Where the provided label must be tagged with a particular outer loop.

EX:

```
class Test{
    public static void main(String[] args){
        l1:for(int i = 0; i < 10; i++){
            for(int j = 0; j < 10; j++){
                if(j == 5){
                    break l1;
                }
                System.out.println(i+"   "+j);
            }

        }

    }
}
```

Status: No Compilation Error

OP:

```
0   0
0   1
0   2
0   3
0   4
```

EX:

--

```
class Test{
    public static void main(String[] args){
        l1:for(int k = 0; k < 10; k++){
            System.out.println(k);
        }
        for(int i = 0; i < 10; i++){
            for(int j = 0; j < 10; j++){
                if(j == 5){
                    break l1;
                }
            }
        }




    }
}
```

Status: Compilation Error.

Reason: Label must be tagged with an outer loop , but not to some other loop.

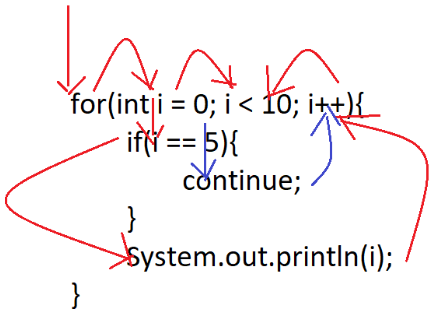
Continue:

'continue' statement will be used in loops, it will bypass the flow of execution to the next iteration by skipping all the remaining instructions in the current iteration.

EX:

```
class Test{
    public static void main(String[] args){
        for(int i = 0; i < 10; i++){
            if(i == 5){
                continue;
            }
            System.out.println(i);
        }
    }
}
```

```
OP:
0
1
2
3
4
6
7
8
9
```



```
for(int i = 0; i < 10; i++){
        if(i == 5){
                continue;
        }
        System.out.println(i);
}
```

0
1
2
3
4

6
7
8
9

```
EX:
----
class Test{
    public static void main(String[] args){
        System.out.println("Before Loop");
        for(int i = 0; i < 10; i++){
            if(i == 5){
                System.out.println("Inside loop, Before 
continue");
                continue;
                System.out.println("Inside Loop, After continue");
            }
            System.out.println(i);
        }
        System.out.println("After Loop");
    }
}
```

Status: Compilation Error, Unreachable Statement

EX:

```
class Test{
    public static void main(String[] args){
        for(int i = 0; i < 10; i++){
            for(int j = 0; j < 10; j++){
                if(j == 5){
                    continue;
                }
                System.out.println(i+"    "+j);
            }
        }
    }
}
```

Status: No Compilation Error

Note: If we provide a continue statement in a nested loop then the continue statement will affect the nested loop only , it will not affect the outer loop.

In the above context, if we provide a continue statement in a nested loop and if we want to affect the respective outer loop then we have to use the “labeled continue” statement.

EX:
—---

```
class Test{
    public static void main(String[] args){
        l1: for(int i = 0; i < 10; i++){
            for(int j = 0; j < 10; j++){
                if(j == 5){
                    continue l1;
                }
                System.out.println(i+"    "+j);
            }
        }
    }
}
```

Status: no Compilation Error

Arrays:
—--------

Array is an object , able to store more than one value of the same data type as per indexing.

There are two types of arrays

1. Single Dimensional Arrays
2. Multi Dimensional Arrays

Single Dimensional Arrays:

These arrays are able to represent all elements in a single row.

Syntax-1:

```
—---------
dataType[] var = {val1, val2,...val_n};
```

EX:

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

Syntax-2:

```
DataType[] var = new DataType[size];
Var[0] = val1;
Var[1] = val2;
—--
—--
Var[size-1] = val_n
```

EX:

```
int[] a = new int[4];
a[0] = 10;
a[1] = 20;
a[2] = 30;
a[3] = 40;
```

If we want to access an element from an array then we have to use the following syntax.

```
var[index]
```

EX: `System.out.println(a[2]);`

In Java, to get array size or length we have to use a predefined variable like “length”.

IN Java, every array is an object and every object has its own reference value, if we display the reference variable of the array then we are able to get value in the following format.

```
DimensionIN’[‘DataType@refValue.
[I@abc123 —> Single Dimensional Array
[[I@abc123 —> two dimensional Array
[[[I@abc123 —-> Three dimensional Array
```

While accessing an element from an array or while inserting an element in an array if we provide an index value which is outside range of the array's index value then JVM will raise an exception like java.lang.ArrayIndexOutOfBoundsException.

If we want to read elements from an array then we have to use for loop, even when compared with for loop it is suggestible to use for-Each loop.

EX:
—--

```
class Test{
    public static void main(String[] args){
        //int[] a = {10,20,30,40,50};
        int[] a = new int[5];
        a[0] = 10;
        a[1] = 20;
        a[2] = 30;
        a[3] = 40;
        a[4] = 50;

        System.out.println(a);// [I@abc123
        System.out.println(a.length);// 5
        System.out.println(a[2]);// 30
        System.out.println(a[4]);// 50
        //System.out.println(a[7]); --->
java.lang.ArrayIndexOutOfBoundsException
        System.out.println(a[a.length-3]);// 30
        System.out.println(a[a[0]-10]);// 10
        //System.out.println(a[a.length]); --->
java.lang.ArrayIndexOutOfBoundsException
        System.out.println();
        for(int index = 0; index < a.length; index++){
            System.out.println(a[index]);
        }
        System.out.println();
        for(int val: a){
            System.out.println(val);
        }
    }
}
```

Output:

```
[I@4617c264
5
30
50
30
10

10
20
30
40
50
```

```
10
20
30
40
50
```

Q)Write a Java program to display the sum of all the elements in an integer array?
—--

Ans:
—--

```
class Test{
    public static void main(String[] args){
        int[] a = {10,20,30,40,50};
        int result = 0;
        for(int i = 0; i < a.length; i++){
            result = result + a[i];
        }
        System.out.println("Result   : "+result);
    }
}
```

Q)Write a Java program to display the sum of all even numbers and sum of all odd numbers which exist in an integer array?
—--

Ans:
—--

```
class Test{
    public static void main(String[] args){
        int[] intArray = {5, 10, 15, 20, 25, 30, 35, 40, 45, 50};
        int evenResult = 0;
        int oddResult = 0;

        System.out.print("Array Elements  : ");
        for(int element: intArray){
            System.out.print(element+" ");
            if(element % 2 == 0){
                evenResult = evenResult + element;
            }else{
                oddResult = oddResult + element;
            }
        }
        System.out.println();
        System.out.println("SUM Of Even Numbers  : "+evenResult);
        System.out.println("SUM of ODD Numbers   : "+oddResult);
    }
}
```

```
Output:
D:\java6>javac Test.java
D:\java6>java Test
Array Elements  : 5 10 15 20 25 30 35 40 45 50
SUM Of Even Numbers  : 150
SUM of ODD Numbers   : 125
```

Q)Write a Java program to display the sum of all the numbers which are at even index values and the sum of all the numbers which exist in the ODD index values in an integer array?

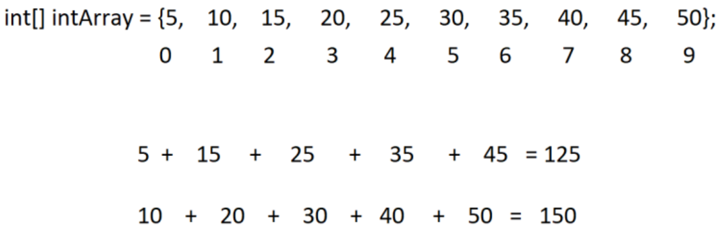
–--

Ans:

–--

```
class Test{
    public static void main(String[] args){
        int[] intArray = {5, 10, 15, 20, 25, 30, 35, 40, 45, 50};
        int evenResult = 0;
        int oddResult = 0;

        System.out.print("Array Elements  : ");
        for(int index = 0; index < intArray.length; index++){
            System.out.print(intArray[index]+" ");
            if(index % 2 == 0){
                evenResult = evenResult + intArray[index];
            }else{
                oddResult = oddResult + intArray[index];
            }
        }
        System.out.println();
        System.out.println("SUM Of Even Index Elements  :
"+evenResult);
        System.out.println("SUM of ODD INdex Elements   :
"+oddResult);
    }
}
```

```
Output:
D:\java6>javac Test.java
D:\java6>java Test
Array Elements  : 5 10 15 20 25 30 35 40 45 50
SUM Of Even Index Elements  : 125
SUM of ODD INdex Elements   : 150
```



```
int[] intArray = {5,   10,   15,   20,   25,   30,   35,   40,   45,   50};
                  0     1     2     3     4     5     6     7     8     9

             5 +  15   +   25   +   35   +  45  = 125

             10  +  20  +  30  +  40   +  50  =  150
```

Q)Write a Java program to copy all the elements of an array into another array in reverse?

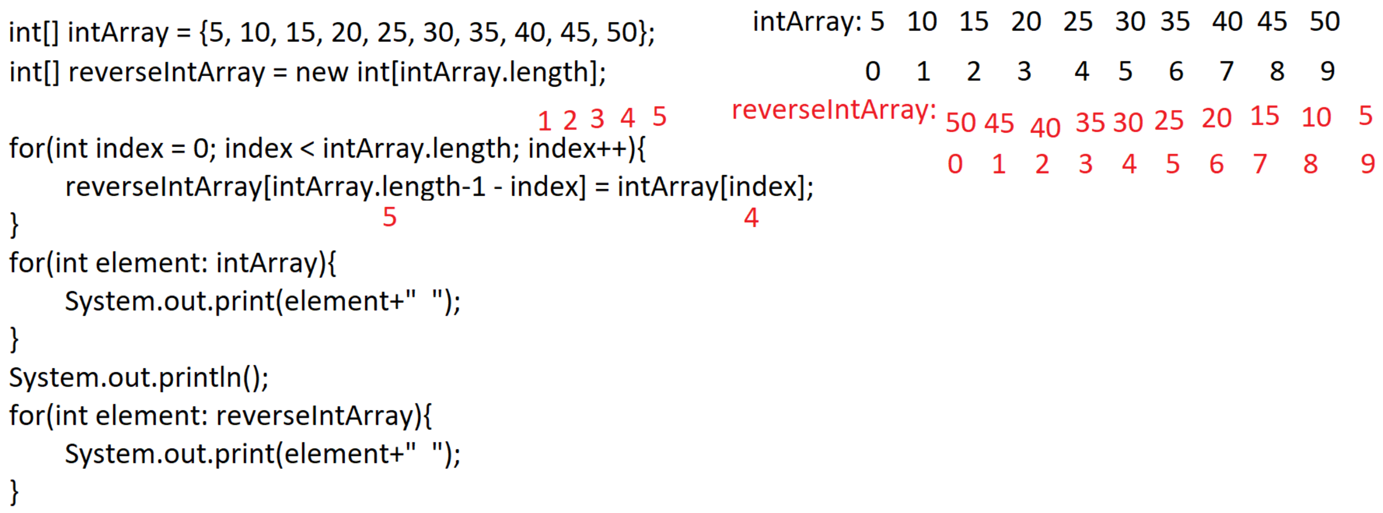
—--

Ans:

—---

```
class Test{
    public static void main(String[] args){
        int[] intArray = {5, 10, 15, 20, 25, 30, 35, 40, 45, 50};
        int[] reverseIntArray = new int[intArray.length];

        for(int index = 0; index < intArray.length; index++){
            reverseIntArray[intArray.length-1 - index] =
intArray[index];
        }
        for(int element: intArray){
            System.out.print(element+"  ");
        }
        System.out.println();
        for(int element: reverseIntArray){
            System.out.print(element+"  ");
        }
    }
}
```

```
Output:
D:\java6>javac Test.java
D:\java6>java Test
5   10   15   20   25   30   35   40   45   50
50   45   40   35   30   25   20   15   10   5
```



```
int[] intArray = {5, 10, 15, 20, 25, 30, 35, 40, 45, 50};
int[] reverseIntArray = new int[intArray.length];
                                                  1 2 3 4 5
for(int index = 0; index < intArray.length; index++){
      reverseIntArray[intArray.length-1 - index] = intArray[index];
}                                5                              4
for(int element: intArray){
      System.out.print(element+"  ");
}
System.out.println();
for(int element: reverseIntArray){
      System.out.print(element+"  ");
}
```

```
      intArray: 5  10  15  20  25  30  35  40  45  50
                0   1   2   3   4   5   6   7   8   9
reverseIntArray: 50 45 40 35 30 25 20 15 10  5
                  0  1  2  3  4  5  6  7  8  9
```

Q)Write a Java program to find max value and min value of an integer array?

```
—------------------------------------------------------------------
-------------------
Ans:
—--
class Test{
    public static void main(String[] args){

        int[] intArray = {9, 2, 10, 4, 7, 3, 8, 6, 5, 1};
        int temp = 0;

        for(int element: intArray){
            System.out.print(element+"  ");
        }
        System.out.println();

        for(int i = 0; i < intArray.length; i++){
            for(int j = i; j < intArray.length; j++){
                if(intArray[i] > intArray[j]){
                    temp = intArray[i];
                    intArray[i] = intArray[j];
                    intArray[j] = temp;
                }
            }
        }

        for(int element: intArray){
            System.out.print(element+"  ");
        }

        System.out.println();
        System.out.println("MIN VALUE   : "+intArray[0]);
        System.out.println("MAX VALUE   :
"+intArray[intArray.length-1]);
    }
}
```

Output:

```
D:\java6>javac Test.java
D:\java6>java Test
9  2  10  4  7  3  8  6  5  1
1  2  3  4  5  6  7  8  9  10
MIN VALUE   : 1
MAX VALUE   : 10
```

Q)Write a Java program to display all elements of an integer array in descending order?

—--

Ans:

—--

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

        int[] intArray = {9, 2, 10, 4, 7, 3, 8, 6, 5, 1};
        int temp = 0;

        for(int element: intArray){
            System.out.print(element+"  ");
        }
        System.out.println();

        for(int i = 0; i < intArray.length; i++){
            for(int j = i; j < intArray.length; j++){
                if(intArray[i] < intArray[j]){
                    temp = intArray[i];
                    intArray[i] = intArray[j];
                    intArray[j] = temp;
                }
            }
        }

        for(int element: intArray){
            System.out.print(element+"  ");
        }


    }
}
```

Q)Write a Java program to replace 4 th highest value with 100 in an integer array in descending order?

Consider the following array.

Int[] intArray = {5, 1, 8, 2, 7, 4, 9, 10, 6, 7, 3, 7};

—---

Ans:

—----

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

        int[] intArray = {5, 1, 8, 2, 7, 4, 9, 10, 6, 7, 3, 7};
        int temp = 0;

        for(int element: intArray){
            System.out.print(element+"  ");
        }
        System.out.println();

        for(int i = 0; i < intArray.length; i++){
            for(int j = i; j < intArray.length; j++){
                if(intArray[i] < intArray[j]){
                    temp = intArray[i];
                    intArray[i] = intArray[j];
                    intArray[j] = temp;
                }
            }
        }

        for(int element: intArray){
            System.out.print(element+"  ");
        }
        System.out.println();

        int num = intArray[3];
        for(int index = 0; index < intArray.length; index++){
            if(intArray[index] == num){
                intArray[index] = 100;
            }
        }

        for(int element: intArray){
            System.out.print(element+"  ");
        }

    }
}
```

Multi Dimensional Arrays:
—----------------------------

These arrays are able to represent data in more than one dimension.

Syntax-1:

```
DataType[][]...[] varName = {{{{...{val1,
val2,...},...},..},{{...,..},..},...},{{},{},....},....};
EX: int[][] intArray =
{{0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15}};
```

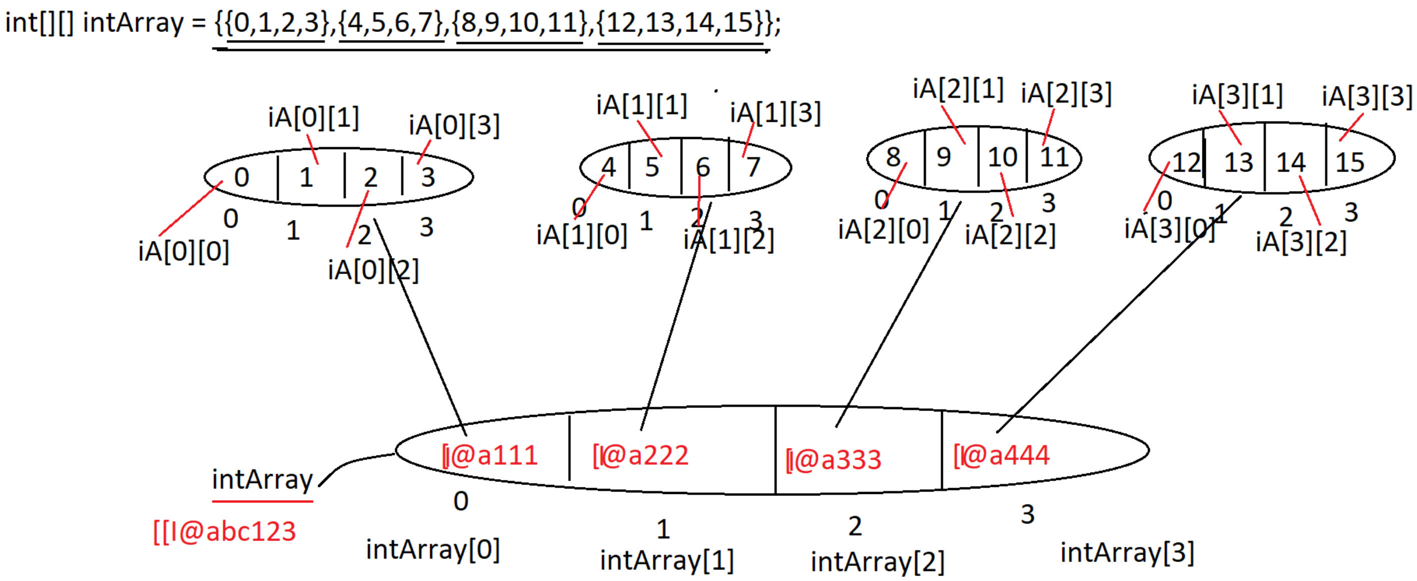
Syntax-2:

```
DataType[][]...[] varName = new
DataType[size_1][size_2].....[size_n];
varName[0][0]...[0] = val_1;
varName[0][0]...[1] = Val_2;
—---
—--
varName[0][0]...[size_n-1] = val_100;
—---
—-
varName[0][size_2-1]...[size_n-1] = val_200;
—-
—-
varName[size_1-1][size_2-1].....[size_n-1] = val_n;
```

EX:

```
int[][] intArray = new int[4][4];
intArray[0][0] = 0;
intArray[0][1] = 1;
intArray[0][2] = 2;
intArray[0][3] = 3;
intArray[1][0] = 4;
intArray[1][1] = 5;
```

```
intArray[1][2] = 6;
intArray[1][3] = 7;
intArray[2][0] = 8;
intArray[2][1] = 9;
intArray[2][2] = 10;
intArray[2][3] = 11;
intArray[3][0] = 12;
intArray[3][1] = 13;
intArray[3][2] = 14;
intArray[3][3] = 15;
```



```
class Test{
    public static void main(String[] args){
        int[][] intArray = 
{{0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15}};
        System.out.println(intArray);// [[I@abc123
        System.out.println(intArray[1]);// [I@a222
        System.out.println(intArray[3]);// [I@a444
        System.out.println(intArray.length);// 4
        System.out.println(intArray[0].length);// 4
        System.out.println(intArray[1].length +
intArray[3].length);// 8
        System.out.println(intArray[0][3]);// 3
        System.out.println(intArray[3][3]);// 15

    //System.out.println(intArray[2][4]);ArrayIndexOutOfBoundsExcep
tion

    //System.out.println(intArray[4][1]);ArrayIndexOutOfBoundsExcep
tion
        System.out.println(intArray[intArray[0].length-
1][intArray[1].length-2]);//14
        System.out.println(intArray[intArray.length-
3][intArray[0][2]]);//6
    }
}

EX:
—--
class Test{
    public static void main(String[] args){
        int[][] intArray =
{{0,1,2,3},{4,5,6,7},{8,9,10,11},{12,13,14,15}};

        for(int row = 0; row < intArray.length; row++){
            for(int column = 0; column < intArray[row].length;
column++){
                System.out.print(intArray[row][column]+"\t");
            }
            System.out.println();
        }
        System.out.println();

        for(int[] row: intArray){
            for(int element: row){
                System.out.print(element+"\t");
            }
            System.out.println();
        }
    }
}
```

Patterns:
—-----------
Q)Write a Java program to display the following Pattern?
—---
EX1:

```
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
```

```
class Test{
    public static void main(String[] args){
        for(int rows = 0; rows < 10; rows++){
            for(int column = 0; column < 10; column++){
                System.out.print("*"+" ");
            }
            System.out.println();
        }
    }
}
```

EX2:

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8 9
```

```
class Test{
    public static void main(String[] args){
        for(int rows = 0; rows < 10; rows++){
            for(int column = 0; column < 10; column++){
                System.out.print(column+" ");
            }
            System.out.println();
        }
    }
}
```

EX3:

```
9 8 7 6 5 4 3 2 1 0
```

```
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
9 8 7 6 5 4 3 2 1 0
```

```
class Test{
    public static void main(String[] args){
        for(int rows = 0; rows < 10; rows++){
            for(int column = 0; column < 10; column++){
                System.out.print((9-column)+" ");
            }
            System.out.println();
        }
    }
}
```

Or

```
class Test{
    public static void main(String[] args){
        for(int rows = 0; rows < 10; rows++){
            for(int column = 9; column >= 0; column--){
                System.out.print(column+" ");
            }
            System.out.println();
        }
    }
}
```

EX4:

```
0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2 2
3 3 3 3 3 3 3 3 3
4 4 4 4 4 4 4 4 4
5 5 5 5 5 5 5 5 5
6 6 6 6 6 6 6 6 6
7 7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10; column++){
                System.out.print(row+" ");
            }
            System.out.println();
        }
    }
}
```

EX5:

```
9 9 9 9 9 9 9 9 9 9
8 8 8 8 8 8 8 8 8 8
7 7 7 7 7 7 7 7 7 7
6 6 6 6 6 6 6 6 6 6
5 5 5 5 5 5 5 5 5 5
4 4 4 4 4 4 4 4 4 4
```

```
3 3 3 3 3 3 3 3 3 3
2 2 2 2 2 2 2 2 2 2
1  1 1  1 1  1  1 1  1 1
0 0 0 0 0 0 0 0 0 0
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10; column++){
                System.out.print((9-row)+" ");
            }
            System.out.println();
        }
    }
}
```

Or

```
class Test{
    public static void main(String[] args){
        for(int row = 9; row >= 0; row--){
            for(int column = 0; column < 10; column++){
                System.out.print(row+" ");
            }
            System.out.println();
        }
    }
}
```

EX6:

```
a b c d e f g h i j
a b c d e f g h i j
```

```
a b c d e f g h i j
a b c d e f g h i j
a b c d e f g h i j
a b c d e f g h i j
a b c d e f g h i j
a b c d e f g h i j
a b c d e f g h i j
a b c d e f g h i j
```

```java
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10; column++){
                System.out.print(((char)(97+column))+" ");
            }
            System.out.println();
        }
    }
}
```

EX7:

```
j i h g f e d c b a
j i h g f e d c b a
j i h g f e d c b a
j i h g f e d c b a
j i h g f e d c b a
j i h g f e d c b a
j i h g f e d c b a
j i h g f e d c b a
j i h g f e d c b a
j i h g f e d c b a
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10; column++){
                System.out.print(((char)(106-column))+" ");
            }
            System.out.println();
        }
    }
}
```

Or

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 106; column >= 97; column--){
                System.out.print(((char)column)+" ");
            }
            System.out.println();
        }
    }
}
```

EX8:

```
a a a a a a a a a a
b b b b b b b b b b
c c c c c c c c c c
d d d d d d d d d d
e e e e e e e e e e
f f f f f f f f f f
g g g g g g g g g g
```

```
h h h h h h h h h h
i i i i i i i i i i
j j j j j j j j j j
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10; column++){
                System.out.print(((char)(97+row))+" ");
            }
            System.out.println();
        }
    }
}
```

EX9:

```
j j j j j j j j j j
i i i i i i i i i i
h h h h h h h h h h
g g g g g g g g g g
f f f f f f f f f f
e e e e e e e e e e
d d d d d d d d d d
c c c c c c c c c c
b b b b b b b b b b
a a a a a a a a a a
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10; column++){
                System.out.print(((char)(106-row))+" ");
            }
            System.out.println();
        }
    }
}
```

Or

```
class Test{
    public static void main(String[] args){
        for(int row = 106; row >= 97; row--){
            for(int column = 0; column < 10; column++){
                System.out.print(((char)row)+" ");
            }
            System.out.println();
        }
    }
}
```

Q)Write a Java program to display the following pattern?

EX1:

```
*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
* * * * * * * * * *
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column <= row; column++){
                System.out.print("*"+" ");
            }
            System.out.println();
        }
    }
}
```

EX2:

```
0
0 1
0 1 2
0 1 2 3
0 1 2 3 4
0 1 2 3 4 5
0 1 2 3 4 5 6
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column <= row; column++){
                System.out.print(column+" ");
            }
            System.out.println();
        }
    }
}
```

EX3:

```
9
9 8
9 8 7
9 8 7 6
9 8 7 6 5
9 8 7 6 5 4
9 8 7 6 5 4 3
9 8 7 6 5 4 3 2
9 8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1 0
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column <= row; column++){
                System.out.print((9-column)+" ");
            }
            System.out.println();
        }
    }
}
```

EX4:

```
0
1 1
2 2 2
3 3 3 3
4 4 4 4 4
5 5 5 5 5 5
6 6 6 6 6 6 6
7 7 7 7 7 7 7 7
8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column <= row; column++){
                System.out.print(row+" ");
            }
            System.out.println();
        }
    }
}
```

EX5:

```
9
8 8
7 7 7
6 6 6 6
5 5 5 5 5
4 4 4 4 4 4
3 3 3 3 3 3 3
2 2 2 2 2 2 2 2
```

```
1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column <= row; column++){
                System.out.print((9-row)+" ");
            }
            System.out.println();
        }
    }
}
```

EX6:

```
a
a b
a b c
a b c d
a b c d e
a b c d e f
a b c d e f g
a b c d e f g h
a b c d e f g h i
a b c d e f g h i j
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column <= row; column++){
                System.out.print((char)(97+column)+" ");
            }
            System.out.println();
        }
    }
}
```

EX7:

```
j
j i
j i h
j i h g
j i h g f
j i h g f e
j i h g f e d
j i h g f e d c
j i h g f e d c b
j i h g f e d c b a
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column <= row; column++){
                System.out.print((char)(106-column)+" ");
            }
            System.out.println();
        }
    }
}
```

EX8:

```
a
b b
c c c
d d d d
e e e e e
f f f f f f
g g g g g g g
h h h h h h h h
i i i i i i i i i
j j j j j j j j j j
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column <= row; column++){
                System.out.print((char)(97+row)+" ");
            }
            System.out.println();
        }
    }
}
```

EX9:

```
j
i i
h h h
g g g g
f f f f f
e e e e e e
d d d d d d d
c c c c c c c c
```

```
b b b b b b b b b
a a a a a a a a a a
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column <= row; column++){
                System.out.print((char)(106-row)+" ");
            }
            System.out.println();
        }
    }
}
```

Q)Write a Java program to display the following pattern?

—---

EX1:

```
                       *
                      * *
                    * * *
                  * * * *
                * * * * *
              * * * * * *
            * * * * * * *
          * * * * * * * *
        * * * * * * * * *
      * * * * * * * * * *
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" "+" ");
            }
            for(int stars = 0; stars <= row; stars++){
                System.out.print("*"+" ");
            }
            System.out.println();
        }
    }
}
```

EX2:

```
                       0
                      0 1
                    0 1 2
                  0 1 2 3
                0 1 2 3 4
              0 1 2 3 4 5
            0 1 2 3 4 5 6
          0 1 2 3 4 5 6 7
        0 1 2 3 4 5 6 7 8
      0 1 2 3 4 5 6 7 8 9
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print(num+" ");
            }
            System.out.println();
        }
    }
}
```

EX3:

```
                  9
                9 8
              9 8 7
            9 8 7 6
          9 8 7 6 5
        9 8 7 6 5 4
      9 8 7 6 5 4 3
    9 8 7 6 5 4 3 2
  9 8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1 0
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((9-num)+" ");
            }
            System.out.println();
        }
    }
}
```

EX4:

```
                  0
                1 1
              2 2 2
            3 3 3 3
          4 4 4 4 4
        5 5 5 5 5 5
      6 6 6 6 6 6 6
    7 7 7 7 7 7 7 7
  8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print(row+" ");
            }
            System.out.println();
        }
    }
}
```

EX5:

```
                  9
                8 8
              7 7 7
            6 6 6 6
          5 5 5 5 5
        4 4 4 4 4 4
      3 3 3 3 3 3 3
    2 2 2 2 2 2 2 2
  1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((9-row)+" ");
            }
            System.out.println();
        }
    }
}
```

EX6:

```
                  a
                a b
              a b c
            a b c d
          a b c d e
        a b c d e f
      a b c d e f g
    a b c d e f g h
  a b c d e f g h i
a b c d e f g h i j
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((char)(97+num)+" ");
            }
            System.out.println();
        }
    }
}
```

EX7:

```
                  j
                j i
              j i h
            j i h g
          j i h g f
        j i h g f e
      j i h g f e d
    j i h g f e d c
  j i h g f e d c b
j i h g f e d c b a
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((char)(106-num)+" ");
            }
            System.out.println();
        }
    }
}
```

EX8:

```
                  a
                b b
              c c c
            d d d d
          e e e e e
```

```
        f f f f f f
      g g g g g g g
    h h h h h h h h
  i i i i i i i i i
j j j j j j j j j j
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((char)(97+row)+" ");
            }
            System.out.println();
        }
    }
}
```

EX9:

```
                  j
                i i
              h h h
            g g g g
          f f f f f
        e e e e e e
      d d d d d d d
    c c c c c c c c
  b b b b b b b b b
a a a a a a a a a a
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((char)(106-row)+" ");
            }
            System.out.println();
        }
    }
}
```

Q)Write a Java program to display the following pattern?

—---

EX1:

```
* * * * * * * * * *
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * * *
* * * * *
* * * *
* * *
* *
*
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10-row; column++){
                System.out.print("*"+" ");
            }
            System.out.println();
        }
    }
}
```

EX2:

```
0 1 2 3 4 5 6 7 8 9
0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7
0 1 2 3 4 5 6
0 1 2 3 4 5
0 1 2 3 4
0 1 2 3
0 1 2
0 1
0
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10-row; column++){
                System.out.print(column+" ");
            }
            System.out.println();
        }
    }
}
```

EX3:

```
9 8 7 6 5 4 3 2 1 0
```

```
9 8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2
9 8 7 6 5 4 3
9 8 7 6 5 4
9 8 7 6 5
9 8 7 6
9 8 7
9 8
9
```

```java
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10-row; column++){
                System.out.print((9-column)+" ");
            }
            System.out.println();
        }
    }
}
```

EX4:

```
0 0 0 0 0 0 0 0 0 0
1 1 1 1 1 1 1 1 1
2 2 2 2 2 2 2 2
3 3 3 3 3 3 3
4 4 4 4 4 4
5 5 5 5 5
6 6 6 6
7 7 7
8 8
9
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10-row; column++){
                System.out.print(row+" ");
            }
            System.out.println();
        }
    }
}
```

EX5:

```
9 9 9 9 9 9 9 9 9 9
8 8 8 8 8 8 8 8 8
7 7 7 7 7 7 7 7
6 6 6 6 6 6 6
5 5 5 5 5 5
4 4 4 4 4
3 3 3 3
2 2 2
1 1
0
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10-row; column++){
                System.out.print((9-row)+" ");
            }
            System.out.println();
        }
    }
}
```

EX6:

```
a b c d e f g h i j
a b c d e f g h i
a b c d e f g h
a b c d e f g
a b c d e f
a b c d e
a b c d
a b c
a b
a
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10-row; column++){
                System.out.print((char)(97+column)+" ");
            }
            System.out.println();
        }
    }
}
```

EX7:

```
j i h g f e d c b a
j i h g f e d c b
j i h g f e d c
j i h g f e d
j i h g f e
j i h g f
```

```
j i h g
j i h
j i
j
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10-row; column++){
                System.out.print((char)(106-column)+" ");
            }
            System.out.println();
        }
    }
}
```

EX8:

```
a a a a a a a a a a
b b b b b b b b b
c c c c c c c c
d d d d d d d
e e e e e e
f f f f f
g g g g
h h h
i i
j
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10-row; column++){
                System.out.print((char)(97+row)+" ");
            }
            System.out.println();
        }
    }
}
```

EX9:

```
j j j j j j j j j j
i i i i i i i i i
h h h h h h h h
g g g g g g g
f f f f f f
e e e e e
d d d d
c c c
b b
a
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int column = 0; column < 10-row; column++){
                System.out.print((char)(106-row)+" ");
            }
            System.out.println();
        }
    }
}
```

Q)Write a Java program to display the following pattern?

—--

EX1:

```
* * * * * * * * * *
  * * * * * * * * *
    * * * * * * * *
      * * * * * * *
        * * * * * *
          * * * * *
            * * * *
              * * *
                * *
                  *
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < row; spaces++){
                System.out.print(" "+" ");
            }
            for(int stars = 0; stars < 10-row; stars++){
                System.out.print("*"+" ");
            }
            System.out.println();
        }
    }
}
```

EX2:

```
0 1 2 3 4 5 6 7 8 9
  0 1 2 3 4 5 6 7 8
    0 1 2 3 4 5 6 7
```

```
      0 1 2 3 4 5 6
        0 1 2 3 4 5
          0 1 2 3 4
            0 1 2 3
              0 1 2
                0 1
                  0
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num < 10-row; num++){
                System.out.print(num+" ");
            }
            System.out.println();
        }
    }
}
```

EX3:

```
9 8 7 6 5 4 3 2 1 0
  9 8 7 6 5 4 3 2 1
    9 8 7 6 5 4 3 2
      9 8 7 6 5 4 3
        9 8 7 6 5 4
          9 8 7 6 5
            9 8 7 6
              9 8 7
                9 8
                  9
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num < 10-row; num++){
                System.out.print((9-num)+" ");
            }
            System.out.println();
        }
    }
}
```

EX4:

```
0 0 0 0 0 0 0 0 0 0
  1 1 1 1 1 1 1 1 1
    2 2 2 2 2 2 2 2
      3 3 3 3 3 3 3
        4 4 4 4 4 4
          5 5 5 5 5
            6 6 6 6
              7 7 7
                8 8
                  9
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num < 10-row; num++){
                System.out.print(row+" ");
            }
            System.out.println();
        }
    }
}
```

EX5:

```
9 9 9 9 9 9 9 9 9 9
  8 8 8 8 8 8 8 8 8
    7 7 7 7 7 7 7 7
      6 6 6 6 6 6 6
        5 5 5 5 5 5
          4 4 4 4 4
            3 3 3 3
              2 2 2
                1 1
                  0
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num < 10-row; num++){
                System.out.print((9-row)+" ");
            }
            System.out.println();
        }
    }
}
```

EX6:

```
a b c d e f g h i j
  a b c d e f g h i
    a b c d e f g h
      a b c d e f g
        a b c d e f
          a b c d e
            a b c d
              a b c
                a b
                  a
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num < 10-row; num++){
                System.out.print((char)(97+num)+" ");
            }
            System.out.println();
        }
    }
}
```

EX7:

```
j i h g f e d c b a
  j i h g f e d c b
    j i h g f e d c
      j i h g f e d
        j i h g f e
```

```
          j i h g f
            j i h g
              j i h
                j i
                  j
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num < 10-row; num++){
                System.out.print((char)(106-num)+" ");
            }
            System.out.println();
        }
    }
}
```

EX8:

```
a a a a a a a a a a
  b b b b b b b b b
    c c c c c c c c
      d d d d d d d
        e e e e e e
          f f f f f
            g g g g
              h h h
                i i
                  j
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num < 10-row; num++){
                System.out.print((char)(97+row)+" ");
            }
            System.out.println();
        }
    }
}
```

EX9:

```
j j j j j j j j j j
 i i i i i i i i i
  h h h h h h h h
   g g g g g g g
    f f f f f f
     e e e e e
      d d d d
       c c c
        b b
         a
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < row; spaces++){
                System.out.print(" "+" ");
            }
            for(int num = 0; num < 10-row; num++){
                System.out.print((char)(106-row)+" ");
            }
            System.out.println();
        }
    }
}
```

Q)Write a Java program to display the following pattern?

```
—-----------------------------------------------------------
          *
         * *
        * * *
       * * * *
      * * * * *
     * * * * * *
    * * * * * * *
   * * * * * * * *
  * * * * * * * * *
 * * * * * * * * * *
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" ");
            }
            for(int stars = 0; stars <= row; stars++){
                System.out.print("*"+" ");
            }
            System.out.println();
        }
    }
}
```

EX2:

```
        0
        0 1
       0 1 2
      0 1 2 3
     0 1 2 3 4
    0 1 2 3 4 5
   0 1 2 3 4 5 6
  0 1 2 3 4 5 6 7
 0 1 2 3 4 5 6 7 8
0 1 2 3 4 5 6 7 8 9
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print(num+" ");
            }
            System.out.println();
        }
    }
}
```

EX3:

```
         9
        9 8
       9 8 7
      9 8 7 6
     9 8 7 6 5
```

```
    9 8 7 6 5 4
   9 8 7 6 5 4 3
  9 8 7 6 5 4 3 2
 9 8 7 6 5 4 3 2 1
9 8 7 6 5 4 3 2 1 0
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((9-num)+" ");
            }
            System.out.println();
        }
    }
}
```

EX4:

```
         0
        1 1
       2 2 2
      3 3 3 3
     4 4 4 4 4
    5 5 5 5 5 5
   6 6 6 6 6 6 6
  7 7 7 7 7 7 7 7
 8 8 8 8 8 8 8 8 8
9 9 9 9 9 9 9 9 9 9
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print(row+" ");
            }
            System.out.println();
        }
    }
}
```

EX5:

```
         9
        8 8
       7 7 7
      6 6 6 6
     5 5 5 5 5
    4 4 4 4 4 4
   3 3 3 3 3 3 3
  2 2 2 2 2 2 2 2
 1 1 1 1 1 1 1 1 1
0 0 0 0 0 0 0 0 0 0
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((9-row)+" ");
            }
            System.out.println();
        }
    }
}
```

EX6:

```
         a
        a b
       a b c
      a b c d
     a b c d e
    a b c d e f
   a b c d e f g
  a b c d e f g h
 a b c d e f g h i
a b c d e f g h i j
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((char)(97+num)+" ");
            }
            System.out.println();
        }
    }
}
```

EX7:

```
         j 
        j i 
       j i h 
      j i h g 
     j i h g f 
    j i h g f e 
   j i h g f e d 
  j i h g f e d c 
 j i h g f e d c b 
j i h g f e d c b a 
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((char)(106-num)+" ");
            }
            System.out.println();
        }
    }
}
```

EX8:

```
         a 
        b b 
       c c c 
      d d d d 
```

```
     e e e e e
    f f f f f f
   g g g g g g g
  h h h h h h h h
 i i i i i i i i i
j j j j j j j j j j
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((char)(97+row)+" ");
            }
            System.out.println();
        }
    }
}
```

EX9:

```
         j
        i i
       h h h
      g g g g
     f f f f f
    e e e e e e
   d d d d d d d
  c c c c c c c c
 b b b b b b b b b
a a a a a a a a a a
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" ");
            }
            for(int num = 0; num <= row; num++){
                System.out.print((char)(106-row)+" ");
            }
            System.out.println();
        }
    }
}
```

Q)Write a Java program to display the following pattern?
—--

EX1:

```
    *
   ***
  *****
 *******
*********
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 5; row++){
            for(int spaces = 0; spaces < 4 - row; spaces++){
                System.out.print(" ");
            }
            for(int stars = 0; stars < (2*row+1); stars++){
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

Q)Write a Java program to display the following pattern?

—--

EX1:

```
       **
      ****
     ******
    ********
   **********
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 5; row++){
            for(int spaces = 0; spaces < 4 - row; spaces++){
                System.out.print(" ");
            }
            for(int stars = 0; stars < (2*row + 2); stars++){
                System.out.print("*");
            }
            System.out.println();
        }
    }
}
```

Q)Write a Java program to display the following pattern?

—---

EX1:

```
* * * * * * * * * *
 * * * * * * * * *
  * * * * * * * *
   * * * * * * *
    * * * * * *
     * * * * *
      * * * *
       * * *
        * *
         *
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < row; spaces++){
                System.out.print(" ");
            }
            for(int stars = 0; stars < 10 - row; stars++){
                System.out.print("*"+" ");
            }
            System.out.println();
        }
    }
}
```

```
Q)Write a Java program to display the following pattern?
—--------------------------------------------------------
EX1:
—--
         *
        * *
       * * *
      * * * *
     * * * * *
    * * * * * *
   * * * * * * *
  * * * * * * * *
 * * * * * * * * *
* * * * * * * * * *
 * * * * * * * * *
  * * * * * * * *
   * * * * * * *
    * * * * * *
     * * * * *
      * * * *
       * * *
        * *
         *
EX:
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 9; row++){
            for(int spaces = 0; spaces < 9-row; spaces++){
                System.out.print(" ");
            }
            for(int stars = 0; stars <= row; stars++){
                System.out.print("*"+" ");
            }
            System.out.println();
        }
        for(int row = 0; row < 10; row++){
            for(int spaces = 0; spaces < row; spaces++){
                System.out.print(" ");
            }
            for(int stars = 0; stars < 10 - row; stars++){
                System.out.print("*"+" ");
            }
            System.out.println();
        }
    }
}
```

Or

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 20; row++){
            if(row < 10){
                for(int spaces = 0; spaces < 9-row; spaces++){
                    System.out.print(" ");
                }
                for(int stars = 0; stars <= row; stars++){
                    System.out.print("*"+" ");
                }
                System.out.println();
            }else{
                for(int spaces = 0; spaces < (row-9); spaces++){
                System.out.print(" ");
                }
                for(int stars = 0; stars < 10 - (row-9); stars++){
                    System.out.print("*"+" ");
                }
                System.out.println();
            }
        }

    }
}
```

Q)Write a Java program to display the following pattern?

—--

```
        *
       ***
      *****
     *******
    *********
     *******
      *****
       ***
        *
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 5; row++){
            for(int spaces = 0; spaces < 4-row; spaces++){
                System.out.print(" ");
            }
            for(int stars = 0; stars < 2*row+1; stars++){
                System.out.print("*");
            }
            System.out.println();
        }
        for(int row = 0; row < 4; row++){
            for(int spaces = 0; spaces <= row; spaces++){
                System.out.print(" ");
            }
            for(int stars = 0; stars < (7-2*row); stars++){
                System.out.print("*");
            }
            System.out.println();
        }

    }
}
```

Or

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 9; row++){
            if(row < 5){
                for(int spaces = 0; spaces < 4-row; spaces++){
                    System.out.print(" ");
                }
                for(int stars = 0; stars < 2*row+1; stars++){
                    System.out.print("*");
                }
                System.out.println();
            }else{
                for(int spaces = 5; spaces <= row; spaces++){
                    System.out.print(" ");
                }
                for(int stars = 0; stars < (17-2*row); stars++){
                    System.out.print("*");
                }
                System.out.println();
            }
        }







    }
}
```

Q)Write a Java program to display the following pattern?

```
—---------------------------------------------------------
        **
       ****
      ******
     ********
    **********
     ********
      ******
       ****
        **
```

```
class Test{
    public static void main(String[] args){
        for(int row = 0; row < 5; row++){
            for(int spaces = 0; spaces < 4-row; spaces++){
                System.out.print(" ");
            }
            for(int stars = 0; stars < 2*row+2; stars++){
                System.out.print("*");
            }
            System.out.println();
        }

        for(int row = 0; row < 4; row++){
            for(int spaces = 0; spaces <= row; spaces++){
                System.out.print(" ");
            }
            for(int stars = 0; stars < (8 - 2*row); stars++){
                System.out.print("*");
            }
            System.out.println();
        }


    }
}
```

Or

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

        for(int row = 0; row < 9; row++){
            if(row < 5){
                for(int spaces = 0; spaces < 4-row; spaces++){
                    System.out.print(" ");
                }
                for(int stars = 0; stars < 2*row+2; stars++){
                    System.out.print("*");
                }
                System.out.println();
            }else{
                for(int spaces = 5; spaces <= row; spaces++){
                    System.out.print(" ");
                }
                for(int stars = 0; stars < (18 - 2*row); stars++){
                    System.out.print("*");
                }
                System.out.println();
            }
        }

    }
}
```

Q)Write a Java program to display the following pattern?

```
----------------------------------------------------------
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
* * * * * * * * * *
      * * * *
      * * * *
      * * * *
      * * * *
      * * * *
```

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

        for(int row = 0; row < 9; row++){
            if(row < 5){
                for(int stars = 0; stars < 10; stars++){
                    System.out.print("*"+" ");
                }
                System.out.println();
            }else{
                for(int spaces = 5; spaces < 8; spaces++){
                    System.out.print(" "+" ");
                }
                for(int stars = 0; stars < 4; stars++){
                    System.out.print("*"+" ");
                }
                System.out.println();
            }
        }








    }
}
```

Q)Write a Java program to display the following pattern?

—--

```
              *
             * *
            * * *
           * * * *
          * * * * *
         * * * * * *
        * * * * * * *
       * * * * * * * *
      * * * * * * * * *
     * * * * * * * * * *
           * * * *
           * * * *
           * * * *
           * * * *
           * * * *
```

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

        for(int row = 0; row < 15; row++){
            if(row < 10){
                for(int spaces = 0; spaces < 9-row; spaces++){
                    System.out.print(" ");
                }
                for(int stars = 0; stars <= row; stars++){
                    System.out.print("*"+" ");
                }
                System.out.println();
            }else{
                for(int spaces = 0; spaces < 3; spaces++){
                    System.out.print(" "+" ");
                }
                for(int stars = 0; stars < 4; stars++){
                    System.out.print("*"+" ");
                }
                System.out.println();
            }
        }
    }
}
```

Q)Write a Java program to display the following pattern?

```
                       Durga
                    Durga Durga
                  Durga    Durga
                Durga       Durga
              Durga          Durga
            Durga             Durga
```

OOPS:
----

durgasoftoffline@gmail.com
Mobile: 7386095600

Advantages of this Course:

—------------------------------

1. All My classes videos records will be provided up to 1 year
2. All my classes material in the form of Hard copy[chargble] for Offline and softcopy for online students.
3. Course completion certificate will be provided at the end of the course.
4. Company drives will be provided for All the
5. Concepts wise Written tests.
6. Mock interviews
7. Working with Multiple databases
    Oracle, MySQL,....
8. Working with Multiple Servers
    a. Weblogic
    b. Wildfly
    c. Glashfish
    d. Tomcat
9. Working with Multiple IDEs
    a. Eclipse
    b. IntelliJ Idea
    c. Netbeans
10. Interview Questions.
11. Up to Latest version[JAVA 18, 19]

6:00PM to 7:30PM

Core Java

-
-
-

Hibernate

Webservices

7:30PM to 9:00PM

Adv Java

Spring
Spring Boot
Microservice

Batch#1

Core Java

3 months

Batch#2

Basic Java Package

1. Core Java
2. Adv Java
3. Oracle
4. Basic UI

4 to 5

Batch#3

Complete Java Package

1. Core Java
2. Adv Java
3. Hibernate
4. Spring
5. Spring Boot
6. Microservices
7. Webservices
8. Oracle
9. Basic UI

5 to 6

Batch#4

Full stacl Package

1. Core Java
2. Adv Java
3. Hibernate
4. Spring
5. Spring Boot
6. Microservices
7. Webservices
8. Oracle
9. Basic UI
10. Angular/React JS
11. Linux
12. AWS
13. Devops
14. Testing Tools

7 to 8

Offline : 8852 8852 63/64/96/97

Online: 7386095600,8885252627, 9246212143,

Online mail: durgasoftonlinetraining@gmail.com
Website: www.durgasoft.com

https://tinyurl.com/corejava6pmnotes

Please check in Handouts part in Webinar
Download question paper from webinar.

For Answer sheets,Take notepad , at top write u r name, mobile number , admission number,...

Next write answers along with question numbers…….
Send mail to durgasoftoffline@gmail.com