



# CORE JAVA

## Material

By

**MR. NAGOOR BABU**



# **INDEX**

|  |     |
|--|-----|
| 1) Introduction .....                            | 5   |
| 2) Steps to Prepare First Java Application ..... | 33  |
| 3) Language Fundamentals .....                   | 43  |
| 4) Patterns .....                                | 101 |
| 5) OOPs .....                                    | 119 |
| 6) Inner Classes .....                           | 268 |
| 7) Wrapper Classes .....                         | 287 |
| 8) Packages .....                                | 292 |
| 9) String Manipulations .....                    | 305 |
| 10) Exception Handling .....                     | 318 |
| 11) Multi Threading .....                        | 345 |
| 12) IO Streams .....                             | 369 |
| 13) Networking .....                             | 400 |
| 14) RMI [Remote Method Invocation] .....         | 406 |
| 15) Collections .....                            | 415 |
| 16) Generics .....                               | 479 |
| 17) Graphical User Interface (GUI) .....         | 489 |
| 18) Internationalization (I18N) .....            | 528 |
| 19) Reflection API .....                         | 534 |



---

|                               |     |
|-------------------------------|-----|
| 20) Annotations .....         | 543 |
| 21) Regular Expressions ..... | 556 |
| 22) Garbage Collection .....  | 568 |
| 23) JVM Architecture .....    | 579 |
| 24) JDBC Basics .....         | 588 |
| 25) Java 8 Features .....     | 607 |
| 26) Java 9 Features .....     | 644 |
| 27) Java 10 Features .....    | 798 |
| 28) Java 11 Features .....    | 812 |
| 29) Java 12 Features .....    | 827 |
| 30) Java 13 Features .....    | 843 |
| 31) Java 14 Features .....    | 857 |



# Introduction



# Java History

## Java Details

**Home :** SUN Mc Systems (Oracle Corporation)

**Author :** James Gosling

**Objective :** To prepare simple electronic consumer goods.

**Project :** Green

**First Version :** JDK 1.0 (1996, Jan-23<sup>rd</sup>)

**Used Version :** Some org JDK5.0, Some other JAVA 6, JAVA 7

**Latest Version :** JAVA7, JAVA8, This April – JAVA 9

**Type of Software :** Open Source Software

**Strong Features :** Object-oriented, Platform Independent, Robust, Portable, Dynamic, Secure.....

| Version                  | Code Name  | Enhancements   |
|--------------------------|------------|--|
| 1) JDK1.0[Jan,23,1996]   | OAK        | Language Introduction  |
| 2) JDK1.1[Feb,19,1997]   | ----       | RMI, JDBC, Reflection API, Java Beans, Inner classes                               |
| 3) JDK1.2[Dec,8,1998]    | Playground | Strictfp, Swing, CORBA, Collection, Framework                                      |
| 4) JDK1.3[May,8,2000]    | Kestrel    | Updations on RMI, JNDI   |
| 5) JDK1.4[Feb,6,2002]    | Merlin     | Regular Expression, NIO, assert, Keyword, JAXP, ...                                |
| 6) JDK5.0[Sep,30,2004]   | Tiger      | Autoboxing, var-arg method, static import, Annotations ,..                         |
| 7) JAVA SE6[Dec,11,2006] | Mustang    | JDBC4.0, GUI updations, Console  |
| 8) JAVA SE7[Jul,28,2011] | Dolphin    | Strings in switch, '_' symbol in literals, try-with- resources                     |
| 9) JAVA SE8[Mar,18,2014] | Spider     | Interface improvements, Lambda Expression, Date-Time API, Updations on Collections |



|                          |       |   |
|--------------------------|-------|---|
| 10)JAVA SE9[Sep,20017]   | ----  | JSHELL, JPMS,Private Methods in Interfaces, .....   |
| 11)JAVA SE10[March,2018] | ----  | Local Variables Type Inference, GarbageCollector interface, Application Class Data Sharing,.... |
| 12)JAVA SE11[Sep,2018]   | ----  | HttpClient,Local Variables Syntax for Lambda Parameter,....                                     |
| 13)JAVA SE12[March,2019] | ----  | Switch Expressions, JVM Constants,....  |
| 14)JAVA SE13[Sep,2019]   | ----  | Text Blocks, Switch Expressions Updations, Dynamic CDS Achieves.....                            |
| 15)JAVA SE14[March,2020] | ----  | Pattern Matching For instanceof, Records, Text Blocks Updations,.....                           |
| 16)JAVA SE15[Sep,2020]   | ----- | Updations on Text Blocks, Pattern Matching for instanceof operator, ....                        |



# Differences between Java and Others [C and C++]

## 1) C and C++ are static programming languages but JAVA is dynamic programming language:

If any programming language allows memory allocation for primitive data types at compilation time [Static 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 primitive data types at compilation time only, not at runtime.

If any programming language allows memory allocation for primitive data types at runtime, not at compilation time then that programming language is called as Dynamic Programming Language.

EX: JAVA

In java applications, memory will be allocated for primitive data types at runtime only, not at compilation time.

Note: In Java applications, memory will be allocated for primitive data types at the time of creating objects only, in java applications, objects are created at runtime only.

## 2) Pre-Processor is required in C and C++, but, Pre-Processor is not required in Java:

In case of C and C++, the complete predefined library is provided in the form of header files

EX:

stdio.h

conio.h

math.h

---

----



If we want to use predefined library in C and C++ applications, we have to include header files in C and C++ applications, for this, we have to use #include<> statement.

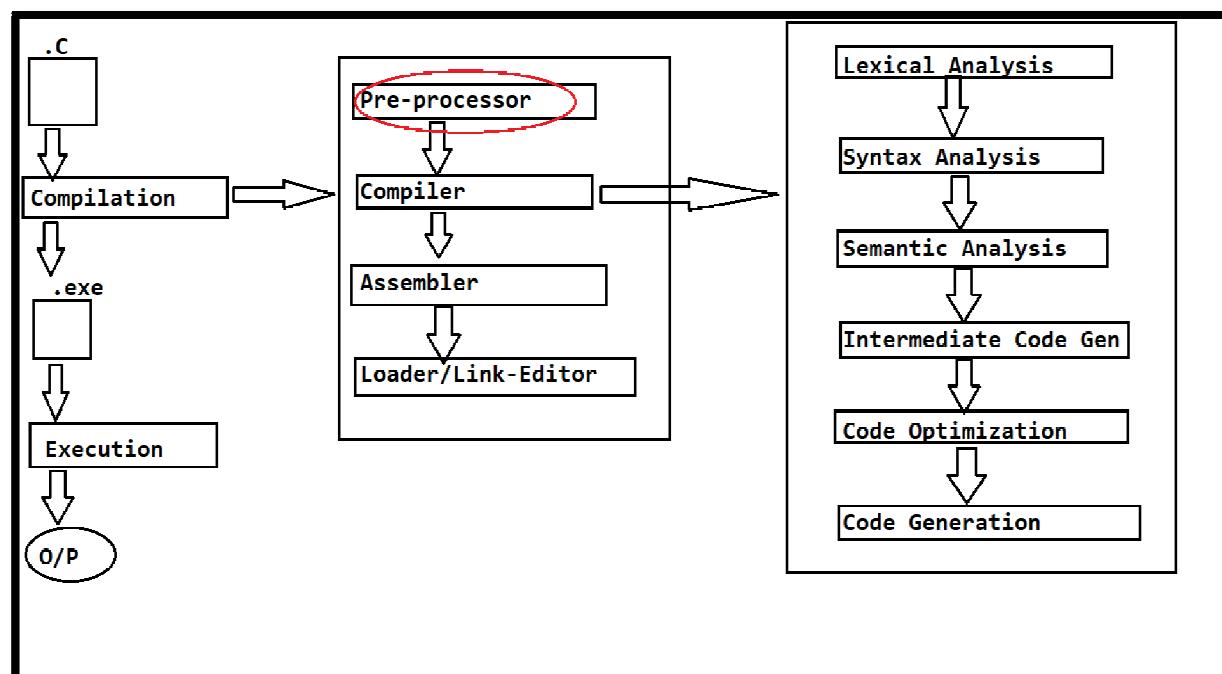
**EX:**

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

If we compile C and C++ applications then Pre-Processor will perform the following actions.

- 1) Pre-Processor will recognize all #include<> statement.
- 2) Pre-Processor will take all the specified header files from #include<> statements.
- 3) Pre-Processor will check whether the specified header files are existed or not in C and C++ softwares.
- 4) If the specified header files are not existed the Pre-Processor will generate some error messages.
- 5) If the specified header files are existed then Pre-Processor will load the specified header files to the memory, this type of loading predefined library at compilation time is called as "Static Loading".

In C and C++ applications, Pre-Processor is required to recognize #include<> statements inorder to load header files to the memory.





---

In java , the complete predefined library is provided in the form of classes and interfaces in packages

EX:

java.io  
java.util  
java.sql

If we want to use predefined library in java applications then we have to include packages in java application, for this we have to use "import" statements

EX:

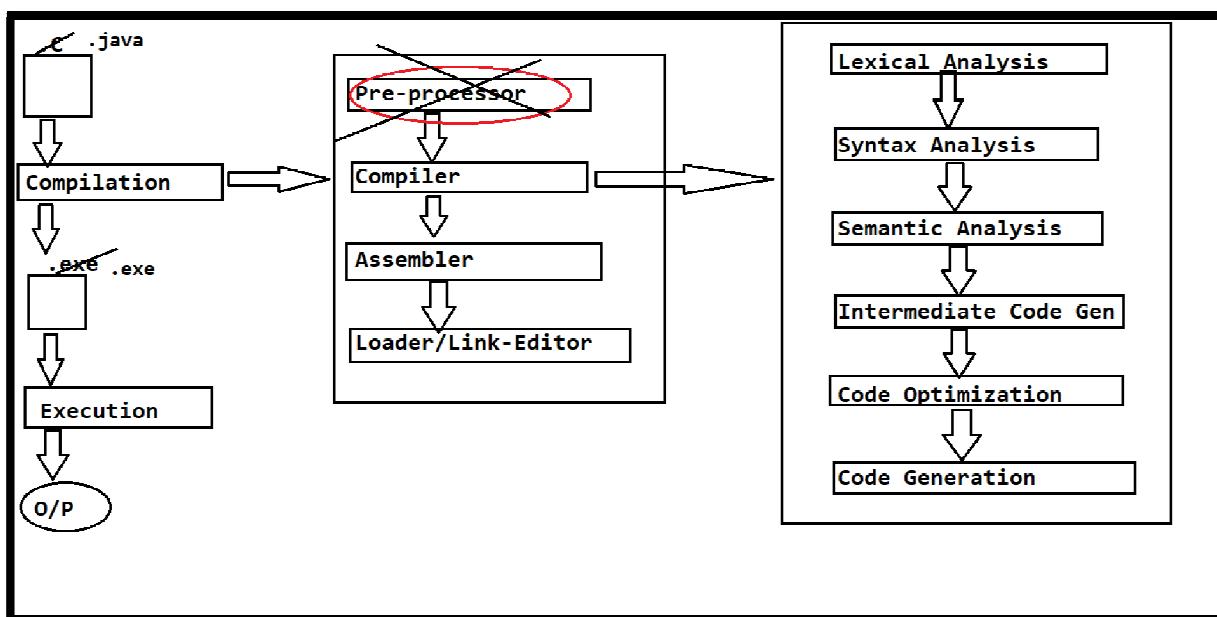
```
import java.io.*;  
import java.util.*;  
import java.sql.*;
```

If we compile java program then compiler will perform the following actions.

1. Compiler will recognize all the import statements.
2. Compiler will take the specified package names from import statements.
3. Compiler will check whether the specified packages are existed or not in java software.
4. If the specified packages are not existed in java predefined library then compiler will rise an error "package xxx does not exist".
5. If the specified packages are existed in java predefined library then Compiler will not rise any error and compiler will not load any package content to the memory.

While executing java program, when JVM[Java Virtual Machine] encounter any class or interface from the specified package then only JVM will load the required classes and interfaces to the memory at runtime, loading predefined library at runtime is called as "Dynamic Loading".

Pre-Processor is not required in JAVA , because, java does not include header files and #include<> statements, alternatively, JAVA has classes and interfaces in the form of packages and import statements.



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

1. #include<> statement is available upto C and C++.  
import statement is available upto JAVA.
2. #include<> statements are used to include the predefined library which is available in the form of header files.  
import statements are used to include the predefined library which are available in the form of packages.
3. #include<> statement is providing static loading.  
import statement is providing dynamic loading.
4. #include<> statements are recognized by Pre-Processor.  
import statements are recognized by both Compiler and JVM.
5. By using Single #include<> statement we are able to include only one header file.

**EX:**

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

By using single import statement we are able to include more than one class or more than one interface of the same package.

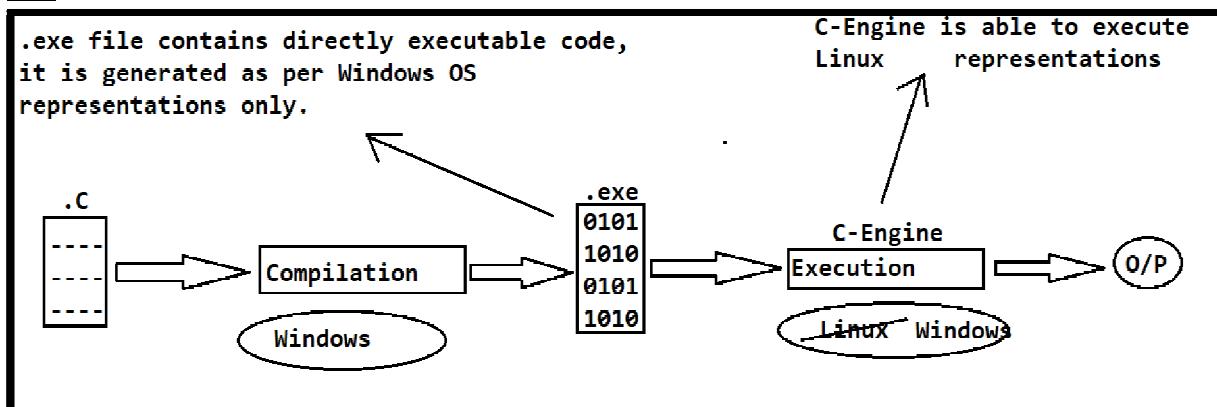
**EX: import java.io.\*;**



## 3) C and C++ are platform dependent programming languages, but, JAVA is platform Independent programming language.

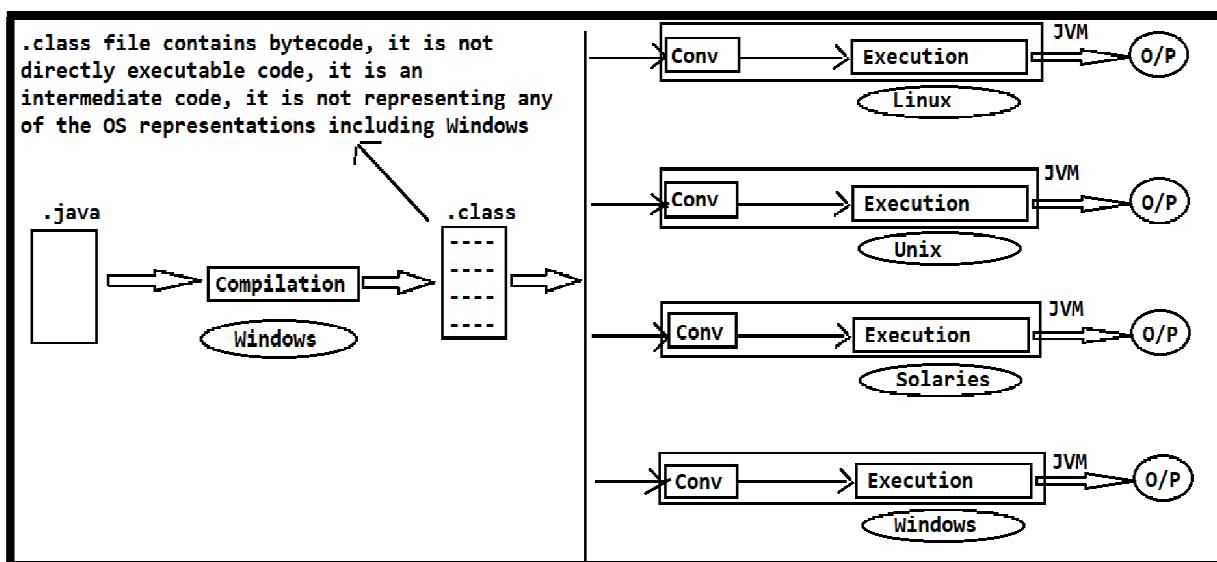
If any PL allows its applications to perform compilation and execution on the same Operating System then that PL is called as Platform Dependent PL.

EX: C and C++



If any PL allows its applications to perform Compilation is on one OS and execution is on another OS then that PL is called as Platform independent PL.

EX: JAVA





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

1. .exe file is available upto C and C++ only.  
.class file is available upto Java.
2. .exe file contains directly executable code.  
.class file contains bytecode, it is not executable code directly, it is an intermediate code.
3. .exe file is platform dependent file.  
.class file is platform independent file.
4. .exe file is less secured file.  
.class file is more secured file.

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

In general, in Progaming languages, Variables are able to store data.

EX: int eno = 111;

In C and C++, to manipulate data through memory locations , C and C++ have provided a type of varable called as "Pointer" variable.

Pointer is a variable in C and C++, it able to store address locations of the data structers, where Data Structer may be a variable, an array, a struct, or may be another pointer variable.

## Q)What are the differences between pointer variables and referece variables?

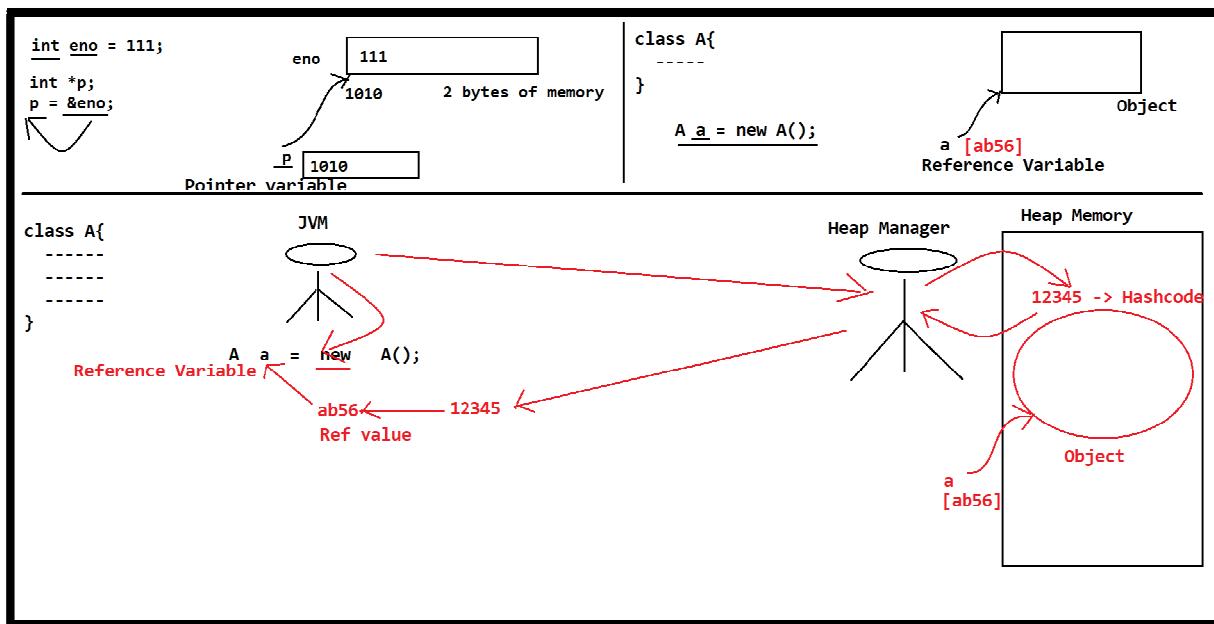
1. Pointer variables are available upto C and C++.  
Reference variables are available upto JAVA mainly.
2. Pointer variables are able to refer a block of memory by storing its address locations.

Reference variables are able to refer a block of memory (Object) by storing object reference values, where Object reference value is hexa decimal form of hashcode, where hashcode is an unique identity provided by Heap manager.

3. Pointer variables are recognized and initialized at compilation time.



## 4. Reference variables are recognized and initialized at runtime.



## 5) Multiple inheritance is not possible in Java:

If any PL allows to represent data in the form of Objects as per Object Oriented principles[Features] then that PL is called as Object Oriented PL.

In general, there are 7 Object Oriented principles or Features.

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

From the above 7 Object Oriented Features, the following features are most powerfull features.

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism

**Inheritance:** inheritance is relation[Parent-Child] between classes,it will bring variables and methods from one class[Super class / Base Class / Parent Class] to another class[Sub class / Derived Class / Child Class] inorder to reuse.



The main advantage of Inheritance is "Code Reusability".

Note: In Java, Inheritance is represented in the form of "extends" keyword.

**EX:**

```
class Employee{ // Super class
    int eno;
    String ename;
    float esal;
    String eaddr;
    ---
    ---
}

class Manager extends Employee{// Sub Class
    ---Reuse Employee class members here-----
    ---
    ---
}

class Accountent extends Employee{// Sub class
    ---Reuse Employee class members here-----
    ---
    ---
}
```

Initially, there are two types of Inheritances.

1. Single Inheritance
2. Multiple Inheritance

On the basis of the above two types of inheritances, three more Inheritances are defined.

3. Multi Level inheritance.
4. Hierarchical Inheritance.
5. Hybrid Inheritance.

**1. Single Inheritance:** It is a relation between classes, where it will bring 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, where it will bring variables and methods from more than one super class to one or more no of sub classes.

Java does not support Multiple Inheritance.



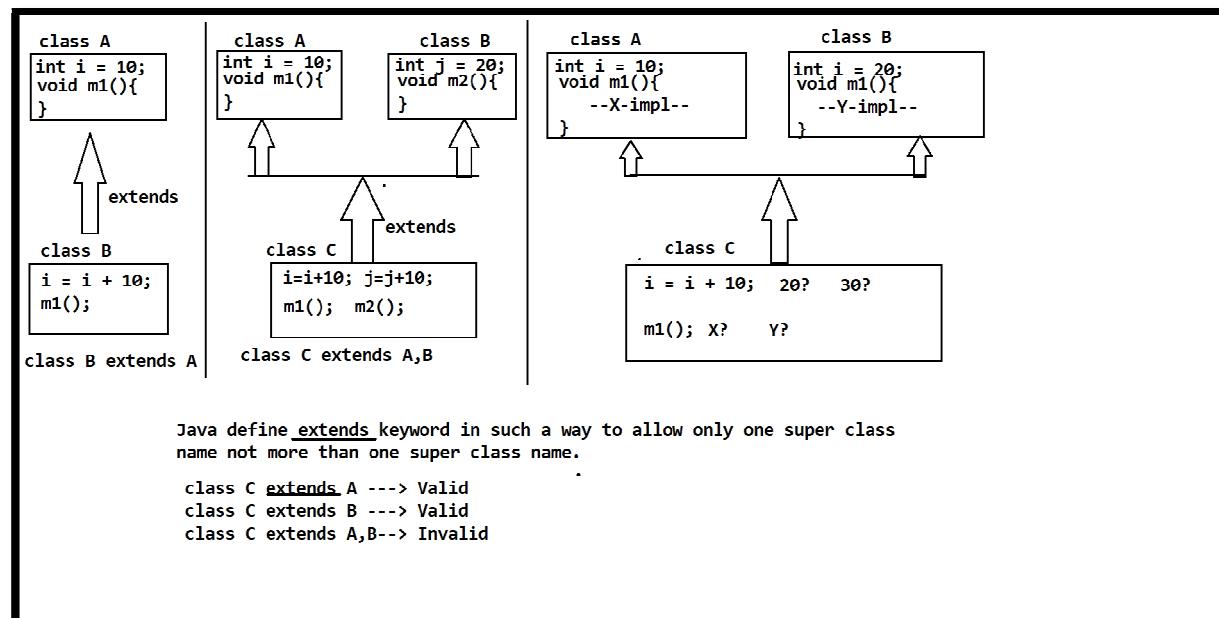
If we declare same variable with different values and same method with different implementation in both the super classes and if we access that common variable and method at sub class then which super class variable and method will be accessed is big confusion for the compiler and JVM, JAVA is simple pl, it will not allow any confusion oriented features, so JAVA does not allow Multiple Inheritances.

Note: Java define "extends" keyword in such a way to allow only one super class name , not to allow more than one super class name.

class A extends B{ } ---> Valid

class A extends C{ } ---> Valid.

class A extends B, C { } ---> invalid.



## 6)Destructors are required in C++, but, Destructors are not required in JAVA:

If any PL allows to represent data in the form of Objects as per Object oriented Features then that PL is called as "Object Oriented Programming Language".

In Object oriented Programming Languages, it is minimum to represent data in the form of objects , to represent data in the form of objects , first, we have to create objects and at end of the program we have to destroy that objects.

In Object Oriented PLs, two operations we have to perform frequently.

1. Creating Objects
2. Destroying Objects



---

To create Objects in Object Oriented Programming Languages, Object oriented Programming Languages have given a feature called as "Constructor".

To Destroy objects in Object Oriented Programming Languages, Object Oriented Programming Languages have a given a separate feature called as "Destructor".

In Java, to destroy objects automatically , JAVA has provided an internal component called as "Garbage Collector", it will destroy unused objects in Java applications.

In Java, Garbage Collector is responsible for Destroying objects, Developer is not responsible to destroy objects, so Developers are not required to use Destructors", so that, Destructors are not required in JAVA.

In C++, Developers are responsible to destroy Objects, because, in C++ Garbage Collector kind of component is not existed, Therefore Developers must use "Destructors" to destroy objects.

## 7) Operator Overloading is not supported in Java:

When a PL represents data in the form of Objects then that PL is called as an Object Oriented Programming Language.

In Object Oriented Programming Languages , while representing data in the form of Objects we have to follow a set of conventions called as "Object oriented Features".

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

From the above list of Object oriented Features the following features are the most powerful features.

1. Encapsulation
2. Abstraction
3. Inheritance
4. Polymorphism



## Polymorphism:

--> "Polymorphism" is a Greek word, where Poly means Many and Morphism means Structures / Forms.

--> If one thing is existed in more than one form then it is called as "Polymorphism".

--> The main advantage of Polymorphism is "Flexibility".

--> There are two types of Polymorphisms.

1. Static Polymorphism
2. Dynamic Polymorphism

### 1. Static Polymorphism:

--> If the Polymorphism is existed at compilation time then it is called as Static Polymorphism.

EX: Overloading

### 2. Dynamic Polymorphism:

--> If the Polymorphism is existed at runtime then that Polymorphism is called as Dynamic Polymorphism.

EX: Overriding

## Overloading:

--> There are two types of overloadings.

1. Method Overloading
2. Operator Overloading

### 1. Method Overloading:

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

EX:

---

```
class A{  
    void add(int i, int j){  
        ---implementation integer addition----  
    }  
    void add(float f1, float f2){  
        ---implementation for Float Addition----  
    }  
    void add(String str1, String str2){  
        ---Concatination on two String values----  
    }  
}
```



## 2. Operator Overloading:

--> If we declare any operator with more than one functionality then it is called as Operator Overloading.

EX:

```
int a = 10;  
int b = 20;  
int c = a + b; // + is for Arithmetic Addition.  
System.out.println(c); //30
```

```
String str1 = "abc";  
String str2 = "def";  
String str3 = str1 + str2; // + is for String concatenation  
System.out.println(str3); //abcdef
```

In Java Operator overloading is not possible , because,

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

**Note:** As per JAVA internal requirement, JAVA has defined some of the predefined operators as overloaded Operators with fixed functionalities, but, JAVA has not given any env to define operator overloading explicitly at developer level.

EX: +, \*, %,..

## 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 any PL, if we pass primitive data[int, long, float, double, boolean, char,...] as parameter to methods or functions then the parameter passing mechanism is "Call By Value".

In any PL, if we pass address locations as parameters to the methods then the parameter passing mechanism is "Call By Reference" Parameter Passing Mechanism.

In C and C++, if we pass pointer variables as parameters to the methods or functions then the parameter passing mechanism is "Call By Reference", because, Pointer variables are able to store address locations.



In Java, if we pass reference variable as parameter to the methods or functions then the parameter passing mechanism is "Call By Value" only, because, in JAVA, reference variables are not storing address locations, reference variables are able to store Object reference value, where Object reference value is hexa decimal form of Hashcode, where Hashcode is an integer value provided by Heap manager as an unique identity for each and every object.

**9.In C and C++ , integers will take only 2 bytes of memory and characters will take 1 byte of memory, but, in JAVA integers will take 4 bytes of memory and characters will take 2 bytes of memory:**

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

In Java, memory allocation for the primitive data types is fixed irrespective of the Operating System which we used.

1. byte -----> 1 byte
2. short -----> 2 bytes
3. int -----> 4 bytes
4. long -----> 8 bytes
5. float -----> 4 bytes
6. double -----> 8 bytes
7. char -----> 2 bytes
8. boolean -----> 1 bit

**Q) In C and C++, to store character value one byte of memory is sufficient, then , what is the requirement for java to assign 2 bytes of memory for character data?**

In C and C++, all characters are represented in the form of ASCII values, where to store any ASCII value 1 byte of memory is sufficient.

In Java, all characters are represented in the form of UNICODE values, where to store UNICODE values one byte of memory is not sufficient, we must provide 2 bytes of memory for characters.



## Q) What is UNICODE and what is the adv of UNICODE representation in Java?

Ans:

----

UNICODE is one of the character representation in Programming Languages, It able to represent all the alphabet from all the natural languages like English, Hindi, Italian, Chinese,.....and it able to provide very good Internationalization support in Java applications.

## Q) What is Internationalization in Java?

Ans:

-----

Internationalization is a service in Java, it enables Java applications to take input data in a particular local language and to provide output data in the same local language.

--> I have a software product, it has customers all over India, All Indians are able to understand English, so I provided all the product Services in the form of English. One fine day, I found Customers from Japan, Germany, Italy,..... and these customers are not good in English and they are not understanding my product services .

### Solutions

1. Can I prepare a separate Product for each and every customer language ---> Not Suggestible.
2. Single product, but, it has to understand Customer Locality and it has to give services as per customer locality  
in customer understandable language.

To achieve 2 requirement we have to use "Internationalization".

Designing Java applications as per Local Conventions is called as "Internationalization".

**Note:** Java is able to provide very good Internationalization support because of UNICODE representation only.



# Java Features

To show the nature of java programming language, JAVA has provided the following features.

- 1) Simple
- 2) Object Oriented
- 3) Platform independent
- 4) Arch Nuetral
- 5) Portable
- 6) Robust
- 7) Secure
- 8) Dynamic
- 9) Distributed
- 10) Multi Threaded
- 11) Interpretive
- 12) High Performance

### **1) Simple:**

Java is simple programming language, because,

- 1) Java applications will take less memory and less execution time.
- 2) Java has removed all most 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 is able to store data in the form of Objects only.

### **3) Platform Independent:**

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

### **4) Arch Nuetral:**

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

### **5) Portable:**

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



## 6) Robust:

Java is Robust programming language, because,

- 1) Java is having very good memory management system in the form of heap memory Management System, it is a dynamic memory management system, it allocates and deallocates memory for the objects at runtime.
- 2) JAVA is having very good Exception Handling mechanisms, because, Java has provided very good predefined library to represent and handle almost all the frequently generated exceptions in java applications.

## 7) Secure:

Java is very good Secure programming language, because,

- 1) JAVA has provided an implicit component inside JVM in the form of "Security Manager" to provide implicit security.
- 2) JAVA has provided a separate middleware service in the form of JAAS [Java Authentication And Authorization Service] in order to provide web security.
- 3) Java has provided very good predefined implementations for almost all well known network security alg.

## 8) Dynamic:

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

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

## 9) Distributed:

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

- 1) Standalone Applications
- 2) Distributed Applications

### 1) Standalone Applications:

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



## 2) Distributed Applications:

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

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

## 10) Multi Threaded:

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

There are 2 thread models

- 1) Single Thread Model
- 2) Multi Thread Model

### 1) Single Thread Model:

It able to allow only one thread to execute the complete application, it follows sequential execution, it will take more execution time, it will reduce application performance.

### 2) Multi Thread Model:

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

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

## 11) Interpretive:

JAVA is both compilative programming language and Interpretive programming language.

- 1) To check developers mistakes in java applications and to translate java program from High level representations to low level representation we need to compile java programs
- 2) To execute java programs , we need an interpretor inside JVM.

## 12) High Performance:

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



# JAVA Naming Conventions

Java is a case sensitive programming language, where in java applications, there is a separate recognition for lower case letters and for upper case letters.

To use lower case letters and Upper case letters separately in java applications, JAVA has provided the following conventions.

All class names , abstract class names, interface names and enum names must be started with upper case letter and the subsequent symbols must also be upper case letters.

EX:

String  
StringBuffer  
InputStreamReader

All java variables must be started with lower case letters, but, the subsequent symbols must be upper case letters.

EX:

in, out, err  
pageContext, bodyContent

All java methods must start with lower case letter , but, the subsequent symbols must be upper case letters

EX:

concat(--)  
forName(--)  
getInputStream()

4.All java Constant variables must be provided in Upper Case letters.

EX:

MIN\_PRIORITY  
NORM\_PRIORITY  
MAX\_PRIORITY

5.All java package names must be provided in lower case letters

EX:

java.util  
java.lang.reflect  
javax.servlet.jsp.tagext

Note: All the above conventions are mandatory for predefined library, they are optional from User defined library, but, suggestible.



EX:

- 1)String str=new String("abc"); ----> Valid
- 2)string str=new string("abc");-----> Invalid
- 3)class Employee{ ----> Valid and Suggestible
- 
- }
  
- 4)class student{ ---> valid, but, not suggestible
- 
- }

## Java Programming Format

To prepare basic Java application we have to use the following Structer.

- 1) Comment Section
- 2) Package Section
- 3) Import Section
- 4) Classes/Interfaces Section
- 5) Main Class Section

### **1.Comment Section:**

Before starting implementation part, it is convention to provide some description about our implementation, here to provide description about our implementation we have to use Comment Section. Description includes author name, Objective, project details, module details, client details,.....

To provide the above specified description in comment section, we will use comments.

There are 3 types of comments.

- 1) Single Line Comments
- 2) Multi Line Comments
- 3) Documentation Comments.

#### **1.Single Line Comment:**

It allows the description with in a single line.

Syntax:

// --- description-----

#### **2.Multi Line Comment:**

It allows description in more than one line



## Syntax:

```
/*
---
--description-----
---*/

```

## 3.Documentation Comment.

It allows description in more than one page.

## Syntax:

```
/*
*_____
*_____
---_
---_
*_____
*/

```

**Note:** We will use documentation comments to prepare API kind of documentations, but, it is not suggestible.

## API kind of documentation:

It is a document in the form of .txt file or .doc file or .pdf file or .html it includes the complete declarative information about our programming elements like variables, methods, classes,..... which we have used in our java file.

## EX: Employee.java

```
1) public class Employee extends Person implements Serializable, Cloneable {
2)     public String eid;
3)     public String ename;
4)     public float esal;
5)     public String eaddr;
6)     public Employee (String eid, String ename, float esal, String eaddr) {
7)     }
8)     public Employee (String eid, String ename, float esal) {
9)     }
10)    public Employee (String eid, String ename) {
11)    }
12)    public void add(String eid, String ename, float esal, String eaddr) {
13)        ----
14)    }
15)    public String search(String eid) {
16)        return "success";

```



```
17)      }
18)      public void delete(String eid) {
19)          ---
20)      }
21} }
```

[Employee.txt](#)[[html](#)/[pdf](#)]

Class : Name: Employee

Super Class: Person

Interfaces: Serializable, Cloneable

Variables: 1) Name: eid

    DataType: String

    Access Mod: public

2) Name: ename

    Data Type: String

    Access Mod: public

---

---

Methods: 1) Name: add

    Return type: void

    access mod: public

    parameters: eid, ename, esal, eaddr

-----

Constructors: 1) Employee

----

To simplify API documentation for JAVA applications, JAVA has provided an implicit command , that is, "javadoc".

EX: D:\javaapps\ Employee.java

```
1) public class Employee implements java.io.Serializable, Cloneable {
2)     public String eid;
3)     public String ename;
4)     public float esal;
5)     public String eaddr;
6)     public Employee(String eid, String ename, float esal, String eaddr) {
7)     }
8)     public Employee(String eid, String ename, float esal) {
9)     }
10)    public Employee(String eid, String ename) {
11)    }
12)        public void add(String eid, String ename, float esal, String eaddr) {
13)        }
14)        public String search(String eid) {
```



```
15)           return "success";
16)       }
17)   public void delete(String eid) {
18)       }
19) }
```

**On Command Prompt:**

```
D:\javaapps>javadoc Employee.java
--- Generating xxx.html files---
```

To provide description[Metadata] in java programs, JDK5.0 version has provided a new feature that is "Annotations".

**Q)In java applications, to provide description we have already comments then what is the requirement to use Annotations?**

If we provide description along with comments in java program then "Lexical Analysis" phase will remove comments and their description which we provided in java program as part of Compilation.

As per the requirement,if we want to make available our description upto .java file, upto .class file and upto RUNTIME of our applications there we have to use "Annotations".

**Note:** If we provide metadata with comments then we are unable to access that metadata programmatically, but, if we provide metadata with Annotations then we are able to access that metadata through java program.

**Q)In java applications, to provide metadata at RUNTIME we are able to use XML documents then what is the requirement to use "Annotations".**

If we use XMI documents to provide description then we are able to get the following problems.

- 1) We have to learn XML tech.
- 2) Every time we have to check whether XML documents are located properly or not.
- 3) Every time, we have to check whether XML documents are formatted properly or not.
- 4) Every time we have to check whether we are using right parsing mechanisms or not to read data from XML documents.

To overcome all the above problems we need a java alternative , that is, Annotations.

**Note:** IN JAVA/J2EE applicatios, we can utilize Annotations as an alternative to XML documents.



## XML Based Tech

- 1) 1.Upto JDK1.4
  - 2) 2.Upto JDBC3.0
  - 3) 3.Servlets2.5
  - 4) 4.Struts1.x
  - 5) 5.JSF1.x
  - 6) 6.EJBs2.x
  - 7) 7.Spring2.x
- 
- 1) JDK5.0 and above
  - 2) JDBC4.0
  - 3) Servlets3.0
  - 4) Struts2.x
  - 5) JSF2.x
  - 6) EJBs3.x
  - 7) Spring3.x
- 

## Annotation Based Tech [XML documents Optional]

## **2) Package Section:**

- 1.Package is the collection of related classes and interfaces as a single unit.
- 2.Package is a folder contains .class files representing related classes and interfaces.

Packages are able to provide some advantages in java applications

- 1) Modularity
- 2) Abstraction
- 3) Security
- 4) Reusability
- 5) Sharability

There are two types of packages in java

- 1) Predefined Packages
- 2) User defined Packages

### **1.Predefined Packages:**

These packages are provided by Java programming language along with java software.

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

### **2.User defined Packages:**

These packages are defined by the developers as per their application requirements.

#### Syntax:

`package package_Name;`  
where package name may be  
1.directly a single name  
2.sub package names with . operator



**EX:**

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

If we want to use package declaration statement in java files then we have to use the following two condition.

- 1.Package declaration statement must be the first statement in java file after the comment section.
- 2.Package name must be unique, it must not be sharable and it must not be duplicated.

## Q)Is it possible to declare more than one package statement with in a single Java file?

No, it is not possible to declare more than one package declaration statement with in a single java file, because, package declaration statement must be first statement, in java files only one package declaration statement must be provided as first statement.

abc.java

```
package p1;---> Valid  
package p2;---> Invalid  
package p3;---> INValid  
--  
--
```

To provide package names, JAVA has given a convention like to include our company domain name in reverse in package names.

**EX:** www.durgasoft.com

durgasoft.com  
com.durgasoft  
package com.durgasoft.icici.transactions.deposit;  
com.durgasoft---> company domain name in reverse.  
icici -----> project name/ client name  
transactions---> module name  
deposit-----> sub module

## 3.Import Section:

The main intention of "import" statement is to make available classes and interfaces of a particular package into the present JAVA file inorder to use in present java file.

### Syntax 1:

```
import package_Name.*;  
-- It able to import all the classes and interfaces of the specified package into the present java file.
```



EX: import java.io.\*;

### Syntax 2:

import package\_Name.Member\_Name;

--> It able to import only the specified member from the specified package into the present java file.

EX: import java.io.BufferedReader;

### Q) Is it possible to write more than one "import" statement with in a single Java file?

Yes, In a single java file, we are able to provide atmost one package declaration statement, but, we are able to provide any no of import statements.

abc.java

```
package p1;  
//package p2;----> Error  
//package p3;----> Error  
import java.io.*;  
import java.util.*;--> No Error  
import java.sql.*;--> No Error  
---  
---  
---
```

### Q) Is it possible to use classes and interfaces of a particular package with out importing that package?

Yes, it is possible to use classes and interfaces of a particular package in the present java file with out importing the respective package, but, just by using fully qualified names of the classes.

**Note:** Specifying class names or interface names along with their respective package names is called as "Fully Qualified Names".

EX: java.io.BufferedReader

java.util.ArrayList

java.sql.Connection

A java program with import statement:

import java.io.\*;

---

---

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));



---

---

A Java program with out import statement:

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

## 4.Classes/Interfaces Section:

The main intention of classes and interfaces is to represent all real world entities in the form of coding part.

EX: Account, Employee, Product, Customer, Student,.....

**Note:** No restrictions for no of classes in a java file or in a java application, depending on the application requirement, we are able to write any no of classes and interfaces in java applications.

## 5.Main Class Section:

Main Class is a java class ,it includes main() method.

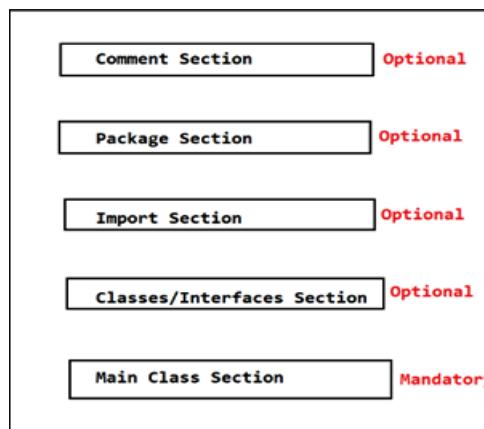
The main intention of main() method is,

- 1.To manage application logic which we want to execute by JVM directly we have to use main() method.
- 2.To define starting point and ending point to the application execution we have to use main() method.

### Syntax:

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

**Note:**main() method is a conventional method with fixed prototype and with user defined implementation part.





# Steps To Prepare First Java Application



# Steps to prepare First Java Application:

- 1) Install Java Software
- 2) Select Java Editor
- 3) Write Java Program
- 4) Save Java File
- 5) Compile Java File
- 6) Execute Java Application

## 1) Install Java Software:

- 1) Download jdk-8-windows-i586.exe file from internet.
- 2) Double click on jdk-8-windows-i586.exe file
- 3) Click on "Yes" button.
- 4) Click On "Next" button.
- 5) Change JDK installation location from "C:\Program Files(x86)\Java\jdk1.8.0" to "C:\Java\jdk1.8.0" by clicking on "Change" button and "OK" button.
- 6) Click on "Next" button.
- 7) Change JRE installation location from "C:\Program Files(x86)\Java\jre8" to "C:\Java\jre8" by clicking on "Change" button and "OK" button.
- h) Click on "Next" button.
  - a. Click on "Close" button.

After installation of JAVA software, we have to set "path" environment variable to the location where all JDK commands are existed that is "C:\Java\JDK1.8.0\bin" inorder to make available all JAVA commands to Operating System.

On command Prompt:

```
set path=C:\Java\JDK1.8.0\bin;
```

If we provide "path" set up like above on the command prompt then this set up is available upto the present command prompt only, it is not available to all the command prompts.

If we want to set "path" environment variable permanently then we have to use the following steps.

- 1) Right Click on "Computers" or "This PC" on desktop .
- 2) Select "Properties".
- 3) Select "Advanced System Settings" hyper link.
- 4) Click on "Advanced" Tab[Bydefault Selected].
- 5) Click on "Environment Variables.." button.
- 6) Goto User Variables part and click on "New" button.
- 7) Provide the following details.

variable name: path

variable value: C:\Java\jdk1.8.0\bin";



- 8) Click on "Ok" button.
- 9) Click on "OK" button.
- 10) Click on "OK" button.

**Q) If we set all three versions [JAVA6, JAVA7, JAVA8] to the "path" environment variable then which version will come to the command prompt?**

If set all JAVA6, JAVA7, JAVA8 versions to "path" environment variable then Command prompt will take the java version which we provided as first one to the path environment variable in the order.

**EX:**

path=C:\Java\jdk1.6.0\bin;C:\Java\jdk1.7.0\bin;C:\Java\jdk1.8.0\bin;

On Command Prompt:

```
D:\javaapps>java -version --> Enter  
Java - Version: JDK1.6.0
```

If we want to switch java from one version to another version in simplified manner as per the requirement then we have to use batch files.

- a) Take a text file.
- b) provide the required "path" command.
- c) Save file with "file\_Name.bat".
- d) Open Command prompt and goto the location where bat files are saved.
- e) write bat file name and click on enter button.

```
D:\java9\ java6.bat  
set path=C:\Java\jdk1.6.0\bin;
```

```
D:\java9\ java7.bat  
set path=C:\Java\jdk1.7.0\bin;
```

```
D:\java9\ java8.bat  
set path=C:\Java\jdk1.8.0\bin;
```

On Command Prompt:

```
D:\java9>java6.bat --> Enter button  
--- we will get java6 setup----
```

```
D:\java9>java7.bat --> Enter button  
--- we will get java7 setup----
```



D:\java9>java8.bat ---> Enter button  
--- we will get java8 setup----

## 2) Select Java Editor:

Editor is a software, it will provide very good env to write java programs and to save java programs in our system.

EX: Notepad, Notepadplus, Editplus,....

Note: In real time application development, it is not suggestible to use Editors, it is always suggestible to use IDEs[Integrated Development Environment].

EX: Eclipse, MyEclipse, Netbeans,.....

## 3) Write Java Program:

To write java program we have to use some predefined library provided by JAVA API[Application Programming Interface].

EX

D:\javaapps\ Test.java

```
1) class Test {  
2)     public static void main(String[] args) {  
3)         System.out.println("First Java Application");  
4)     }  
5) }
```

## 4) Save Java File:

To save java file in our system, we have to follow the following two conditions.

1.If the present java file contains any public element[class, abstract class, interface, enum] then we must save java file with public element name only.If we violate this condition then compiler will rise an error.

2.If no public element is identified in our java file then it is possible to save java file with any name like abc.java or xyz.java, but, it is suggestible to save java file with main() method class name.



## Q) Is it possible to provide more than one public class with in a single java file?

No, it is not possible to provide more than one public class with in a single java file, because, 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.

### EX1: File Name: abc.java

```
1) class FirstApp {  
2)     public static void main(String[] args) {  
3)         System.out.println("First Java Application");  
4)     }  
5) }
```

Status: No Compilation Error, but not suggestible.

### EX2: File Name: FirstApp.java

```
1) class FirstApp {  
2)     public static void main(String[] args) {  
3)         System.out.println("First Java Application");  
4)     }  
5) }
```

Status: No Compilation Error, it is suggestible.

### EX3: File Name: FirstApp.java

```
1) public class A {  
2) }  
3) class FirstApp {  
4)     public static void main(String[] args) {  
5)         System.out.println("First Java Application");  
6)     }  
7) }
```

Status: Compilation Error.



## EX4: File Name: A.java

```
1) public class A {  
2) }  
3) class FirstApp {  
4)     public static void main(String[] args) {  
5)         System.out.println("First Java Application");  
6)     }  
7) }
```

Status: No Compilation Error.

## EX5: File Name: A.java

```
1) public class A {  
2) }  
3) public class B {  
4) }  
5) class FirstApp {  
6)     public static void main(String[] args) {  
7)         System.out.println("First Java Application");  
8)     }  
9) }
```

Status: Compilation Error

## 5) Compile Java File:

1. The main purpose of compiling JAVA file is to convert JAVA programme from High Level Representation to LowLevelRepresentation.

2. To Check compilation errors.

To compile JAVA file, we have to use the following command on command prompt from the location where JAVA File is Saved

javac File\_Name.java

EX: d:\java9>javac FirstApp.java

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

1. Operating System will take "javac" command from command prompt and search for it at its predefined commands lists and at the locations referred by "path" environment variable.



2.If the required "javac" program is not available at the above two locations then operating System will give a message on command prompt.

"javac can not be recognized as an internal command,external command or operable program and a batch file"

**NOTE:**To make available all JDK tools like javac,java...to the operating System we have to set "path" environment variable to "c:\java\jdk1.7.0\bin" location.

```
D:\java9>set path=C:\Java\jdk1.7.0\bin;
```

3.If the required "javac" program is identified at "c:\java\jdk1.7.0\bin" location through "path" environment variable then operating System will execute "javac" program and activate "Java Compiler" software.

4.When Java Complier Software is activated then Java Complier will perform the following tasks.

a)Compiler will take java File name from command prompt.

b)Compiler will search java file at current location.

c)If the required java file is not available at current location then compiler will provide the following message on command prompt.

```
javac : file not found : FirstApp.java
```

d)If the required java file is available at current location then compiler will start compilation from starting point to ending point of the java file.

e)In compilation process,if any syntax violations are identified then compiler will generate error messages on command prompt.

f)In compilation process,if no syntax errors are identified then compiler will generate .class files at the same current location.

**NOTE:**Generating no of .class files is completely depending on the no of classes,no of abstract classes,no of interfaces,no of enums and no of inner classes which we used in the present java File

If we want to compile java file from current location and if we want to send the generated .class Files to some other target location then we have to use the following command on command prompt

```
javac -d target_location File_Name.java
```

**EX:**

```
d:\java9>javac -d c:\abc FirstApp.java
```



If we use package declaration statement in java file and if we want to store the generated .class files by creating directory structure w.r.t the package name in compilation process then we have to use  
"-d" option along with "javac" command.

EX: File Name :D:\java9\FirstApp.java

```
1) package com.durgasoft.core;
2) enum E {
3) }
4) interface I {
5) }
6) abstract class A {
7) }
8) class B {
9)   class C {
10) }
11) }
12) class FirstApp {
13)   public static void main(String[] args) {
14)   }
15) }
```

D:\java9>javac -d c:\abc FirstApp.java

Compiler will compile FirstApp.java File from "D:\java9" location and it will generate E.class, I.class, A.class, B.class, B\$C.class, FirstApp.class files at the specified target location "C:\abc" by creating directory structure w.r.t the package name "com.durgasoft.core".

If we want to compile all the java files which are available at current location then we have to use the following command.

D:\java9>javac \*.java

If we want to compile all the java files which are prefixed with a word then we have to use the following command.

D:\java9>javac Employee\*.java

If we want to compile all the java files which are postfixed with a common word then we have to use the following command.

D:\java9>javac \*Address.java

If we want to compile all the java Files which contain particular word then we have to use the following command



```
D:\java9>javac *Account*.java  
D:\java9>javac -d C:\abc *Account*.java
```

## 6) Execute JAVA Application:

If we want to execute java program then we have to use the following command prompt.

```
java Main_Class_Name
```

EX:

```
d:\java9>java FirstApp
```

If we use the above command on command prompt then operating System will execute "java" operable program at "C:\java\jdk1.7.0\bin",with this,JVM software will be activated and JVM will perform the following actions.

JVM will take Main\_Class name from command prompt.

JVM will search for Main\_Class at current location,at Java predefined library and at the locations referred by "classpath" environment variable.

If the required Main\_Class .class file is not available at all the above locations then JVM will provide the following.

JAVA6: java.lang.NoClassDefFoundError:FirstApp

JAVA7: Error:Could not find or load main class FirstApp

**NOTE:**If main class .class file is available at some other location then to make available main class.class file to JVM we have to set "classpath" environment variable.

```
D:\java7>set classpath=E:\XYZ;
```

If the required main class .class file is available at either of the above locations then JVM will load main class bytecode to the memory by using "Class Loaders".

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

If main() method is not available at main class byteCode then JVM will provide the following.

JAVA6: java.lang.NoSuchMethodError:main

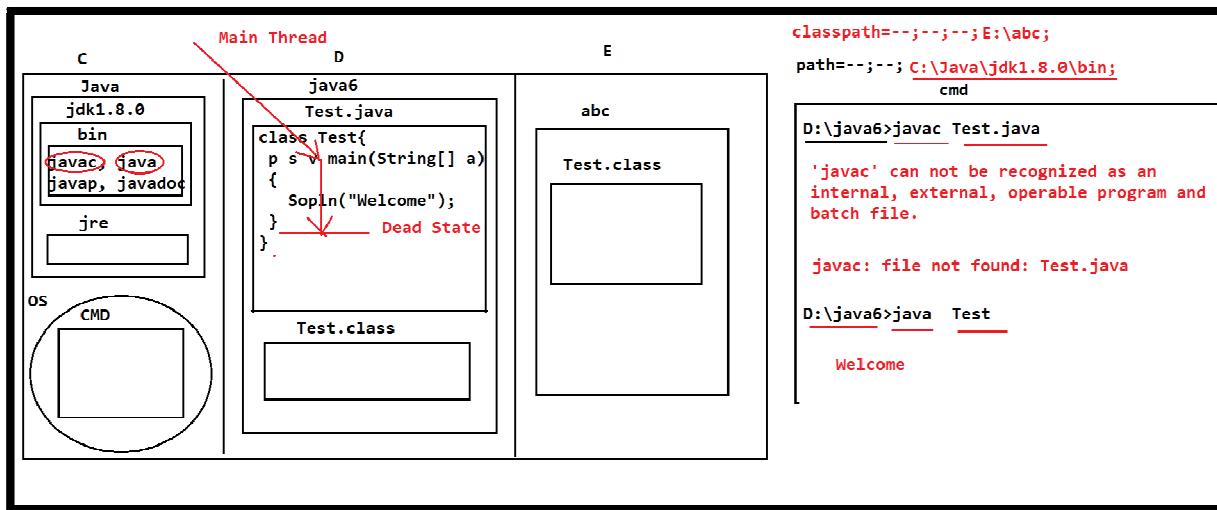
JAVA7: Error:Main method not found in class FirstApp,please define main method as:  
public static void main(String args[])



If main() method is available at main class bytecode then JVM will access main() method by creating a thread called as "Main thread".

JVM will access main() method to start application execution by using main thread, when main thread reached to the ending point of main() method then main thread will be in destroyed/dead state.

When main Thread is in dead state then JVM will stop all of its internal processes and JVM will go to ShutDown mode.





# Language Fundamentals



# Language Fundamentals

To prepare java applications , we need some fundamentals provided by Java programming language.

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

### 1) Tokens:

Smallest logical unit in java programming is called as "Lexeme".

The Collection of Lexemes come under a particular group is called as "Token"

```
int a=b+c*d;
```

Lexemes: int, a, =, b, +, c, \*, d, ;-----> 9

Tokens:

- 1) Data Types: int
- 2) Identifiers: a, b, c, d
- 3) Operators: =, +, "\*
- 4) Special Symbol: ;

Types of tokens: 4

To prepare java applications, java has provided the following list of tokens.

- 1) Identifiers
- 2) Literals
- 3) Keywords/ Reserved Words
- 4) Operators



## Identifiers:

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

EX:

```
int a=10;  
int ----> Data Types  
a -----> variable[Identifier]  
= -----> Operator  
10 -----> constant  
; -----> Terminator
```

To provide identifiers in java programming, we have to use the following rules and regulations.

Identifiers should not be started with any number, identifiers may be started with an alphabet, '\_' symbol, '\$' symbol, but, the subsequent symbols may be a number, an alphabet, '\_' symbol, '\$' symbol.

```
int eno=111;-----> Valid  
int 9eno=999;-----> Invalid  
String _eaddr="Hyd";---> Valid  
float $esal=50000.0f;----> valid  
String emp9No="E-9999";----> Valid  
String emp_Name="Durga";----> Valid  
float emp$Sal=50000.0f;-----> Valid
```

Identifiers are not allowing all operators and all special symbols except '\_' and '\$' symbols.

```
int empNo=111; -----> valid  
int emp+No=111;-----> Invalid  
String emp*Name="Durga";-----> Invalid  
String #eaddr="Hyd";--->Invalid  
String emp@Hyd="Durga";-----> Invalid  
float emp.Sal=50000.0f;-----> Invalid  
String emp-Addr="Hyd";-----> Invalid  
String emp_Adr="Hyd";-----> Valid
```

Identifiers are not allowing spaces in the middle.

```
concat(--)----> Valid  
forName(--)----> Valid  
for Name()----> Invalid  
getInputStream()--> valid  
get Input Stream()----> Invalid
```



Identifiers should not be duplicated with in the same scope, identifiers may be duplicated in two different scopes.

```
1) class A {  
2)     int i=10;----> Class level  
3)     short i=20;----> Error  
4)     double f=33.33---> No Error  
5)     void m1() {  
6)         float f=22.22f; ----> local variable  
7)         double f=33.33;---> Error  
8)         long i=30;---> No Error  
9)     }  
10) }
```

In java applications, we can use all predefined class names and interface names as identifiers.

**EX1:**

```
int Exception=10;  
System.out.println(Exception);  
Status: No Compilation Error
```

**Output:** 10

**EX2:**

```
String String="String";  
System.out.println(String);  
Status: No Compilation Error
```

**Output:** String

**EX3:**

```
int System=10;  
System.out.println(System);  
Status: COmpilation Error
```

**Reason:** Once if we declare "System"[Class Name] as an integer variable then we must use that "System" name as integer variable only in the remaining program, in the remaining program if we use "System" as class name then compiler will rise an error.  
In the above context, if we want to use "System" as class name then we have to use its fully qualified .

**Note:** Specifying class names and interface names along with package names is called as Fully Qualified Name.



**EX:** `java.io.BufferedReader`  
`java.util.ArrayList`

**EX:**

```
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);
Status: No Compilation Error
```

**Output:** 10

```
20
30
```

Along with the above rules and regulations, JAVA has provided a set of suggestions to use identifiers in java programs

In java applications, it is suggestible to provide identifiers with a particular meaning.

**EX:**

```
String xxx="abc123";----> Not Suggestible
String accNo="abc123";----> Suggestible
```

In java applications, we dont have any restrictions over the length of the identifiers, we can declare identifiers with any length, but, it is suggestible to provide length of the identifiers around 10 symbols.

**EX:**

```
String permanentemployeeaddress="Hyd";----> Not Suggestible
String permEmpAddr="Hyd";----> Suggestible
```

If we have multiple words with in a single identifier then it is suggestible to seperate multiple words with special notations like '\_' symbols.

**EX:**

```
String permEmpAddr="Hyd";----> Not Suggestible
String perm_Emp_Addr="Hyd";----> Suggestible
```



## Literals

Literal is a constant assigned to the variables .

EX:

int a=10;  
int ----> data types  
a -----> variables/ identifier  
= -----> Operator  
10 -----> constant[Literal].  
; -----> Special symbol.

To prepare java programs, JAVA has provided the following set of literals.

### 1.Integer / Integral Literals:

byte, short, int, long ----> 10, 20, 30,....  
char -----> 'A','B',....

### 2.Floating Point Literals:

float ----> 10.22f, 23.345f,.....  
double----> 11.123, 456.345,....

### 3.Boolean Literals:

boolean -----> true, false

### 4.String Literals:

String ---> "abc", "def",.....

Note: JAVA7 has given a flexibility like to include '\_' symbols in the middle of the literals inorder to improve readability.

EX:

float f=12345678.2345f;  
float f=1\_23\_45\_678.2345f;

If we provide '\_' symbols in the literals then compiler will remove all '\_' symbols which we provided, compiler will reformat that number as original number and compiler will process that number as original number.



## Number Systems in Java:

In general, in any programming language, to represent numbers we have to use a particular system .

There are four types of number systems in programming languages.

1. Binary Number Systems[BASE-2]
2. Octal Number Systems[BASE-8]
3. Decimal Number Systems[BASE-10]
4. Hexa Decimal Number Systems[BASE-16]

In java , all number systems are allowed, but, the default number system in java applications is "Decimal Number Systems".

## Binary Number Systems [BASE-2]:

If we want to represent numbers in Binary number system then we have to use 0's and 1's, but, the number must be prefixed with either '0b' or '0B'.

int a=10;----> It is not binary number, it is decimal num.  
int b=0b10;---> valid  
int c=0B1010;---> valid  
int d=0b1012;---> Invalid, 2 symbol is not binary numbers alphabet.

Note: Binary Number system is not supported by all the java versions upto JAVA6, but , JAVA7 and above versions are supporting Binary Number Systems, because, it is a new feature introduced in JAVA7 version.

## Octal Number Systems [BASE-8]:

If we want to prepare numbers in Octal number System then we have to use the symbols like 0,1,2,3,4,5,6 and 7, but, the number must be prefixed with '0'[zero].

EX:

int a=10; -----> It is decimal number, it is not octal number.  
int b=012345;---> Valid  
int c=0234567;---> Invalid, number is prefixed with 0, not zero  
int d=04567;---> Valid  
int e=05678;---> Invalid, 8 is not octal number systems alphabet.

## Decimal Number Systems [BASE-10]:

If we want to represent numbers in Decimal number system then we have to use the symbols like 0,1,2,3,4,5,6,7,8 and 9 and number must not be prefixed with any symbols.

EX:

int a=10;----> Valid



int b=20;----> Valid  
int c=30;----> valid

## Hexa Decimal Number Systems [BASE-16]:

If we want to prepare numbers in Hexa decimal number system then we have to use the symbols like 0,1,2,3,4,5,6,7,8,9, a,b,c,d,e and f, but the number must be prefixed with either '0x' or '0X'.

EX:

int a=10;----> It is not hexa decimal number, it is decimal number.  
int b=0x102345;---> Valid  
int c=0X56789;---> Valid  
int d=0x89abcd;---> valid  
int e=0x9abcdefg;----> Invalid, 'g' is not in hexa decimal alphabet.

Note: If we provide numbers in all the above number systems in java applications then compiler will recognize all the numbers and their number systems on the basis of their prefix values , compiler will convert these numbers into decimal system and compilers will process that numbers as decimal numbers.

## Keywords/ Reserved Words

If any predefined word having both word recognition and internal functionality then that predefined word is called as Keyword.

If any predefined word having only word recognition with out internal functionality then that predefined word is called as Reserved word.

EX: goto const.

To prepare java applications, Java has provided the following list of keywords.

### Data types and Return types:

bytes, short, int, long, float, double, char, boolean, void.

### Access Modifiers:

public, protected, private, static, final, abstract, native, volatile, transient, synchronized, strictfp,.....

### Flow Controllers:

if, else, switch, case, default, for, while, do, break, continue, return,....



## Class/ Object Related:

class, enum, extends, interface, implements, package, import, new, this, super, ....

## Exception Handling Related Keywords:

throw, throws, try, catch, finally

# OPERATORS

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

To prepare java applications, JAVA has provided the following list of operators.

### 1) Arithmetic Operators:

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

### 2) Assignment Operators:

=, +=, -=, \*=, /=, %=,.....

### 3) Comparision Operators:

==, !=, <, >, <=, >=,.....

### 4) Boolean Logical Operators:

&, |, ^

### 5) Bitwise Logical Operators:

&, |, ^, <<, >>,...

### 6) Short-Circuit Operators:

&&, ||

### 7. Ternary Operator:

Expr1? Expr2: Expr3;

#### Ex1:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int a=10;
6)     System.out.println(a);
7)     System.out.println(a++);
```



```
8) System.out.println(++a);
9) System.out.println(a--);
10) System.out.println(--a);
11) System.out.println(a);
12) }
13} }
```

Status: No Compilation Error

OUTPUT: 10

```
10
12
12
10
10
```

EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int a=5;
6)     System.out.println(++a-++a);
7)   }
8) }
```

Status: No Compilation Error

OUTPUT: -1

EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int a=5;
6)     System.out.println((-a---a)*(++a-a--)+(-a+a--)*(++a+a++));
7)   }
8) }
```

Status: No Compilation Error

OUTPUT: 16



| 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:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         boolean b1=true;
6)         boolean b2=false;
7)
8)         System.out.println(b1&b1);//true
9)         System.out.println(b1&b2);//false
10)        System.out.println(b2&b1);//false
11)        System.out.println(b2&b2);//false
12)
13)        System.out.println(b1|b1);//true
14)        System.out.println(b1|b2);//true
15)        System.out.println(b2|b1);//true
16)        System.out.println(b2|b2);//false
17)
18)        System.out.println(b1^b1);//false
19)        System.out.println(b1^b2);//true
20)        System.out.println(b2^b1);//true
21)        System.out.println(b2^b2);//false
22)    }
23} }
```

| 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   |

T----> 1

F----> 0



EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int a=10;
6)         int b=2;
7)         System.out.println(a&b);
8)         System.out.println(a|b);
9)         System.out.println(a^b);
10)        System.out.println(a<<b);
11)        System.out.println(a>>b);
12)    }
13} }
```

int a=10;----> 1010

int b=2;----> 0010

a&b--->10&2--> 0010----> 2

a|b--->10|2--> 1010----> 10

a^b--->10^2--> 1000----> 8

a<<b ---> 10<<2 ----> 00001010

                  00101000---> 40

--> Remove 2 symbols at left side and append 2 0's at right side.

a>>b ---> 10>>2 ----> 00001010

                  00000010

--> Remove 2 symbols at right side and append 2 0's at left side.

**Note:** Removable Symbols may be 0's and 1's but appendable symbols must be 0's.

## Short-Circuit Operators:

The main intention of Short-Circuit operators is to improve java applications performance.

EX: 1.&& 2.||

|   Vs   ||

In the case of Logical-OR operator, if the first operand value is true then it is not required to check second operand value, directly, we can predict the result of overall expression is true.



In the case of '| ' operator, even first operand value is true , still, JVM evaluates second opointer value then only JVM will get the result of overall expression is true, here evaluating second operand value is unnecessary, it will increase execution time and it will reduce application performance.

In the case of '| |' operator, if the first operand value is true then JVM will get the overall expression result is true with out evaluating second operand value, here JVM is not evaluating second operand expression unneccessarily, it will reduce execution time and it will improve application performance.

**Note:** If the first operand value is false then it is mandatory for JVM to evaluate second operand value inorder to get overall expression result.

**EX:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int a=10;
6)         int b=10;
7)         if( (a++ == 10) | (b++ == 10) )
8)         {
9)             System.out.println(a+ " "+b); //OUTPUT: 11 11
10)        }
11)        int c=10;
12)        int d=10;
13)        if( (c++ == 10) || (d++ == 10) )
14)        {
15)            System.out.println(c+ " "+d); //OUTPUT: 11 10
16)        }
17)    }
18} }
```

## & Vs &&

In the case of Logical-AND operator, if the first operand value is false then it is not required to check second operand value, directly, we can predict the result of overall expression is false.

In the case of '&' operator, even first operand value is false , still, JVM evaluates second opointer value then only JVM will get the result of overall expression is false, here evaluating second operand value is unnecessary, it will increase execution time and it will reduce application performance.

In the case of '&&' operator, if the first operand value is false then JVM will get the overall expression result is false with out evaluating second operand value, here JVM is not



evaluating second operand expression unnecessarily, it will reduce execution time and it will improve application performance.

**Note:** If the first operand value is true then it is mandatory for JVM to evaluate second operand value inorder to get overall expression result.

**EX:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int a=10;
6)         int b=10;
7)         if( (a++ != 10) & (b++ != 10) )
8)         {
9)         }
10)        System.out.println(a+ " " +b); //OUTPUT: 11 11
11)        int c=10;
12)        int d=10;
13)        if( (c++ != 10) && (d++ != 10) )
14)        {
15)        }
16)        System.out.println(c+ " " +d); //OUTPUT: 11 10
17)    }
18) }
```

## 2) Data Types:

Java is strictly a typed programming language, where in java applicatins before representing data first we have to confirm which type of data we representing. In this context, to represent type of data we have to use "data types".

**EX:** `i = 10;`----> invalid, no data type representation.

`int i=10;`--> Valid, type is represented then data is represented.

In java applications , data types are able to provide the following advatages.

1. We are able to identify memory sizes to store data.

**EX:** `int i=10;`--> int will provide 4 bytes of memory to store 10 value.

2. We are able to identify range values to the variable to assign.

**EX:** `byte b=130;`--> Invalid

`byte b=125;`--> Valid



**Reason:** 'byte' data type is providing a particular range for its variables like -128 to 127, in this range only we have to assign values to byte variables.

To prepare java applications, JAVA has provided the following data types.

## 1. Primitive Data Types / Primary Data types

### Numeric Data Types

#### Integral data types/ Integer Data types:

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

#### Non-Integral Data Types:

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

#### Non-Numeric Data types:

char -----> 2 bytes---> '' [single space]  
boolean-----> 1 bit-----> false

## 2. User defined data types / Secondary Data types

All classes, all abstract classes, all interfaces, all arrays,.....

No fixed memory allocation for User defined data types

If we want to identify range values for variables onthe basis of data types then we have to use the following formula.

$n-1$        $n-1$   
 $-2$     to    $2 - 1$

Where 'n' is no of bits.

EX: Data Type: byte , size= 1 byte = 8 bits.

$8-1$        $8-1$   
 $-2$     to    $2 - 1$

$7$        $7$   
 $-2$     to    $2 - 1$

$-128$  to  $128 - 1$

$-128$  to  $127$



**Note:** This formula is applicable upto Integral data types, not applicable for other data types.

To identify "min value" and "max value" for each and every data type, JAVA has provided the following two constant variables from all the wrapper classes.

**MIN\_VALUE** and **MAX\_VALUE**

**Note:** Classes representation of primitive data types are called as **Wrapper Classes**

## Primitive Data Types    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   |

**EX:**

```
1) class Test{  
2)   public static void main(String[] args){  
3)     System.out.println(Byte.MIN_VALUE+----->+Byte.MAX_VALUE);  
4)     System.out.println(Short.MIN_VALUE+----->+Short.MAX_VALUE);  
5)     System.out.println(Integer.MIN_VALUE+----->+Integer.MAX_VALUE);  
6)     System.out.println(Long.MIN_VALUE+----->+Long.MAX_VALUE);  
7)     System.out.println(Float.MIN_VALUE+----->+Float.MAX_VALUE);  
8)     System.out.println(Double.MIN_VALUE+----->+Double.MAX_VALUE);  
9)     System.out.println(Character.MIN_VALUE+----->+Character.MAX_VALUE);  
10)    //System.out.println(Boolean.MIN_VALUE+----->+Boolean.MAX_VALUE); -  
--> Error  
11)  }  
12} }
```



# 3) Type Casting:

The process of converting data from one data type to another data type is called as "Type Casting".

There are two types of type castings are existed in java.

1. Primitive data Types Type Casting
2. User defined Data Types Type Casting

**Note:** To perform User defined data types type casting we need either "extends" relation or "implements" relation between user defined data types.

## 1. Primitive data Types Type Casting:

The process of converting data from one primitive data type to another primitive data type is called as Primitive data types type casting.

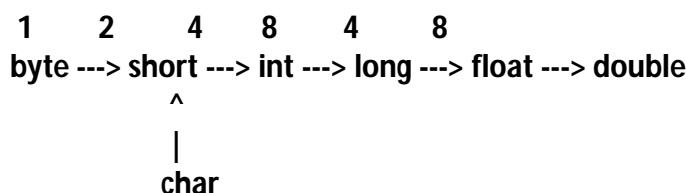
There are two types of primitive data types type castings.

1. Implicit Type Casting
2. Explicit Type Casting

### Implicit Type Casting:

The process of converting data from lower data type to higher data type is called as Implicit Type Casting.

To cover all the possibilities of implicit type casting JAVA has provided the following chart.



If we want to perform implicit type casting in java applications then we have to assign lower data type variables to higher data type variables.

**EX:**

```
byte b=10;  
int i = b;  
System.out.println(b+" "+i);
```

Status: No Compilation Error



---

OUTPUT: 10 10

If we compile the above code, when compiler encounter the above assignment statement then compiler will check whether right side variable data type is compatible with left side variable data type or not, if not, compiler will rise an error like "possible loss of precision". If right side variable data type is compatible with left side variable data type then compiler will not rise any error and compiler will not perform any type casting.

When we execute the above code, when JVM encounter the above assignment statement then JVM will perform the following two actions.

- 1.JVM will convert right side variable data type to left side variable data type implicitly[Implicit Type Casting]
- 2.JVM will copy the value from right side variable to left side variable.

**Note:** Type Checking is the responsibility of compiler and Type Casting is the responsibility of JVM.

EX2:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int i=10;
6)         byte b=i;
7)         System.out.println(i+ " "+b);
8)     }
9) }
```

Status: Compilation Error, Possible loss of precision.

EX3:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         byte b=65;
6)         char c=b;
7)         System.out.println(b+ " "+c);
8)     }
9) }
```

Status: Compilation Error



## EX4:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         char c='A';
6)         short s=c;
7)         System.out.println(c+ " "+s);
8)     }
9) }
```

Status: Compilation Error, Possible loss of precision.

Reason: byte and short internal data representations are not compatible to convert into char.

## EX5:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         char c='A';
6)         int i=c;
7)         System.out.println(c+ " "+i);
8)     }
9) }
```

Status: No Compilation Error

OUTPUT: A 65

Reason: Char internal data representation is compatible with int.

## EX6:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         byte b=128;
6)         System.out.println(b);
7)     }
8) }
```

Status: Compilation Error,possible loss of precision.



**Reason:** When we assign a value to a variable of data type, if the value is greater than the max limit of the left side variable data type then that value is treated as of the next higher data type value.

**Note:** For both byte and short next higher data type is int only.

**EX7:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         byte b1=60;
6)         byte b2=70;
7)         byte b=b1+b2;
8)         System.out.println(b);
9)     }
10) }
```

**Status:** Compilation Error, possible loss of precision.

**EX8:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         byte b1=30;
6)         byte b2=30;
7)         byte b=b1+b2;
8)         System.out.println(b);
9)     }
10) }
```

**Status:** Compilation Error, Possible loss of precision.

**Reason:** X,Y and Z are three primitive data types.

X+Y=Z

- 1.If X and Y belongs to {byte, short, int} then Z should be int.
- 2.If either X or Y or both X and Y belongs to {long, float, double} then Z should be higher(X,Y).

byte+byte=int  
byte+short=int  
short+int=int



byte+long=long  
long+float=float  
float+double=double

EX9:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         long l=10;
6)         float f=l;
7)         System.out.println(l+" "+f);
8)     }
9) }
```

Status: No Compilation Error

OUTPUT: 10 10.0

EX10:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         float f=22.22f;
6)         long l=f;
7)         System.out.println(f+" "+l);
8)     }
9) }
```

Status: Compilation Error, possible loss of precision.

Reason:

two class rooms

Class Room A: 25[Size] banches---> 3 members per bench---> 75

Class Room B: 50[Size] banches---> 1 member per bench----> 50

long---> 8 bytes---> less data as per its internal data arrangement.

float---> 4 bytes---> more data as per its internal data arrangement.

Due to the above reason, float data type is higher when compared with long data type so that, we are able to assign long variable to float variable directly, but, we are unable to assign float variable to long variable directly.



## Explicit Type Casting:

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

To perform explicit type casting we have to use the following pattern.

P a = (Q) b;

(Q) → Cast operator

Where P and Q are two primitive data types, where Q must be either same as P or lower than P as per implicit type casting chart.

EX1:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int i=10;
6)     byte b=(byte)i;
7)     System.out.println(i+ " "+b);
8)   }
9) }
```

Status: No Compilation Error

OUTPUT: 10 10

When we compile the above code, when compiler encounter the above assignment statement, compiler will check whether cast operator provided data type is compatible with left side variable data type or not, if not, compiler will rise an error like "Possible loss of precision". If cast operator provided data type is compatible with left side variable data type then compiler will not rise any error and compiler will not perform type casting.

When we execute the above program, when JVM encounter the above assignment statement then JVM will perform two actions.

- 1.JVM will convert right side variable data type to cast operator provided data type.
- 2.JVM will copy value from right side variable to left side variable.

EX2:

```
1) class Test
2) {
3)   public static void main(String[] args)
4) }
```



```
5) int i=10;
6) short s=(byte)i;
7) System.out.println(i+" "+s);
8) }
9) }
```

Status: No Compilation Error

OUTPUT: 10 10

EX3:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         byte b=65;
6)         char c=(char)b;
7)         System.out.println(b+" "+c);
8)     }
9) }
```

Status: No Compilation Error

OUTPUT: 65 A

EX4:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         char c='A';
6)         short s=(short)c;
7)         System.out.println(c+" "+s);
8)     }
9) }
```

Status: No Compilation Error

OUTPUT: A 65

**Note:** In Implicit type casting, conversions are not possible between char and byte, short and char, but, in explicit type casting conversions are possible in between char and byte, char and short, why because, explicit type casting is forceable type casting.



EX5:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         short s=65;
6)         char c=(byte)s;
7)         System.out.println(s+ " "+c);
8)     }
9) }
```

Status: COmpilation Error.

EX6:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         byte b1=30;
6)         byte b2=30;
7)         byte b=(byte)b1+b2;
8)         System.out.println(b);
9)     }
10) }
```

Status: Compilation Error

EX7:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         byte b1=30;
6)         byte b2=30;
7)         byte b=(byte)(b1+b2);
8)         System.out.println(b);
9)     }
10) }
```

Status: No Compilation Error

OUTPUT: 60



EX8:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         double d=22.22;
6)         byte b=(byte)(short)(int)(long)(float)d;
7)         System.out.println(b);
8)     }
9) }
```

Status: No Compilation Error

OUTPUT: 22

EX9:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int i=130;
6)         byte b=(byte)i;
7)         System.out.println(b);
8)     }
9) }
```

Status: No Compilation Error

OUTPUT: -126

Reason: REf Diagram

## 4.Java Statements:

Statement is the collection of expressions.

To design java applications JAVA has provided the following statements.

### General Purpose Statements

Declaring variables, methods, classes,....

Creating objects, accessing variables, methods,.....



## Conditional Statements:

1. if 2.switch

## Iterative Statements:

1.for 2.while 3.do-while

## Transfer statements:

1.break 2.continue 3.return

## Exception Handling statements:

throw, try-catch-finally

## Synchronized statements:

1.synchronized methods  
2.synchronized blocks

## Conditional Statements:

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

### EX:

1. if 2.switch

1. if:

### Syntax-1:

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

### Syntax-2:

```
if(condition)
{
    ---instuctions---
}
else
{
    ----instructions---
}
```

### Syntax-3:

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



```
else if(condition)
{
---instruction---
}
else if(condition)
{
---instructions---
}
-----
-----
else
{
---instructions---
}
```

EX1:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int i=10;
6)     int j;
7)     if(i==10)
8)     {
9)       j=20;
10)    }
11)    System.out.println(j);
12)  }
13) }
```

Status: Compilation Error, Variable j might not have been initialized.

EX2:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int i=10;
6)     int j;
7)     if(i==10)
8)     {
9)       j=20;
10)    }
11)    else
```



```
12)  {
13)      j=30;
14)  }
15)  System.out.println(j);
16) }
17} }
```

Status: No Compilation Error

OUTPUT: 20

EX3:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int i=10;
6)         int j;
7)         if(i==10)
8)         {
9)             j=20;
10)        }
11)        else if(i==20)
12)        {
13)            j=30;
14)        }
15)        System.out.println(j);
16)    }
17} }
```

Status: Compilation Error, Variable j might not have been initialized.

EX4:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int i=10;
6)         int j;
7)         if(i==10)
8)         {
9)             j=20;
10)        }
11)        else if(i==20)
12)        {
```



```
13)      j=30;
14)    }
15)  else
16)  {
17)      j=40;
18)  }
19)  System.out.println(j);
20) }
21} }
```

Status: no Compilation Error

OUTPUT: 20

EX5:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     final int i=10;
6)     int j;
7)     if(i == 10)
8)     {
9)       j=20;
10)    }
11)    System.out.println(j);
12)  }
13} }
```

Status: No Compilation Error

OUTPUT: 20

EX6:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int j;
6)     if(true)
7)     {
8)       j=20;
9)     }
10)    System.out.println(j);
11)  }
12} }
```



Status: No Compilation Error

## Reasons:

1. In java applications, only class level variables are having default values, local variables are not having default values. If we declare local variables in java applications then we must provide initializations for that local variables explicitly, if we access any local variable without having initialization explicitly then compiler will rise an error like "Variable x might not have been initialized".

### EX:

```
1) class A{  
2)     int i;----> class level variable, default value is 0..  
3)     void m1(){  
4)         int j;---> local variable, no default value.  
5)         System.out.println(i);// OUTPUT: 0  
6)         //System.out.println(j);--> Error  
7)         j=20;  
8)         System.out.println(j);---> No Error  
9)     }  
10) }
```

**Note:** Local variables must be declared in side methods, blocks, if conditions,... and these variables are having scope upto that method only, not having scope to outside of that method. Class level variables are declare at class level that is in out side of the methods, blocks,... these variables are having scope through out the classes that is in all methods, in all blocks which we provided in the respective class.

2. In java, there are two types of conditional Expressions.

1. Constant Expressions
2. Variable Expressions

## Constant Expressions:

These expressions includes only constants including final variables and these expressions would be evaluated by "Compiler" only, not by JVM.

### EX:

```
1.if( 10 == 10){ } ----> Constant Expression  
2.if( true ){ } -----> Constant Expression  
3.final int i=10;  
if( i == 10){ } ----> Constant Expression
```

**Note:** If we declare any variable as final variable with a value then compiler will replace final variables with their values in the remaining program, this process is called "Constant



---

Folding", it is one of the code optimization tech followed by Compiler.

## Variable Expressions:

These expressions are including at least one variable [not including final variables] and these expressions are evaluated by JVM, not by Compiler.

EX:

1.int i=10;  
int j=10;  
if(i == j){ } ----> variable expression.

2.int i=10;  
if(i == 10){ } ----> Variable expression

## switch

'if' is able to provide single condition checking by default, but, switch is able to provide multiple conditions checkings.

Syntax:

```
switch(Var)
{
    case 1:
        -----instructions-----
        break;
    case 2:
        -----instructions-----
        break;
        -----
        -----
    case n:
        -----instructions-----
        break;
    default:
        -----instructions-----
        break;
}
```

Note: We will utilize switch programming element in "Menu Driven" Applications.

## Stack Operations

- 1.PUSH
- 2.POP
- 3.PEEK
- 4.EXIT



Enter Your Option: 1  
Enter element to PUSH: A  
PUSH operation success

#### Stack Operations

1.PUSH

2.POP

3.PEEK

4.EXIT

Enter Your Option:2

POP operation success.

#### Stack Operations

1.PUSH

2.POP

3.PEEK

4.EXIT

Enter Your Option:3

PEEK Operation Success, TOUTPUT: A

#### Stack Operations

1.PUSH

2.POP

3.PEEK

4.EXIT

Enter Your Option:4

Thanks for using STACK Operations.

#### EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int i=10;
6)     switch(i)
7)     {
8)       case 5:
9)         System.out.println("Five");
10)      break;
11)      case 10:
12)        System.out.println("Ten");
13)      break;
14)      case 15:
15)        System.out.println("Fifteen");
16)      break;
```



```
17)     case 20:  
18)         System.out.println("Twenty");  
19)     break;  
20)     default:  
21)         System.out.println("Default");  
22)     break;  
23) }  
24}  
25}
```

## Rules to write switch:

1.switch is able to allow the data types like byte, short, int and char.

EX: byte b=10;

```
switch(b)  
{  
----  
}
```

Status: No Compilation Error

EX: long l=10;

```
switch(l)  
{  
----  
}
```

Status: Compilatiopn Error.

EX:

```
1) class Test  
2) {  
3)     public static void main(String[] args)  
4)     {  
5)         char c='B';  
6)         switch(c)  
7)         {  
8)             case 'A':  
9)                 System.out.println("Five");  
10)            break;  
11)            case 'B':  
12)                System.out.println("Ten");  
13)                break;  
14)            case 'C':  
15)                System.out.println("Fifteen");  
16)                break;
```



```
17)     case 'D':
18)         System.out.println("Twenty");
19)     break;
20)   default:
21)     System.out.println("Default");
22)   break;
23) }
24}
25}
```

Status: no Compilation Error

Note:

Upto JAVA6 version, switch is not allowing "String" data type as parameter, "JAVA7" version onwards switch is able to allow String data type.

EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     String str="BBB";
6)     switch(str)
7)     {
8)       case "AAA":
9)         System.out.println("AAA");
10)        break;
11)       case "BBB":
12)         System.out.println("BBB");
13)        break;
14)       case "CCC":
15)         System.out.println("CCC");
16)        break;
17)       case "DDD":
18)         System.out.println("DDD");
19)        break;
20)     default:
21)       System.out.println("Default");
22)     break;
23)   }
24}
25}
```

Status: No Compilation Error

OUTPUT: BBB



2. In switch, all cases and default are optional, we can write switch with out cases and with default, we can write switch with cases and with out default, we can write switch with out both cases and default.

EX:

```
int i=10;
switch(i)
{
}
Status: No Compilation Error
OUTPUT: No Output.
```

EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int i=10;
6)     switch(i)
7)     {
8)       default:
9)         System.out.println("Default");
10)        break;
11)      }
12)
13)  }
14} }
```

Status:No Compilation Error

OUTPUT: Default

EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int i=10;
6)     switch(i)
7)     {
8)       case 5:
9)         System.out.println("Five");
10)        break;
11)       case 10:
```



```
12)     System.out.println("Ten");
13)     break;
14) case 15:
15)     System.out.println("Fifteen");
16)     break;
17) case 20:
18)     System.out.println("Twenty");
19)     break;
20) }
21) }
22} }
```

Status: No Compilation Error

OUTPUT: Ten

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int i=50;
6)         switch(i)
7)         {
8)             case 5:
9)                 System.out.println("Five");
10)            break;
11)            case 10:
12)                System.out.println("Ten");
13)            break;
14)            case 15:
15)                System.out.println("Fifteen");
16)            break;
17)            case 20:
18)                System.out.println("Twenty");
19)            break;
20)        }
21)    }
22} }
```

Status: No Compilation Error

OUTPUT: No Output

In switch, "break" statement is optional, we can write switch without break statement, in this context, JVM will execute all the instructions continuously right from matched case until it encounters either break statement or end of switch.



EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int i=10;
6)         switch(i)
7)         {
8)             case 5:
9)                 System.out.println("Five");
10)
11)            case 10:
12)                System.out.println("Ten");
13)
14)            case 15:
15)                System.out.println("Fifteen");
16)
17)            case 20:
18)                System.out.println("Twenty");
19)
20)        default:
21)            System.out.println("Default");
22)        }
23)    }
24} }
```

Status: No Compilation Error

OUTPUT: Ten

Fifteen

Twenty

Default

In switch, all case values must be provided with in the range of the data type which we provided as parameter to switch.

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         byte b=126;
6)         switch(b)
7)         {
8)             case 125:
```



```
9)     System.out.println("125");
10)    break;
11)    case 126:
12)        System.out.println("126");
13)        break;
14)    case 127:
15)        System.out.println("127");
16)        break;
17)    case 128:
18)        System.out.println("128");
19)        break;
20)    default:
21)        System.out.println("Default");
22)        break;
23)    }
24) }
25}
```

Status: Compilation Error

In switch, all case values must be constants including final variables, they should not be normal variables.

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         final int i=5,j=10,k=15,l=20;
6)         switch(10)
7)         {
8)             case i:
9)                 System.out.println("Five");
10)                break;
11)            case j:
12)                System.out.println("Ten");
13)                break;
14)            case k:
15)                System.out.println("Fifteen");
16)                break;
17)            case l:
18)                System.out.println("Twenty");
19)                break;
20)            default:
21)                System.out.println("Default");
```



```
22)           break;
23)     }
24)   }
25) }
```

Status: No Compilation Error

OUTPUT: 10

Note: in the above example, if we remove final keyword then compiler will rise an error.

## Iterative Statements:

These statements are able to allow JVM to execute a set of instructions repeatedly on the basis of a particular condition.

EX: 1.for 2.while 3.do-while

### 1) for

Syntax: for(Expr1; Expr2; Expr3)  
{  
 ----instructions----  
}

EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     for(int i=0;i<10;i++)
6)     {
7)       System.out.println(i);
8)     }
9)   }
10} }
```

Status: No Compilation Error

Output: 0

```
1
---
---
9
```



Expr1----> 1 time  
Expr2----> 11 times  
Expr3----> 10 times  
Body ----> 10 times

EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int i=0;
6)     for(;i<10;i++)
7)     {
8)       System.out.println(i);
9)     }
10)  }
11} }
```

Status: No Compilation Error

OUTPUT: 0 ---- 9

EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     int i=0;
6)     for(System.out.println("Hello");i<10;i++)
7)     {
8)       System.out.println(i);
9)     }
10)  }
11} }
```

Status: No Compilation Error

OUTPUT: Hello 0 1 ---- 9

Reason:

In for loop, Expr1 is optional, we can write for loop even with out Expr1 , we can write any statement like System.out.println--> as Expr1, but, always, it is suggestible to provide loop variable declaration and initialization kind of statements as Expr1.



EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i=0, float f=0.0f ;i<10 && f<10.0f; i++,f++)
6)         {
7)             System.out.println(i+" "+f);
8)         }
9)     }
10) }
```

Status: Compilation Error

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i=0, int j=0 ;i<10 && j<10; i++,j++)
6)         {
7)             System.out.println(i+" "+j);
8)         }
9)     }
10) }
```

Status: Compilation Error

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i=0, j=0 ;i<10 && j<10; i++,j++)
6)         {
7)             System.out.println(i+" "+j);
8)         }
9)     }
10) }
```

Status: No Compilation Error



---

**OUTPUT:** 0 0  
1 1  
2 2  
-----  
-----  
9 9

## Reason:

In for loop, Expr1 is able to allow at most one declarative statement, it will not allow more than one declarative statement, we can declare more than one variable within a single declarative statement.

## EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     for(int i=0; ;i++)
6)     {
7)       System.out.println(i);
8)     }
9)   }
10}
```

**Status:** No Compilation Error

**OUTPUT:** Infinite Loop

## EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     for(int i=0; System.out.println("Hello") ;i++)
6)     {
7)       System.out.println(i);
8)     }
9)   }
10}
```

**Status:** Compilation Error

**Reason:** In for loop, Expr2 is optional, we can write for loop even without Expr2, if we write for loop without Expr2 then for loop will take "true" value as Expr2 and it will make



for loop as an infinite loop. If we want to write any statement as Expr2 then that statement must be boolean statement, it must return either true value or false value.

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         System.out.println("Before Loop");
6)         for(int i=0;i<=0 || i>=0 ;i++)
7)         {
8)             System.out.println("Inside Loop");
9)         }
10)        System.out.println("After Loop");
11)    }
12) }
```

Status: No Compilation Error

OUTPUT: Before Loop  
Inside Loop  
----  
----

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         System.out.println("Before Loop");
6)         for(int i=0; true ;i++)
7)         {
8)             System.out.println("Inside Loop");
9)         }
10)        System.out.println("After Loop");
11)    }
12) }
```

Status: Compilation Error, Unreachable Statement



EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         System.out.println("Before Loop");
6)         for(int i=0;i++)
7)         {
8)             System.out.println("Inside Loop");
9)         }
10)        System.out.println("After Loop");
11)    }
12) }
```

Status: Compilation Error, Unreachable Statement

Reasons:

In java applications, if we provide any statement immediately after infinite loop then that statement is called as "Unreachable Statement". If compiler identifies the provided loop as an infinite loop and if compiler identifies any followed statement for that infinite loop then compiler will rise an error like "Unreachable Statement". If compiler does not aware the provided loop as an infinite loop then there is no chance for compiler to raise "Unreachable Statement Error".

**Note:** Deciding whether a loop as an infinite loop or not is completely depending on the conditional expression, if the conditional expression is constant expression and it returns true value always then compiler will recognize the provided loop as an infinite loop. If the conditional expression is variable expression then compiler will not recognize the provided loop as an infinite loop even the loop is really infinite loop.

```
int i=0;
for(int i=10; i<10;i++)
{
}
```

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i=0;i<10)
6)         {
7)             System.out.println(i);
8)         }
9)     }
10) }
```



```
8)      i=i+1;  
9)    }  
10)   }  
11) }
```

Status: No Compilation Error

OUTPUT: 0 ,1 ..... 9

EX:

```
1) class Test  
2) {  
3)   public static void main(String[] args)  
4)   {  
5)     for(int i=0;i<10;System.out.println("Hello"))  
6)     {  
7)       System.out.println(i);  
8)       i=i+1;  
9)     }  
10)   }  
11) }
```

Status: No Compilation Error

Output: 0

```
Hello  
1  
Hello  
----  
----  
9  
Hello
```

Note: In for loop, Expr3 is optional, we can write for loop without expr3, we can provide any statement as expr3, but, it is suggestible to provide loop variable increment/decrement kind of statements as expr3.

EX:

```
1) class Test {  
2)   public static void main(String[] args) {  
3)     for(:;)  
4)   }  
5) }
```



Status: Compilation Error

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(;;);
6)     }
7) }
```

Status: No Compilation Error

OUTPUT: No Output, but, JVM will be in infinite loop

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(;;)
6)     {
7)     }
8) }
9) }
```

Status: No Compilation Error

OUTPUT: No Output, but, JVM will be in infinite loop

**Reason:** In for loop, if we want to write single statement in body then curly braces [{ }] are optional, if we don't want to write any statement as body then we must provide either ; or curly braces to the for loop.

In general, we will utilize for loop when we aware no of iterations in advance before writing loop.

EX:

```
1) int[] a={1,2,3,4,5};
2) int size=a.length;
3) for(int i=0; i<size; i++)
4) {
5)     System.out.println(a[i]);
6) }
```



If we use above for loop to retrieve elements from arrays and from Collection objects then we are able to get the following problems.

- 1) We have to manage a separate loop variable.
- 2) At each and every iteration we have to execute a conditional expression that is expr2, which is more strenghful operation and it may consume more no of system resources [Memory and execution time].
- 3) At each and every iteration we have to perform either increment operation or decrement operation explicitly.
- 4) In this approach, we are retrieving elements from arrays on the basis of the index value , if it is not proper then there may be a chance to get an exception like "java.lang.ArrayIndexOutOfBoundsException".

The above drawbacks are able to reduce java application performance.

To overcome the above problems and to improve java application performance explicitly we have to use "for-Each" loop provided by JDK5.0 version.

### Syntax:

```
for(Array_Data_Type var: Array_Ref_Var)
{
    ----
}
```

### EX:

```
1) int[] a={1,2,3,4,5};
2) for(int x: a)
3) {
4)     System.out.println(x);
5) }
```

### EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int[] a={1,2,3,4,5};
6)         for(int i=0;i<a.length;i++)
7)         {
8)             System.out.println(a[i]);
9)         }
}
```



```
10) System.out.println();
11) for(int x: a)
12) {
13)     System.out.println(x);
14) }
15)
16}
```

Status: No Compilation Error

### Output:

```
1
2
3
4
5
```

```
1
2
3
4
5
```

## 2) While Loop:

In java applications, when we are not aware the no of iterations in advance before writing loop there we have to utilize 'while' loop.

### EX:

```
D:\javaapps>java Insert.java
Enter Employee Number : 111
Enter Employee Name : AAA
Enter Employee Salary : 5000
Enter Employee Address: Hyd
Employee Inserted Successfully
One more Employee[yes/no]? :yes
```

```
Enter Employee Number : 222
Enter Employee Name : BBB
Enter Employee Salary : 6000
Enter Employee Address: Hyd
Employee Inserted Successfully
One more Employee[yes/no]? : yes
```

```
Enter Employee Number : 333
Enter Employee Name : CCC
```



Enter Employee Salary : 7000  
Enter Employee Address: Hyd  
Employee Inserted Successfully  
One more Employee[yes/no]? no

--- Thank You for using this appl.----

Syntax:

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

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int i=0;
6)         while(i<10)
7)         {
8)             System.out.println(i);
9)             i=i+1;
10)        }
11)    }
12) }
```

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int i=0;
6)         while()
7)         {
8)             System.out.println(i);
9)             i=i+1;
10)        }
11)    }
12) }
```

Status: Compilation Error

Reason: Conditional Expression is mandatory.



EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         System.out.println("Before Loop");
6)         while(true)
7)         {
8)             System.out.println("Inside Loop");
9)         }
10)        System.out.println("After Loop");
11)    }
12} }
```

Status: Compilation Error, Unreachable Statement.

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         System.out.println("Before Loop");
6)         int i=0;
7)         while(i<=0 || i>=0)
8)         {
9)             System.out.println("Inside Loop");
10)        }
11)        System.out.println("After Loop");
12)    }
13} }
```

### 3) do-while:

**Q) What are the differences between while loop and do-while loop?**

- 1) While loop is not giving any guarantee to execute loop body minimum one time. do-while loop will give guarantee to execute loop body minimum one time.
- 2) In case of while, first, conditional expression will be executes, if it returns true then only loop body will be executed.



In case of do-while loop, first loop body will be executed then condition will be executed.

3) In case of while loop, condition will be executed for the present iteration.

In case of do-while loop, condition will be executed for the next iteration.

## Syntaxes:

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

```
do
{
    ---instructions---
}
```

## While(Condition):

### EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int i=0;
6)         do
7)         {
8)             System.out.println(i);
9)             i=i+1;
10)        }
11)        while (i<10);
12)    }
13} }
```

Status: No Compilation Error

OUTPUT: 0, 1, 2,..., 9



EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         System.out.println("Before Loop");
6)         do
7)         {
8)             System.out.println("Inside Loop");
9)         }
10)        while (true);
11)        System.out.println("After Loop");
12)    }
13} }
```

Status: Compilation Error, Unreachable Statement

## 4.Transfer Statements:

These statements are able to bypass flow of execution from one instruction to another instruction.

EX: 1.break 2.continue 3.return

### 1) break:

break statement will bypass flow of execution to outside of the loops or outside of the blocks by skipping the remaining instructions in the current iteration and by skipping all the remaining iterations.

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i=0;i<10;i++)
6)         {
7)             if(i==5)
8)             {
9)                 break;
10)            }
11)            System.out.println(i);
12)        }
13} }
```



| 14) }

Status: No Compilation Error

**Output:**

```
0  
1  
2  
3  
4
```

**EX:**

```
1) class Test  
2) {  
3)   public static void main(String[] args)  
4)   {  
5)     System.out.println("Before loop");  
6)     for(int i=0;i<10;i++)  
7)     {  
8)       if(i==5)  
9)       {  
10)         System.out.println("Inside loop, before break");  
11)         break;  
12)         System.out.println("Inside loop, after break");  
13)       }  
14)     }  
15)     System.out.println("After Loop");  
16)   }  
17} }
```

Status: Compilation Error, unreachable Statement

Reason: If we provide any statement immediately after break statement then that statement is Unreachable Statement, where compiler will rise an error.

**EX:**

```
1) class Test  
2) {  
3)   public static void main(String[] args)  
4)   {  
5)     for(int i=0;i<10;i++)// Outer loop  
6)     {  
7)       for(int j=0;j<10;j++)// Nested Loop  
8)     {
```



```
9)     if(j==5)
10)    {
11)        break;
12)    }
13)    System.out.println(i+" "+j);
14) }
15) // Out side of nested Loop
16) }
17) }
18} }
```

Status: No Compilation Error

Output:

```
0 0
0 1
0 2
0 3
0 4
-----
-----
-----
9 0
9 1
9 2
9 3
9 4
```

Note: If we provide "break" statement in nested loop then that break statement is applicable for only nested loop, it will not give any effect to outer loop.

In the above context, if we want to give break statement effect to outer loop , not to the nested loop then we have to use "Labelled break" statement.

Syntax:

```
break label;
```

Where the provided label must be marked with the respective outer loop.

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         l1:for(int i=0;i<10;i++)// Outer loop
6)     {
```



```
7)     for(int j=0;j<10;j++)// Nested Loop
8)     {
9)         if(j==5)
10)        {
11)            break l1;
12)        }
13)        System.out.println(i+" "+j);
14)    }
15)    // Out side of nested Loop
16) }
17) //Out side of outer loop
18} }
19}
```

Status: no Compilation Error

OUTPUT: 0 0

```
0 1
0 2
0 3
0 4
```

## 2) Continue:

This transfer statement will bypass flow of execution to starting point of the loop by skipping all the remaining instructions in the current iteration inorder to continue with next iteration.

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i=0;i<10;i++)
6)         {
7)             if(i == 5)
8)             {
9)                 continue;
10)            }
11)            System.out.println(i);
12)        }
13)    }
14} }
```

Status: No Compilation Error



## OUTPUT:

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

## EX:

```
1) class Test  
2) {  
3)   public static void main(String[] args)  
4)   {  
5)     System.out.println("before Loop");  
6)     for(int i=0;i<10;i++)  
7)     {  
8)       if(i == 5)  
9)       {  
10)         System.out.println("Inside Loop, before continue");  
11)         continue;  
12)         System.out.println("Inside Loop, After continue");  
13)       }  
14)     }  
15)     System.out.println("After loop");  
16)   }  
17} }
```

Status: Compilation Error, Unreachable Statement.

Reason: If we provide any statement immediately after continue statement then that statement is unreachable statement, where compiler will rise an error.

## EX:

```
1) class Test  
2) {  
3)   public static void main(String[] args)  
4)   {  
5)     for(int i=0;i<10;i++)  
6)     {  
7)       for(int j=0;j<10;j++)
```



```
8)    {
9)        if(j==5)
10)       {
11)           continue;
12)       }
13)       System.out.println(i+" "+j);
14)   }
15) }
16) }
17} }
```

Status: No Compilation Error

OUTPUT:

---

--

---

--

If we provide continue statement in nested loop then continue statement will give effect to nested loop only, it will not give effect to outer loop

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i=0;i<10;i++)
6)         {
7)             for(int j=0;j<10;j++)
8)             {
9)                 if(j==5)
10)                {
11)                    continue;
12)                }
13)                System.out.println(i+" "+j);
14)            }
15)        }
16)    }
17} }
```

In the above context, if we want to give continue statement effect to outer loop, not to the nested loop then we have to use labelled continue statement.



## Syntax:

continue label;

Where the provided label must be marked with the respective outer loop.

## EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     l1:for(int i=0;i<10;i++)
6)     {
7)       for(int j=0;j<10;j++)
8)       {
9)         if(j==5)
10)        {
11)          continue l1;
12)        }
13)        System.out.println(i+" "+j);
14)      }
15)    }
16)  }
17) }
```

Status: No Compilation Error

## OUTPUT:

```
0 0
0 1
0 2
0 3
0 4
1 0
1 1
1 2
1 3
1 4
---
---
---
9 0
0 1
9 2
9 3
9 4
```



# PATTERN

S



## 1) Write Java programs to display the following patterns?

1)

```
* * * * * * * * * *  
* * * * * * * * * *  
* * * * * * * * * *  
* * * * * * * * * *  
* * * * * * * * * *  
* * * * * * * * * *  
* * * * * * * * * *  
* * * * * * * * * *  
* * * * * * * * * *  
* * * * * * * * * *
```

EX:

```
1) class Test  
2) {  
3)     public static void main(String[] args)  
4)     {  
5)         for(int i = 0; i < 10; i++)  
6)         {  
7)             for(int j = 0; j < 10; j++)  
8)             {  
9)                 System.out.print("*"+ " ");  
10)            }  
11)            System.out.println();  
12)        }  
13)    }  
14) }
```

2)

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

EX:

```
1) class Test  
2) {  
3)   public static void main(String[] args)  
4)   {  
5)     for(int i = 0; i < 10; i++)  
6)     {  
7)       for(int j = 0; j < 10; j++)  
8)       {  
9)         System.out.print(j+ " ");  
10)      }  
11)    System.out.println();  
12)  }  
13) }  
14} }
```

3)

```
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
```

EX:

```
1) class Test
```



```
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 10; i++)
6)         {
7)             for(int j = 0; j < 10; j++)
8)             {
9)                 System.out.print(9-j+" ");
10)            }
11)            System.out.println();
12)        }
13)    }
14} }
```

4)

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

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 10; i++)
6)         {
7)             for(int j = 0; j < 10; j++)
8)             {
9)                 System.out.print(i+j+" ");
10)            }
11)            System.out.println();
12)        }
13)    }
14} }
```

5)

```
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
```

EX:

```
1) class Test  
2) {  
3)   public static void main(String[] args)  
4)   {  
5)     for(int i = 0; i < 10; i++)  
6)     {  
7)       for(int j = 0; j < 10; j++)  
8)       {  
9)         System.out.print(9-i+" ");  
10)      }  
11)    System.out.println();  
12)  }  
13) }  
14} }
```

6)

```
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
```

EX:

```
1) class Test
```



```
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 10; i++)
6)         {
7)             for(int j = 0; j < 10; j++)
8)             {
9)                 System.out.print((char)(65+j)+" ");
10)            }
11)            System.out.println();
12)        }
13)    }
14} }
```

7)

```
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
```

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 10; i++)
6)         {
7)             for(int j = 0; j < 10; j++)
8)             {
9)                 System.out.print((char)(74-j)+" ");
10)            }
11)            System.out.println();
12)        }
13)    }
14} }
```

8)

```
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
```

EX:

```
1) class Test  
2) {  
3)   public static void main(String[] args)  
4)   {  
5)     for(int i = 0; i < 10; i++)  
6)     {  
7)       for(int j = 0; j < 10; j++)  
8)       {  
9)         System.out.print((char)(65+i)+ " ");  
10)      }  
11)    System.out.println();  
12)  }  
13) }  
14} }
```

9)

```
 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
```

EX:

```
1) class Test
```



```
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 10; i++)
6)         {
7)             for(int j = 0; j < 10; j++)
8)             {
9)                 System.out.print((char)(74-i)+ " ");
10)            }
11)            System.out.println();
12)        }
13)    }
14} }
```

## 2) Write Java programs to display the following patterns?

1)

```
*  
* *  
* * *  
* * * *  
* * * * *  
* * * * * *  
* * * * * * *  
* * * * * * * *  
* * * * * * * * *
```

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 10; i++)
6)         {
7)             for(int j = 0; j <= i; j++)
8)             {
9)                 System.out.print("**"+ " ");
10)            }
11)            System.out.println();
12)        }
13)    }
14} }
```



### 3) Write Java programs to display the following patterns?

1)

```
    *
   * *
  * * *
 * * * *
* * * * *
* * * * * *
* * * * * * *
* * * * * * * *
* * * * * * * * *
```

EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     for(int i = 0; i < 10; i++)
6)     {
7)       for(int j = 0; j < (9-i); j++)
8)       {
9)         System.out.print(" ");
10)      }
11)      for(int k = 0; k <= i; k++)
12)      {
13)        System.out.print(" * " + " ");
14)      }
15)      System.out.println();
16)    }
17)  }
18) }
```

### 4) Write Java programs to display the following patterns?

1)

```
    * * * * * * * * *
   * * * * * * * *
  * * * * * * *
 * * * * * *
* * * * *
* * * *
* * *
* *
```



EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 10; i++)
6)         {
7)             for(int j = 0; j < i; j++)
8)             {
9)                 System.out.print(" " + " ");
10)            }
11)            for(int k = 0; k < (10-i); k++)
12)            {
13)                System.out.print("*" + " ");
14)            }
15)            System.out.println();
16)        }
17)    }
18} }
```

5)

```
* * * * * * * * * *
* * * * * * * * *
* * * * * * * *
* * * * * * *
* * * * *
* * * *
* * *
* *
*
```

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 10; i++)
6)         {
7)             for(int j = 0; j < (10-i); j++)
8)             {
9)                 System.out.print("*" + " ");
10)            }
11)            System.out.println();
```



12) }  
13) }  
14) }

6)

A triangular pattern of asterisks arranged in seven rows. The pattern is centered and has a single asterisk at the top. Each subsequent row contains one more asterisk than the previous one, creating a symmetrical shape.

**EX:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 10; i++)
6)         {
7)             for(int j = 0; j < (9-i); j++)
8)             {
9)                 System.out.print(" ");
10)            }
11)            for(int k = 0; k <= i; k++)
12)            {
13)                System.out.print("*"+ " ");
14)            }
15)            System.out.println();
16)        }
17)    }
18) }
```

7)

\*

\* \* \*

\* \* \* \* \*

\* \* \* \* \* \*

**EX:**

## 1) class Test



```
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 5; i++)
6)         {
7)             for(int j = 0; j < (4-i); j++)
8)             {
9)                 System.out.print(" ");
10)            }
11)            for(int k = 0; k < (2*i+1); k++)
12)            {
13)                System.out.print("*");
14)            }
15)            System.out.println();
16)        }
17)    }
18} }
```

8)

```
*  
**  
****  
*****  
*****  
*****
```

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 5; i++)
6)         {
7)             for(int j = 0; j < (4-i); j++)
8)             {
9)                 System.out.print(" ");
10)            }
11)            for(int k = 0; k < 2 * (i + 1); k++)
12)            {
13)                System.out.print("*");
14)            }
15)            System.out.println();
16)        }
17)    }
18} }
```



9) \* \* \* \* \* \* \* \* \*  
\* \* \* \* \* \* \* \* \*  
\* \* \* \* \* \* \* \* \*  
\* \* \* \* \* \* \* \*  
\* \* \* \* \* \* \* \*  
\* \* \* \* \* \* \*  
\* \* \*  
\* \*

**EX:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 10; i++)
6)         {
7)             for(int j = 0; j < i; j++)
8)             {
9)                 System.out.print(" ");
10)            }
11)            for(int k = 0; k < (10-i); k++)
12)            {
13)                System.out.print("*"+ " ");
14)            }
15)            System.out.println();
16)        }
17)    }
18) }
```

**10) Write Java programs to display the following patterns?**

\* \* \* \* \* \* \* \*  
\* \* \* \* \* \* \*  
\* \* \* \* \* \*  
\* \* \*  
\* \*

**EX:**

## 1) class Test



```
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 5; i++)
6)         {
7)             for(int j = 0; j < i; j++)
8)             {
9)                 System.out.print(" ");
10)            }
11)            for(int k = 0; k < (10 - 2*i); k++)
12)            {
13)                System.out.print("*");
14)            }
15)            System.out.println();
16)        }
17)    }
18) }
```

## 11) Write Java Programs to display the following patterns?

```
*****
 *****
 ****
 ***
 *
 *
```

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 5; i++)
6)         {
7)             for(int j = 0; j < i; j++)
8)             {
9)                 System.out.print(" ");
10)            }
11)            for(int k = 0; k < 10-2*i-1; k++)
12)            {
13)                System.out.print("*");
14)            }
15)            System.out.println();
16)        }
17)    }
18) }
```



**12) Write Java programs to display the following patterns?**

A grid pattern of asterisks arranged in 12 rows and 10 columns. The grid is composed of two distinct parts: a larger inner rectangle of 10 columns by 9 rows, and a smaller outer border of 10 columns by 1 row. The asterisks are black and have a consistent size and spacing throughout the pattern.

**EX:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 10; i++)
6)         {
7)             for(int j = 0; j < (9-i); j++)
8)             {
9)                 System.out.print(" ");
10)            }
11)        }
12)    }
13) }
```



```
10)    }
11)    for(int k = 0; k <= i; k++)
12)    {
13)        System.out.print("*" + " ");
14)    }
15)    System.out.println();
16)    }
17)    for(int i = 0; i < 9; i++)
18)    {
19)        for(int j = 0; j <= i; j++)
20)        {
21)            System.out.print(" ");
22)        }
23)        for(int k = 0; k < (9 - i); k++)
24)        {
25)            System.out.print("*" + " ");
26)        }
27)        System.out.println();
28)    }
29) }
30} }
```

### 13) Write Java program to display the following patterns?

```
*****
*****
*****
*****
*****
 ****
 ****
 ****
 ****
 ****
```

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 5; i++)
6)         {
7)             for(int j = 0; j < 10; j++)
8)             {
9)                 System.out.print("*");
10)            }
```



```
11)     System.out.println();
12) }
13) for(int i = 0; i < 5; i++)
14) {
15)     for(int j = 0; j < 3; j++)
16)     {
17)         System.out.print(" ");
18)     }
19)     for(int k = 0; k < 4; k++)
20)     {
21)         System.out.print("*");
22)     }
23)     System.out.println();
24) }
25)
26}
```

Q) Write a Java program to display the following patterns?

```

*
* *
* * *
* * * *
* * * * *
* * * * * *
* * * * * * *
* * * *
* * *
* * *
* * *
* * *
* * *
* * *
```

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         for(int i = 0; i < 8; i++)
6)         {
7)             for(int j = 0; j < (8 - i); j++)
8)             {
9)                 System.out.print(" ");
10)            }
```



```
11)    for(int k = 0; k <= i; k++)
12)    {
13)        System.out.print("*"+ " ");
14)    }
15)    System.out.println();
16) }
17) for(int i = 0; i < 7; i++)
18) {
19)     for(int j = 0; j < 3; j++)
20)     {
21)         System.out.print(" "+ " ");
22)     }
23)     for(int k = 0; k < 3; k++)
24)     {
25)         System.out.print("*"+ " ");
26)     }
27)     System.out.println();
28) }
29) }
30} }
```



# Object Orientation



## Object Orientation

To prepare applications we have to select a particular type of programming language from the following.

- 1) Unstructured Programming Languages
- 2) Structured Programming Languages
- 3) Object Oriented Programming Languages
- 4) Aspect Oriented Programming Languages

### Q) What are the differences between Unstructured Programming Languages and Structured Programming Languages?

- 1) Unstructured Programming Languages are out dated programming languages, they were introduced at starting point of computers and these programming languages are not suitable for our at present application requirements.  
EX: BASIC, FOTRAN,.....

Structured Programming languages are not out dated programming languages, these programming languages are utilized for our at present application requirements.  
EX: C , PASCAL,....

- 2) Unstructured Programming Languages are not following any proper structure to prepare applications.

Structured Programming Languages are following a particular structure to prepare applications.

- 3) Unstructured Programming languages are using mnemonic [ADD, SUB, MUL,..] [low level code] codes to prepare applications, which are available in very less number and which may provide very less no of features to prepare applications.



Structured Programming Languages are using high level syntaxes, which are available in more number and which may provide more no of features to the applications.

- 4) Unstructured Programming Languages are using only "goto" statement to define flow of execution, which is not sufficient to provide very good flow of execution in applications.

Structured Programming Languages are using more and more no of flow controllers like if, switch, for, while, do-while, break,..... to defined very good flow of execution in applications.

- 5) Unstructured Programming languages are not having functions feature, it will increase code redundancy [Code duplication], it is not suggestible in application development.

Structured Programming languages are having functions feature to improve code reusability.

## **Q) What are the differences between Structured Programming Languages and Object Oriented Programming Languages?**

- 1) Structured Programming Languages are providing difficult approach to prepare applications.

EX: C, PASCAL,....

Object Oriented Programming Languages are providing very simplified approach to prepare applications.

EX: JAVA, C++,....

- 2) Structured Programming languages are not having modularity.

Object Oriented Programming Languages are having very good modularity.

- 3) Structured Programming languages are not having very good abstraction.

Object Oriented Programming Languages are having very good abstraction levels.

- 4) Structured Programming languages are not providing very good security for the applications.

Object oriented programming Languages is providing very good security for the application data.

- 5) Structured Programming languages are not providing very good sharability.

Object oriented programming Languages are providing very good sharability.

- 6) Structured Programming Languages are not providing very good code reusability.



---

Object oriented programming languages are providing very good code reusability.

## Aspect Oriented Programming Languages:

If we want to prepare applications by using Object orientation then we have to provide both business logic and Services logic in combined manner, it will provide tightly coupled design, it will reduce sharability and code reusability.

To overcome the above problem we have to use Aspect Orientation. Aspect Orientation is a methodology or a set of rules and regulations or a set of guide lines which are applied on object oriented programming to get loosely coupled design in order to improve sharability and code reusability.

In case of Aspect orientation, we will separate all services from business logic, we will declare each and every service as an aspect and we will inject these services to the applications at runtime as per the requirement.

## Object Oriented Features:

To show the nature of Object orientation, Object Orientation has provided the following features.

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

There are two types of programming languages on the basis of object oriented features.

- 1) Object Oriented Programming Languages
- 2) Object based Programming Languages

### Q) What is the difference between Object Oriented Programming Languages and Object Based programming languages?

- Object Oriented Programming languages are able to allow all the object oriented features including "Inheritance".

EX: Java



- Object based programming languages are able to allow all the object oriented features excluding "Inheritance".  
EX: Java Script

## Q) What are the differences between class and Object?

- 1) Class is a group of elements having common properties and behaviors.  
Object is an individual element among the group of elements having physical behaviors and physical properties
- 2) Class is virtual.  
Object is real
- 3) Class is virtual encapsulation of properties and behaviors.  
Object is physical encapsulation of properties and behaviors.
- 4) Class is Generalization.  
Object is Specialization.
- 5) Class is Model or Blue Print for the objects.  
Object is an instance of the class.

## Q) What is the difference between Encapsulation and Abstraction?

The process of binding data and coding part is called as "Encapsulation".

The process showing necessary data or implementation and hiding unnecessary data or implementation is called as "Abstraction".

Note: In Object Oriented programming languages, both encapsulation and abstraction are improvising "Security".

## 5. Inheritance:

The process of getting variables and methods from one class to another class is called as "Inheritance".

The main objective of inheritance is "Code Reusability".



EX:

```
1) class Employee {  
2)     String eid;  
3)     String ename;  
4)     String eaddr;  
5)     ---  
6)     ---  
7)     void getEmpDetails() {  
8)         ----  
9)     }  
10) }  
11) class Manager extends Employee {  
12)     --- Reuse variables and methods of Employee class---  
13) }  
14) class Accountant extends Employee {  
15)     --- Reuse variables and methods of Employee class here----  
16) }  
17) class Engineer extends Employee {  
18)     --- Reuse variables and methods of Employee class here----  
19) }
```

## 6.Polymorphism:

- Polymorphism is "Greak" word, where poly means many and morphism means structures [forms].
- If one thing is existed in multiple forms then it is called as Polymorphism.
- The main advantage of Polymorphism is "Flexibility" to design application.

## 7.Message Passing:

The process of transferring data along with flow of execution from one instruction to another instructions is called as Message passing.

The main advantage of message passing is to improve "Communication" between entities and "data navigation" between entities.

## Containers in Java:

Container is a Java element, it able to manage some other programming elements like variables, methods, blocks,.....

There are three types of containers in JAVA.



- 
- 1) Class
  - 2) Abstract Class
  - 3) Interface

## 1) Class:

The main Purpose of the class is to represent all real world entities in Java applications.  
EX: Student, Employee, Product, Account,....

To represent entities data in Java applications we will use Variables.  
To represent entities behaviours we will use methods.

Syntax:

```
[Access_Modifiers] class Class_Name [extends Super_Class_Name]  
    [implements interface_List]  
{  
    --- variables----  
    --- methods-----  
    --- constructors----  
    --- blocks----  
    --- classes-----  
    --- abstract classes----  
    --- interfaces-----  
    --- enums-----  
}
```

### Access Modifiers:

There are two types of Access modifiers

1. To define scopes to the programming elements, there are four types of access modifiers. public, protected, <default> and private.  
Where public and <default> are allowed for classes, protected and private are not allowed for classes.

**Note:** All public, protected , <default> and private are allowed for inner classes.

**Note:** Where private and protected access modifiers are defined on the basis of classes boundaries, so that, they are not applicable for classes, they are applicable for members of the classes including inner classes.



2. To define some extra nature to the programming elements , we have the following access modifiers.

static, final, abstract, native, volatile, transient, synchronized, strictfp,.....

Where final, abstract and strictfp are allowed for the classes.

**Note:** The access modifiers like static, abstract, final and strictfp are allowed for inner classes.

**Note:** Where static keyword was defined on the basis of classes origin, so that, it is not applicable for classes, it is applicable for members of the classes including inner classes.

Where 'class' is a java keyword, it can be used to represent 'Class' object oriented feature.

Where 'Class\_Name' is name is an identifier, it can be used to recognize the classes individually.

Where 'extends' keyword is able to specify a particular super class name in class syntax in order to get variables and methods from the specified super class to the present java class.

**Note:** In class syntax, 'extends' keyword is able to allow only one super class name, it will not allow more than one super class, because, it will represent multiple inheritance, it is not possible in Java.

Where 'implements' keyword is able to specify one or more no of interfaces in class syntax in order to provide implementation for all abstract methods of that interfaces in the present class.

**Note:** In class syntax, extends keyword is able to allow only one super class, but, implements keyword is able to allow more than one interface.

**Note:** In class syntax, 'extends' and 'implements' keywords are optional, we can write a class with extends keyword and without implements keyword, we can write a class with implements keyword and without extends keyword, we can write a class without both extends and implements keywords, if we want to write both extends and implements keywords first we have to write extends keyword then only we have to write implements keyword, we must not interchange extends and implements keywords.

- 1) class A{ } ----> valid
- 2) public class A{ }----> valid
- 3) private class A{ }----> Invalid
- 4) protected class A{ }----> Invalid



- 5) class A{ public class B{ } }----> valid
- 6) class A{ private class B{ } }----> valid
- 7) class A{ protected class B { } }----> valid
- 8) class A{ class B{ } }----> valid
- 9) static class A{ }----> Invalid
- 10) final class A{ }----> valid
- 11) abstract class A{ }----> valid
- 12) synchronized class A{ }----> Invalid
- 13) native class A{ }----> Invalid
- 14) class A{ static class B{ } }----> valid
- 15) class A{ abstract class B{ } }----> valid
- 16) class A{ strictfp class B{ } }----> valid
- 17) class A{ native class B{ } }----> Invalid
- 18) class A extends B{ }----> valid
- 19) class A extends A{ }----> Invalid, cyclic inheritance Error
- 20) class A extends B, C{ }----> Invalid
- 21) class A implements I{ }----> valid
- 22) class A implements I1, I2{ }----> valid
- 23) class A implements I extends B{ }----> Invalid
- 24) class A extends B implements I{ }----> valid
- 25) class A extends B implements I1, I2{ }----> valid

## Procedure To Write Classes In Java Applications:

- 1) Declare a class by using 'class' keyword.
- 2) Declare variables and methods in class as per the requirement.
- 3) In main class and in main() method, create object for the respective class.
- 4) Access members of the class by using the generated reference variable.

Entities[Student, Customer, Employee,...]----> classes

Entities data[sid, sname, saddr,...] ----> variables

Entities actions or behaviours[add, search, delete,..] ----> methods

EX:

```
1) class Employee
2) {
3)     String eid="E-111";
4)     String ename="Durga";
5)     float esal=25000.0f;
6)     String eaddr="Hyderabad";
7)     String eemail="durga@durgasoft.com";
8)     String emobile="91-9988776655";
9)     public void display_Emp_Details() {
10)         System.out.println("Employee Details");
11)         System.out.println("-----");
```



```
12) System.out.println("Employee Id :"+eid);
13) System.out.println("Employee Name :"+ename);
14) System.out.println("Employee Saslary:"+esal);
15) System.out.println("Employee Address:"+eaddr);
16) System.out.println("Employee Email :" +eemail);
17) System.out.println("Employee Mobile :" +emobile);
18) }
19) }
20) class Test
21) {
22)     public static void main(String[] args)
23)     {
24)         Employee emp = new Employee();
25)         emp.display_Emp_Details();
26)     }
27) }
```

There are two types of methods.

- 1) Concrete methods
- 2) Abstract methods

## Q) What are the differences between *Concrete methods* and *abstract methods*?

- 1) Concrete method is a method, it will have both method declaration and method implementation.

EX: void add(int i, int j)//Method Declaration  
    // Method implementation started here  
        int k=i+j;  
        System.out.println(k);  
    // Method Implementation ended here

Abstract method is a method, it will have only method declaration.

EX: abstract void add(int i, int j);

- 2) To declare concrete methods, no need to use any special keyword.  
To declare abstract method, we must use abstract keyword.
- 3) Concrete methods are allowed in classes and in abstract classes.  
Abstract methods are allowed in abstract classes and interfaces.
- 4) Concrete methods will provide less sharability.  
Abstract methods will provide more sharability.

## 2) Abstract Classes:



**Abstract class is a java class, it able to allow zero or more no of concrete methods and zero or more no of abstract methods.**

**Note:** To declare abstract classes, it is not at all mandatory condition to have at least one abstract method, we can declare abstract classes with 0 no of abstract methods, but, if we want to declare a method as an abstract method then the respective class must be abstract class.

For abstract classes, we are able to create only reference variables; we are unable to create objects.

```
abstract class A {  
    ---  
}  
A a = new A();--> Error  
A a = null; → No Error
```

Abstract classes are able to provide more sharability when compared with classes.

## **Procedure to use Abstract Classes:**

- 1) Declare an abstract class with "abstract".
- 2) Declare concrete methods and abstract methods in abstract class as per the requirement.
- 3) Declare sub class for abstract class.
- 4) Implement abstract methods in sub class.
- 5) In main class and in main() method, create object for sub class and declare reference variable either for abstract class or for sub class.
- 6) Access abstract class members.

**Note:** If we declare reference variable for abstract class then we are able to access only abstract class members, but, if we declare reference variable for sub class then we are able to access both abstract class members and sub class members.

**EX:**

```
1) abstract class A  
2) {  
3)     void m1()  
4)     {  
5)         System.out.println("m1-A");  
6)     }  
7)     abstract void m2();  
8)     abstract void m3();  
9) }  
10) class B extends A
```



```
11)
12) void m2()
13) {
14)     System.out.println("m2-B");
15) }
16) void m3()
17) {
18)     System.out.println("m3-B");
19) }
20) void m4()
21) {
22)     System.out.println("m4-B");
23) }
24)
25) class Test
26) {
27)     public static void main(String[] args)
28)     {
29)         //A a=new A();--> Error
30)         A a = new B();
31)         a.m1();
32)         a.m2();
33)         a.m3();
34)         //a.m4();--> Error
35)         B b = new B();
36)         b.m1();
37)         b.m2();
38)         b.m3();
39)         b.m4();
40)     }
41) }
```

## Q) What are the differences between Concrete Classes and Abstract Classes?

- 1) Classes are able to allow only concrete methods.  
Abstract classes are able to allow both concrete methods and abstract methods.
- 2) To declare concrete classes, only, 'class' keyword is sufficient.  
To declare abstract classes we need to use 'abstract' keyword along with class keyword.
- 3) For classes, we are able to create both reference variables and objects,  
For abstract classes, we are able to create only reference variables; we are unable to create objects.



- 4) Concrete classes will provide less sharability.  
Abstract classes will provide more sharability.

## 3) Interfaces:

- Interface is a java feature, it able to allow zero or more no of abstract methods only.
- For interfaces, we are able to declare only reference variables; we are unable to create objects.
- In case of interfaces, by default, all the variables as "public static final".
- In case of interfaces, by default, all the methods are "public and abstract".
- When compared with classes and abstract classes, interfaces will provide more sharability.

### Procedure to Use interfaces in Java applications:

- 1) Declare an interface with "interface" keyword.
- 2) Declare variables and methods in interface as per the requirement.
- 3) Declare an implementation class for interface by including "implements" keyword.
- 4) Provide implementation for abstract methods in implementation class which are declared in interface.
- 5) In main class, in main() method, create object for implementation class ,but, declare reference variable either for interface or for implementation class.
- 6) Access interface members.

**Note:** If declare reference variable for interface then we are able to access only interface members, we are unable to access implementation class own members. If we declare reference variable for implementation class then we are able to access both interface members and implementation class own members.

Example:

```
1) interface I
2) {
3)     int x=20;// public static final
4)     void m1();// public and abstract
5)     void m2();// public and abstract
6)     void m3();// public and abstract
7) }
8) class A implements I
9) {
10)    public void m1()
```



```
11) {
12)     System.out.println("m1-A");
13) }
14) public void m2()
15) {
16)     System.out.println("m2-A");
17) }
18) public void m3()
19) {
20)     System.out.println("m3-A");
21) }
22) public void m4()
23) {
24)     System.out.println("m4-A");
25) }
26}
27) class Test
28) {
29)     public static void main(String[] args)
30)     {
31)         //I i=new I();---> Error
32)         I i=new A();
33)         System.out.println(i.x);
34)         System.out.println(i.x);
35)         i.m1();
36)         i.m2();
37)         i.m3();
38)         //i.m4();----> Error
39)         A a=new A();
40)         System.out.println(a.x);
41)         System.out.println(A.x);
42)         a.m1();
43)         a.m2();
44)         a.m3();
45)         a.m4();
46)     }
47} }
```

## Q) What are the differences between Class, abstract Clacss and Interface?

- 1) Class is able to allow concrete methods only.  
Abstract class is able to allow both concrete methods and abstract methods  
Interface is able to allow abstract methods only.
- 2) To declare class, only, class keyword is sufficient.



To declare abstract class, we have to use "abstract" keyword along with class keyword.

To declare interface we have to use "interface" keyword explicitly.

- 3) For classes only, we are able to create both reference variables and objects.  
For abstract classes and interfaces, we are able to declare reference variables; we are unable to create objects.
- 4) In case of interfaces, by default, all variables are "public static final".  
In case of classes and abstract classes, no default cases for variables.
- 5) In case of interfaces, by default, all methods are "public and abstract".  
In case of classes and abstract classes, no default cases for methods.
- 6) Constructors are allowed in classes and abstract classes.  
Constructors are not allowed in interfaces.
- 7) Classes are able to provide less sharability.  
Abstract classes are able to provide middle level sharability.  
Interfaces are able to provide more sharability.

## Methods In Java:

Method is a set of instructions, it will represent a particular action in java applications.

### Syntax:

```
[Access_Modifiers] return_Type method_Name([param_List])[throws Exception_List]
{
    ----- instructions to represent a particular action-----
}
```

Java methods are able to allow the access modifiers like public, protected, <default> and private.

Java methods are able to allow the access modifiers like static, final, abstract, native, synchronized, strictfp.

Where the purpose of return\_Type is to specify which type of data the present method is returning. In java applications, all primitive data types, all user defined data types and 'void' are allowed for methods as return types.

Note: 'void' return type is representing "Nothing is returned" from methods.

Where "method\_name" is an identifier, it can be used to recognize the methods individually.



Where param\_List can be used to pass some input data to the methods in order to perform an action. In java applications, we are able to provide all primitive data types and all user defined data types as parameter types.

Where "throws" keyword can be used to bypass or delegate the generated exception from present method to caller method of the present method

To describe method information, there are two approaches.

- 1) Method Signature
- 2) Method Prototype

## **Q) What is the difference between Method Signature and Method Prototype?**

Method signature is the description of the method, it includes method name and parameter list.

EX: `forName(String Class_Name)`

Method prototype is the description of the method, it will include access modifiers list, return type, method name, parameters list, throws exception list.

EX: `public static Class forName(String class_Name) throws ClassNotFoundException`

There are two types of methods in Java w.r.t the Object state manipulations.

- 1) Mutator Methods
- 2) Accessor Methods

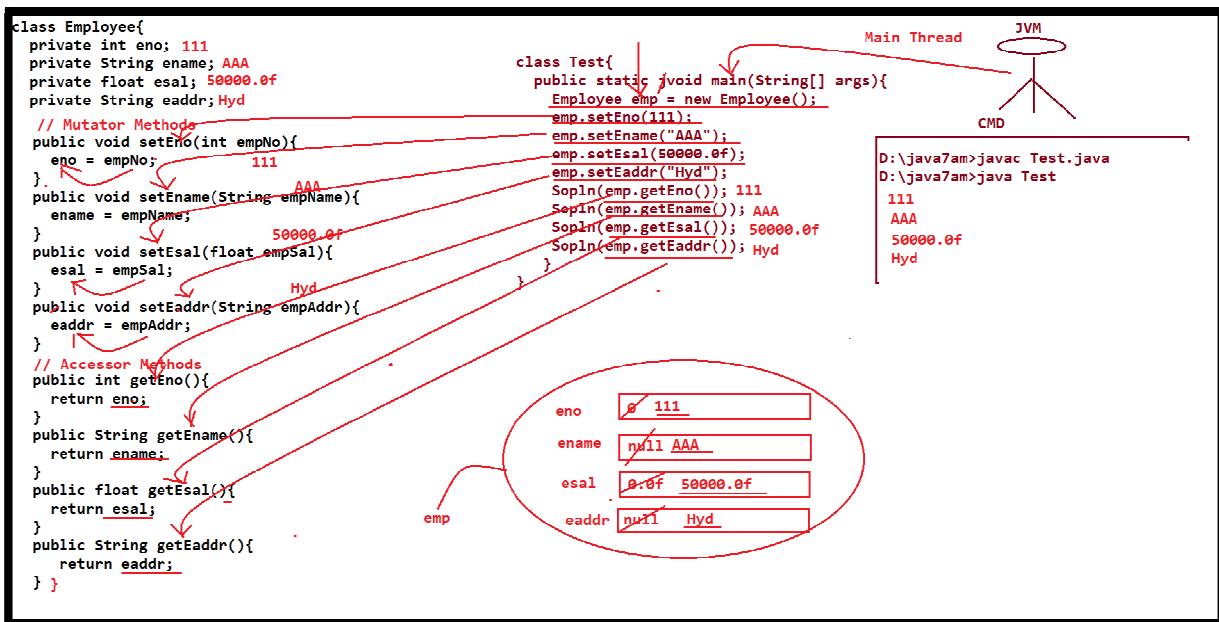
## **Q) What are the differences between Mutator methods and Accessor methods?**

Mutator methods are the java methods, which are used to set/modify data in Objects.

EX: All `setXXX(-)` methods in Java Bean classes are mutator methods.

Accessor methods are the java methods, which are used to get/access data from Objects.

EX: All `getXXX()` methods in java bean classes are Accessor methods



**Note:** Java bean is a normal java class, it will include properties and the respective setXXX(-) methods and getXXX() methods.

## Variable-Argument Method [Var-Arge method]

In general, in java applications, if we declare any method with 'n' no of parameters then we must access that method with the same 'n' no of parameter values , it is not possible to access that method by passing 'n+1' no of parameter values and 'n-1' no of parameter values.

**EX:**

```
1) class A  
2) {  
3)     void add(int i, int j)  
4)     {  
5)         System.out.println(i+j);  
6)     }  
7) }  
8) class Test  
9) {  
10)    public static void main(String[] args)  
11)    {  
12)        A a=new A();  
13)        a.add(10,20); // Valid.  
14)        //a.add();--> Invalid  
15)        //a.add(10);--> Invalid  
16)        //a.add(10,20,30);--> Invalid  
17)    }
```



18) }

As per the requirement, we want to access a method with variable no of parameter values [0 no of params or 1 no of params or 2 no of params,....]

To achieve this requirement we have to use "Var-Ag Method" provided by JDK5.0 version.

Var-Ag method is a java method including Var-Ag parameter.

Syntax for Var-Ag Parameter:

Data\_Type ... Var\_Name

EX: for var-Ag method:

```
void m1(int ... a)//int[] a={---- argumentvalues----}  
{  
----  
}
```

When we access Var-Ag method with 'n' no of parameter values then all these 'n' no of parameter values are stored in the form of Array which is generated from var-ARg parameter.

EX:

```
1) class A  
2) {  
3)   void add(int ... a)// int[] a={--- listof arguments---}  
4)   {  
5)     System.out.println("No Of Arguments :" +a.length);  
6)     int result=0;  
7)     System.out.print("Argument List :");  
8)     for(int i=0;i<a.length;i++)  
9)     {  
10)       System.out.print(a[i]+ " ");  
11)       result=result+a[i];  
12)     }  
13)     System.out.println();  
14)     System.out.println("Addition      :" +result);  
15)     System.out.println("-----");  
16) }
```



```
17)
18) class Test
19) {
20)     public static void main(String[] args)
21)     {
22)         A a=new A();
23)         a.add();
24)         a.add(10);
25)         a.add(10,20);
26)         a.add(10,20,30);
27)     }
28}
```

**Note:** In Var-Ag methods, normal parameters are allowed along with Var-Ag parameter but they must be provided before var-Ag parameter, not after var-Ag parameter, because, var-Ag parameter must be last parameter in var-Ag method.

**Note:** Due to the above reason, Var-Ag methods are able to allow at most one Var-Ag parameter, not allowing more than one Var-Ag parameters.

**EX:**

- 1) void m1(int ... i, float f){ } ----> Invalid
- 2) void m1(float f, int ... i){ } ----> valid
- 3) void m1(int ... i, float ... f){ } ---> Invalid

## Object Creation Process:

**Q) What is the Requirement to Create Objects in Java?**

- 1) To store entities data temporarily in java applications we need objects.
- 2) To Access the members of any particular class we need objects.

If we want to create Objects we have to use the following syntax.

Class\_Name ref\_Var = new Class\_Name([param\_List]);

**EX:**

```
class A{
---
```

```
A a = new A();
```

When JVM encounter the above instruction, JVM will perform the following actions.

- 1) Creating memory for the Object
- 2) Generating Identities for the object



### 3) Providing initializations inside the Object

#### 1) Creating Memory for the object:

When JVM encounter "new" keyword in Object creation statement then JVM will check to which class we are creating object on the basis of Constructor name then JVM will load the respective class byte code to the memory [Method Area]

After loading class byte code, JVM will identify minimal object memory size by recognizing all the instance variables and their data types

After getting memory size, JVM will send a request to Heap manager about to create an object with the specified minimal memory.

As per JVM requirement, Heap Manager will create the required block of memory at Heap Memory.

#### 2) Generating Identities for the Object:

After creating a block of memory [Object] for JVM requirements, Heap manager will assign an integer value as an identity for the object called as "HashCode".

After getting HashCode value, Heap Manager will send that hashCode value to JVM, where JVM will convert that hashCode value to its Hexa Decimal form called as "Reference Value".

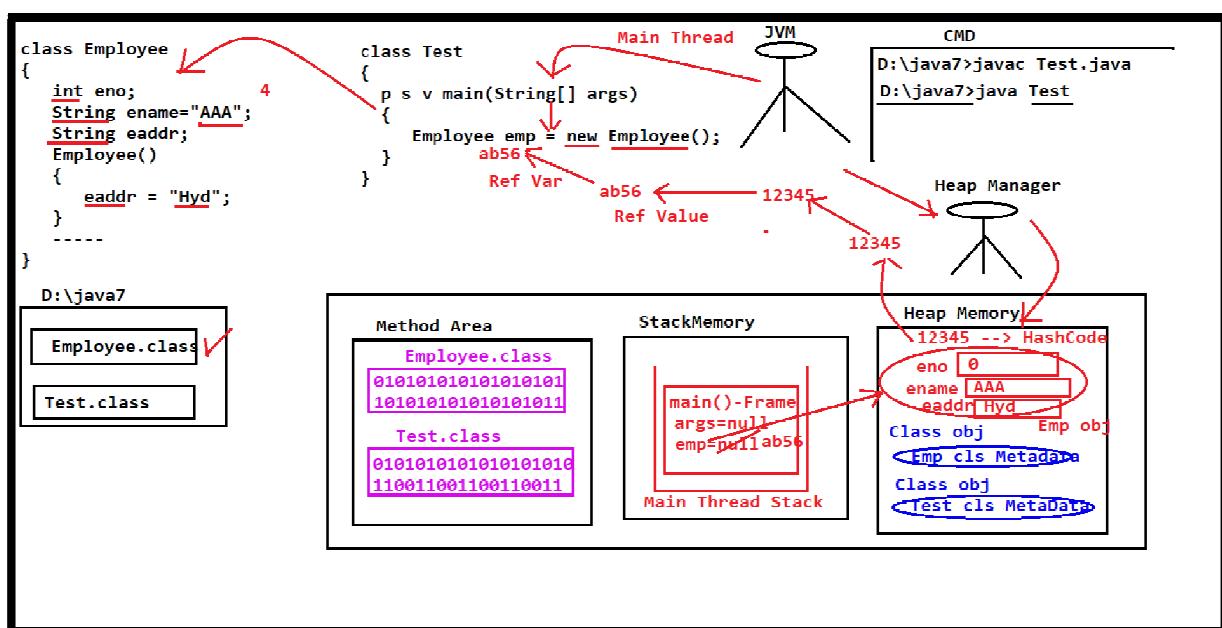
After getting Object reference value, JVM will assign that reference value to a variable called as "Reference Variable".

#### 3) Providing initializations inside the Object:

After creating object and its identities, JVM will allocate memory for all the instance variables inside the object on the basis of their data types.

After getting memory for the instance variables, JVM will provide initial values to the instance variables by searching initializations at class level declaration and at constructor.

If any instance variable is not having initialization at both constructor and at class level declaration then JVM will store default value on the basis of their data type as initial value inside the object.



**Note:** In java applications, when a class byte code is loaded to the memory, automatically, JVM will create `java.lang.Class` object at heap memory with the metadata of the loaded class.

**Note:** In java application, when we create a thread [Main Thread or User Thread], automatically, a stack will be created at stack memory and it able to store methods details in the form of Frames which are accessed by the respective thread.

To get hashCode value and reference value of an object we have to use the following two methods.

```
public native int hashCode()  
public String toString()
```

**Note:** Native method is a java method, declared in Java, but, implemented in non Java programming languages may be C, ASL, .....

**EX:**

```
1) class A  
2) {  
3) }  
4) class Test  
5) {  
6)     public static void main(String[] args)  
7)     {  
8)         A a=new A();  
9)         int hashCode=a.hashCode();
```



```
10) System.out.println("Object HashCode :" + hashCode);
11) String ref=a.toString();
12) System.out.println("Object Reference :" + ref);
13)
14}
```

**OUTPUT:**

Object Hashcode: 31168322  
Object Reference: A@adb9742

In Java applications, there is a common and default super class for each and every java class [predefined classes and user defined classes] that is "java.lang.Object" class, where java.lang.Object class contains the following 11 methods in order to share to all java classes.

- 1) hashCode()
- 2) toString()
- 3) getClass()
- 4) clone()
- 5) equals(Object obj)
- 6) finalize()
- 7) wait()
- 8) wait(long time)
- 9) wait(long time, int time)
- 10) notify()
- 11) notifyAll()

**Q) If we extend a class to some other class explicitly then the respective sub class is having two super classes [default super class [Object] and explicit super class], it represents multiple inheritance, how can we say multiple inheritance is not possible in JAVA?**

In java, java.lang.Object class is common and default super class for every class when that class is not extending from any other super class explicitly. If our class is extending some other class explicitly then Object class is not directly super class to the respective sub class, Object class is indirectly super class to the respective sub class through the explicit super class , that is, Object class is not super class to the respective sub class through Multiple Inheritance, that is through Multi Level Inheritance.

**EX:**

```
class A{
---}
class B extends A{
```



---

}

**Note:** Here Object class is super class to A, A is super class to B

Object <--- A <--- B

In java applications, when we pass a particular class object reference variable as parameter to System.out.println() method then JVM will access toString() over the provided reference variable internally.

**EX:**

```
1) class A
2) {
3) }
4) class Test
5) {
6)     public static void main(String[] args)
7)     {
8)         A a=new A();
9)
10)        String ref=a.toString();
11)        System.out.println(ref);
12)
13)        System.out.println(a.toString());
14)
15)        System.out.println(a);//System.out.println(a.toString());
16)    }
17} }
```

**OUTPUT:**

A@1db9742

A@1db9742

A@1db9742

In the above program, when we access toString() method internally or externally , first, JVM has to execute toString(), to execute toString() method JVM will search for toString() method in the respective class whose reference variable we passed as parameter to System.out.println() method, if the required toString() method is not existed in the respective class then JVM will search for toString() method in its super class, here if super class is not existed then JVM will search for toString() method in the common and default super class Object class. In Object class, toString() method was implemented insuch a way to return a string contains "Class\_Name@ref\_Val" .



As per the requirement, if we want to display our own data instead of Object reference value when we pass reference variable as parameter to `System.out.println()` method then we have to provide our own `toString()` method in the respective class.

EX:

```
1) class Employee
2) {
3)     String eid="E-111";
4)     String ename="Durga";
5)     float esal=50000.0f;
6)     String eaddr="Hyd";
7)     String eemail="durga@durgasoft.com";
8)     String emobile="91-9988776655";
9)
10)    public String toString()
11)    {
12)        System.out.println("Employee Details");
13)        System.out.println("-----");
14)        System.out.println("Employee Id :"+eid);
15)        System.out.println("Employee Name :"+ename);
16)        System.out.println("Employee Salary :"+esal);
17)        System.out.println("Employee Address :"+eaddr);
18)        System.out.println("Employee Email :"+eemail);
19)        System.out.println("Employee Mobile :"+emobile);
20)        return "";
21)    }
22) }
23) class Test
24) {
25)     public static void main(String[] args)
26)     {
27)         Employee emp=new Employee();
28)         System.out.println(emp);
29)     }
30) }
```

Note: In Java, some predefined classes like `String`, `StringBuffer`, `Exception`, `Thread`, all wrapper classes, all Collection classes are not depending on `Object` class `toString()` method, they are having their own `toString()` method inorder to display their own data.

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
```



```
4) {
5)     String str=new String("abc");
6)     System.out.println(str);
7)
8)     Exception e=new Exception("My Own Exception");
9)     System.out.println(e);
10)
11)    Thread t=new Thread();
12)    System.out.println(t);
13)
14)    Integer in=new Integer(10);
15)    System.out.println(in);
16)
17)    java.util.ArrayList al=new java.util.ArrayList();
18)    al.add("AAA");
19)    al.add("BBB");
20)    al.add("CCC");
21)    al.add("DDD");
22)    System.out.println(al);
23) }
24} }
```

## OUTPUT:

abc  
java.lang.Exception: My Own Exception  
Thread[Thread-0,5,main]  
10  
[AAA,BBB,CCC,DDD]

In JAVA, there are two types of Objects.

- 1) 1.Immutable Objects
- 2) 2.Mutable Objects

Q) What is the difference between *Immutable object* and *mutable object*?

*OR*

Q) What is the difference between *String* and *StringBuffer*?

Immutable Objects are Java objects; they will not allow modifications on their content. If we are trying to perform operations over immutable objects content then data is allowed for operations, but, the resultant data is not stored back in original object, the modified data will be stored by creating new Object.



EX: All String class objects are immutable objects.

All Wrapper class objects are immutable Objects.

Mutable Objects are java objects, they will allow modifications on their content directly.

EX: By default, all JAVA objects are mutable objects.

EX: StringBuffer

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         String str1=new String("Durga ");
6)         String str2=str1.concat("Software ");
7)         String str3=str2.concat("Solutions");
8)         System.out.println(str1);
9)         System.out.println(str2);
10)        System.out.println(str3);
11)        System.out.println();
12)        StringBuffer sb1=new StringBuffer("Durga ");
13)        StringBuffer sb2=sb1.append("Software ");
14)        StringBuffer sb3=sb2.append("Solutions");
15)        System.out.println(sb1);
16)        System.out.println(sb2);
17)        System.out.println(sb3);
18)    }
19} }
```

OUTPUT:

Durga

Durga Software

Durga Software Solutions

Durga Software Solutions

Durga Software Solutions

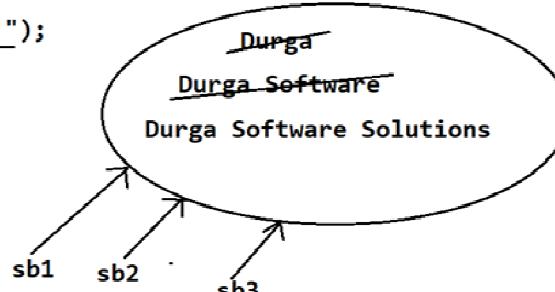
Durga Software Solutions



```
String str1 = new String("Durga ");      str1 → Durga
String str2 = str1.concat("Software ");   str2 → Durga Software
String str3 = str2.concat("Solutions");   str3 → Durga Software Solutions
```

---

```
StringBuffer sb1 = new StringBuffer("Durga ");
StringBuffer sb2 = sb1.append("Software ");
StringBuffer sb3 = sb2.append("Solutions");
str1 == str2, str2 == str3, str3 == str1
      False       false      False
sb1 == sb2, sb2 == sb3, sb3 == sb1
      True       True      True
```

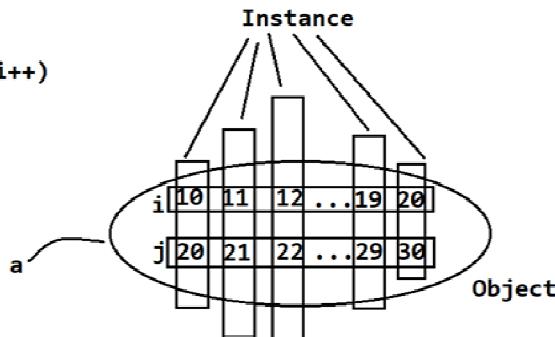


## Q) What is the difference between Object and instance?

Object is a memory unit to store data.

Instance is a copy or layer of values existed in an object at a particular point of time.

```
class A
{
    int i = 10;
    int j = 20;
}
A a = new A();
for(int i = 1; i <= 10; i++)
{
    a.i = a.i + 1;
    a.j = a.j + 1;
}
.
.
.
No of Objects : 1
No of Instances : 11
```



## Constructors:

- 1) Constructor is a java feature; it can be used to create Object.
- 2) The role of the constructors in Object creation is to provide initial values inside the object.
- 3) In java applications, Constructors are executed exactly at the time of creating objects, not before creating objects and not after creating objects.
- 4) The main utilization of constructors is to provide initializations for class level variables mainly instance variable.
- 5) Constructors must have same name of the respective class.
- 6) Constructors are not having return types.
- 7) Constructors are not allowing the access modifiers like static, final,...
- 8) Constructors are able to allow the access modifiers like public, protected, <default> and private.



- 
- 9) Constructors are allowing 'throws' keyword in its syntax to bypass exceptions from present constructor to the caller.

**Syntax:** [Access\_Modifier] Class\_Name([Param\_List])[throws Exception\_List]

```
{  
----  
}
```

### Note 1:

If we provide constructor name other than class name then compiler will rise an error like "Invalid Method declaration, return type required", because, compiler has treated the provided constructor as normal java method without the return type, but, for methods return is mandatory.

### EX:

```
1) class A  
2) {  
3)   B()  
4) {  
5)   System.out.println("A-Con");  
6) }  
7)  
8) class Test  
9) {  
10) public static void main(String[] args)  
11) {  
12)   A a=new A();  
13) }  
14)}
```

Status: Compilation Error

### Note 2:

If we provide return type to the constructors then Compiler will not rise any error and JVM will not rise any exception and JVM will not provide any output, because, the provided constructor is converted as normal java method. In this context, if we access the provided constructor as normal java method then it will be executed as like normal java method.

### EX:

```
1) class A  
2) {  
3)   void A()
```



```
4) {
5)     System.out.println("A-Con");
6) }
7) }
8) class Test
9) {
10)    public static void main(String[] args)
11)    {
12)        A a=new A();
13)        a.A();
14)    }
15) }
```

Status: No Compilation Error

OUTPUT: A-Con

#### Note 3:

If we provide the access modifiers like static, final,... to the constructors then Compiler will rise an error like "modifier xxx not allowed here".

EX:

```
1) class A
2) {
3)     static A()
4) {
5)     System.out.println("A-Con");
6) }
7) }
8) class Test
9) {
10)    public static void main(String[] args)
11)    {
12)        A a=new A();
13)
14)    }
15) }
```

Status: Compilation Error.

#### Note 4:

If we declare constructor as "private" then that constructor is available upto the respective class only, not available to out side of the respective class.

In this context, If we want to create object for the respective class then it is possible inside the same class only.



EX:

```
1) class A
2) {
3)     private A()
4) {
5)     System.out.println("A-Con");
6) }
7)
8) class Test
9) {
10)    public static void main(String[] args)
11)    {
12)        A a=new A();
13)    }
14)}
```

Status: Compilation Error

There are two types constructors in java

- 1) Default Constructors
- 2) User Defined Constructors

## **1) Default Constructors:**

If we have not provided any constructor explicitly in java class then compiler will provide a 0-arg constructor automatically, here the compiler provided 0-arg constructor is called as "Default Constructor".

If we provide any constructor explicitly then compiler will not provide any default constructor.

```
D:\javaapps\ Test.java
public class Test {
```

On Command Prompt:

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

```
D:\javaapps>javap Test
```

```
Compiled from Test.java
```

```
public class Test {
    public Test() { -----> Default constructor
    }
}
```

Note: Default Constructors are having the same scope of the respective class.



## 2) User Defined Constructors:

- These constructors are provided by the developers as per their application requirements.
- If we provide any constructor explicitly without parameters then that constructor is called as 0-arg constructor.
- If we provide any constructor with at least one parameter then that constructor is called as parameterized Constructor.

**Note:** By default, all the default constructors are 0-arg constructors, but, all 0-arg constructors are not default constructors, some 0-arg constructors are provided by the compiler are called as Default Constructors and some other 0-arg constructors are provided by the users they are called as User defined constructors.

**EX:**

```
1) class Employee
2) {
3)     String eid;
4)     String ename;
5)     float esal;
6)     String eaddr;
7)
8)     Employee()
9)     {
10)         eid="E-111";
11)         ename="Durga";
12)         esal=50000.0f;
13)         eaddr="Hyd";
14)     }
15)
16)     public void getEmpDetails()
17)     {
18)         System.out.println("Employee Details");
19)         System.out.println("-----");
20)         System.out.println("Employee Id    :" +eid);
21)         System.out.println("Employee Name   :" +ename);
22)         System.out.println("Employee Salary  :" +esal);
23)         System.out.println("Employee Address :" +eaddr);
24)     }
25) }
26) class Test
27) {
28)     public static void main(String[] args)
29) {
```



```
30) Employee emp=new Employee();
31) emp.getEmpDetails();
32)
33}
```

In the above program, if we create more than one object for the Employee class by executing 0-arg constructor then same data will be stored in all multiple objects of the Employee.

As per the requirement, if we want to create multiple Employee objects with different data then we have to provide data to the Objects explicitly, for this we have to use parameterized constructor.

EX:

```
1) class Employee
2) {
3)     String eid;
4)     String ename;
5)     float esal;
6)     String eaddr;
7)
8)     Employee(String emp_Id, String emp_Name, float emp_Sal, String emp_Addr)
9)     {
10)         eid=emp_Id;
11)         ename=emp_Name;
12)         esal=emp_Sal;
13)         eaddr=emp_Addr;
14)     }
15)
16)     public void getEmpDetails()
17)     {
18)         System.out.println("Employee Details");
19)         System.out.println("-----");
20)         System.out.println("Employee Id    :" +eid);
21)         System.out.println("Employee Name   :" +ename);
22)         System.out.println("Employee Salary  :" +esal);
23)         System.out.println("Employee Address :" +eaddr);
24)     }
25)
26) class Test
27) {
28)     public static void main(String[] args)
29)     {
30)         Employee emp1=new Employee("E-111", "Durga", 50000.0f, "Hyd");
31)         emp1.getEmpDetails();
```



```
32) System.out.println();
33) Employee emp2=new Employee("E-222", "Anil", 60000.0f, "Hyd");
34) emp2.getEmpDetails();
35) System.out.println();
36) Employee emp3=new Employee("E-333", "Rahul", 80000.0f, "Hyd");
37) emp3.getEmpDetails();
38)
39}
```

## Constructor Overloading:

If we declare more than one constructor with the same name and with the different parameter list then it is called as "Constructor Overloading".

EX:

```
1) class A
2) {
3)     int i,j,k;
4)     A()
5)     {
6)     }
7)     A(int i1)
8)     {
9)         i=i1;
10)    }
11)    A(int i1, int j1)
12)    {
13)        i=i1;
14)        j=j1;
15)    }
16)    A(int i1, int j1, int k1)
17)    {
18)        i=i1;
19)        j=j1;
20)        k=k1;
21)    }
22)    void add()
23)    {
24)        System.out.println("Addition :" +(i+j+k));
25)    }
26}
27) class Test
28) {
29)     public static void main(String[] args)
30)     {
```



```
31) A a1=new A();  
32) a1.add();  
33) A a2=new A(10);  
34) a2.add();  
35) A a3=new A(10,20);  
36) a3.add();  
37) A a4=new A(10,20,30);  
38) a4.add();  
39) }  
40} }
```

**Note:** If we declare more than one method with the same name and with different parameter list then it is called as "Method Overloading"

## **Instance Context/Instance Flow of Execution:**

In Java, for every class loading a separate context will be created called as Static Context and for every Object a separate context will be created called as Instance context.

In Java, instance context is represented in the form of the following elements.

- 1) Instance Variables
- 2) Instance Methods
- 3) Instance Blocks

### **1) Instance Variables:**

- \* Instance Variable is a normal Java variable, whose values will be varied from one instance to another instance of an object.
- \* Instance Variable is a variable, which will be recognized and initialized just before executing the respective class constructor.
- \* In Java applications, instance variables must be declared at class level and non-static, instance variables never be declared as local variables and static variables.
- \* In Java applications, instance variables data will be stored in Object memory that is in "Heap Memory".

### **2) Instance Methods:**



- \* Instance Method is a normal Java method, it is a set of instructions, it will represent an action of an entity.
- \* In Java applications, instance methods will be executed when we access that method. In Java applications, all the methods won't be executed without the method call.

EX:

```
1) class A {  
2)     int i=m10;  
3)     A0 {  
4)         System.out.println("A-Con");  
5)     }  
6)     int m10 {  
7)         System.out.println("M1-A");  
8)         return 10;  
9)     }  
10}   
11) class Test {  
12)     public static void main(String args[]) {  
13)         A a = new A();  
14)     }  
15} 
```

OUTPUT:

M1-A  
A-con

EX:

```
1) class A {  
2)     int j = m10;  
3)     int m20 {  
4)         System.out.println("m2-A");  
5)         return 10;  
6)     }  
7)     A0 {  
8)         System.out.println("A-con");  
9)     }  
10)    int m10 {  
11)        System.out.println("m1-A");  
12)        return 20;  
13)    }  
14)    int i = m20; 
```



```
15)
16) class Test {
17)     public static void main(String args[]) {
18)         A a = new A();
19)
20} }
```

OUTPUT: m1-A  
m2-A  
A-con

### 3) Instance Block:

- Instance Block is a set of instructions which will be recognized and executed just before executing the respective class constructors.
- Instance Block as are having the same power of constructors, it can be used as like constructors.

#### Syntax:

```
{  
    ---instructions----  
}
```

#### EX:

```
1) class A {  
2)     A0 {  
3)         System.out.println("A-CON");  
4)     }  
5)     {  
6)         System.out.println("IB-A");  
7)     }  
8) }  
9) class Test {  
10)    public static void main(String args[]) {  
11)        A a = new A();  
12)    }  
13} }
```

#### OUTPUT:

IB-A  
A-CON

#### EX:

```
1) class A {
```



```
2) class A0 {
3)     System.out.println("A-CON");
4) }
5) {
6)     System.out.println("IB-A");
7) }
8) int m10 {
9)     System.out.println("m1-A");
10)    return 10;
11) }
12) int i = m10;
13}
14) class Test {
15)     public static void main(String args[]) {
16)         A a = new A0;
17)     }
18}
```

OUTPUT: IB-A  
m1-A  
A-CON

EX:

```
1) class A {
2)     int m10 {
3)         System.out.println("m1-A");
4)         return 10;
5)     }
6) {
7)     System.out.println("IB-A");
8) }
9) int i=m20;
10) A0 {
11)     System.out.println("A-con");
12) }
13) int i=m10;
14) {
15)     System.out.println("IB1-A");
16) }
17) int m20;
18) {
19)     System.out.println("m2-A");
20)     return 20;
21) }
22}
```



```
23) class Test {  
24)     public static void main(String args[]) {  
25)         A a1=new A();  
26)         A a2=new A();  
27)     }  
28) }
```

## OUTPUT:

IB-A  
m2-A  
m1-A  
IB1-A  
A-con  
IB-A  
m2-A  
m1-A  
IB1-A  
A-con

## 'this' Keyword:

'this' is a Java keyword, it can be used to represent current class object.

In Java applications, we are able to utilize 'this' keyword in the following 4 ways.

- 1) To refer current class variable
- 2) To refer current class methods
- 3) To refer current class constructors
- 4) To refer current class object

### **1) To Refer Current Class Variables:**

If we want to refer current class variables by using 'this' keyword then we have to use the following syntax.

`this.var_Name`

**NOTE:** In Java applications, if we provide same set of variables at local and at class level and if we access that variables then JVM will give first priority for local variables, if local variables are not available then JVM will search for that variables at class level, even at class level also if that variables are not available then JVM will search at super class level. At all the above locations, if the specified variables are not available then compiler will rise an error.

**NOTE:** In Java applications, if we have same set of variables at local and at class level then to access class level variables over local variables we have to use 'this' keyword.



EX:

```
1) class A {  
2)     int i=10;  
3)     int j=20;  
4)     A(int i,int j) {  
5)         System.out.println(i+" "+j);  
6)         System.out.println(this.i+" "+this.j);  
7)     }  
8) }  
9) class Test {  
10)    public static void main(String args[]) {  
11)        A a = new A(30,40);  
12)    }  
13) }
```

OUTPUT:

30 40  
10 20

**NOTE:** In general, in enterprise applications, we are able to write no. of Java bean classes as per application requirement. In Java bean classes, we are able to provide variables and their setter methods and getter methods. In Java bean classes, in setter methods, we have to declare parameter variable with the same name of the respective class level variable, here we have to transfer data from local variable to class level variable, for this, we have to assign local variable to class level variable. In this context, to refer class level variable over local variable separately we have to use 'this' keyword. In getter methods, always, we have to return class level variables only, so that, it is not required to use 'this' keyword.

EX:

```
1) class User {  
2)     private String uname;  
3)     private String upwd;  
4)     public void setUsername(String uname) {  
5)         this.uname = uname;  
6)     }  
7)     public void setUpwd(String upwd) {  
8)         this.upwd = upwd;  
9)     }  
10)    public String getUsername() {  
11)        return uname;  
12)    }  
13)    public String getUpwd() {  
14)        }  
15) }
```



```
16) class Test {  
17)     public static void main(String[] args) {  
18)         User u = new User();  
19)         u.setUname("abc");  
20)         u.setUpwd("abc123");  
21)         System.out.println("User Login Details");  
22)         System.out.println("-----");  
23)         System.out.println("User Name :"+u.getUname());  
24)         System.out.println("Password :"+getEaddr());  
25)     }  
26} }
```

EX:

```
1) class Employee  
2) {  
3)     private String eid;  
4)     private String ename;  
5)     private float esal;  
6)     private String eaddr;  
7)  
8)     public void setEid(String eid)  
9)     {  
10)        this.eid=eid;  
11)    }  
12)    public void setEname(String ename)  
13)    {  
14)        this.ename=ename;  
15)    }  
16)    public void setEsal(float esal)  
17)    {  
18)        this.esal=esal;  
19)    }  
20)    public void setEaddr(String eaddr)  
21)    {  
22)        this.eaddr=eaddr;  
23)    }  
24)  
25)    public String getEid()  
26)    {  
27)        return eid;  
28)    }  
29)    public String getEname()  
30)    {  
31)        return ename;  
32)    }
```



```
33) public float getEsal()
34) {
35)     return esal;
36) }
37) public String getEaddr()
38) {
39)     return eaddr;
40) }
41) }
42) class Test
43) {
44)     public static void main(String[] args)
45)     {
46)         Employee emp=new Employee();
47)         emp.setEid("E-111");
48)         emp.setEname("AAA");
49)         emp.setEsal(5000.0f);
50)         emp.setEaddr("Hyd");
51)
52)         System.out.println("Employee Details");
53)         System.out.println("-----");
54)         System.out.println("Employee Id    :" +emp.getEid());
55)         System.out.println("Employee Name   :" +emp.getEname());
56)         System.out.println("Employee Salary  :" +emp.getEsal());
57)         System.out.println("Employee Address :" +emp.getEaddr());
58)     }
59) }
```

## 2) To Refer Current Class Method:

If we want to refer current class method by using 'this' keyword then we have to use the following syntax.

`this.method_Name([param_List]);`

**Note:** To access current class methods, it is not required to use any reference variable and any keyword including 'this', directly we can access.

**EX:**

```
1) class A {
2)     void m1()
3)     {
3)         System.out.println("m1-A");
4)     }
5)     void m2()
6)     {
6)         System.out.println("m2-A");
6)     }
6) }
```



```
7)     m10;
8)     this.m10;
9)   }
10) }
11) class Test {
12)   public static void main(String args[]) {
13)     A a = new A();
14)     a.m20();
15)   }
16) }
```

**OUTPUT:**

m2-A  
m1-A  
m1-A

### **3) To Refer Current Class Constructors:**

If we want to refer current class constructor by using 'this' keyword then we have to use the following syntax.

`this([param_List]);`

**EX:**

```
1) class A {
2)   A0 {
3)     this(10);
4)     System.out.println("A-0-arg-con");
5)   }
6)   A(int i) {
7)     this(22.22f);
8)     System.out.println("A-int-param-con");
9)   }
10)  A(float f) {
11)    this(33.3333);
12)    System.out.println("A-float-param-con");
13)  }
14)  A(double d) {
15)    System.out.println("A-double-param-con");
16)  }
17) }
18) class Test {
19)   public static void main(String args[]) {
20)     A a = new A0();
21)   }
```



| 22 ) }

## OUTPUT:

A-double-param-con  
A-float-param-con  
A-int-param-con  
A-0-arg-con

**NOTE:** In the above program we have provided more than one constructor with the same name and with the different parameter list, this process is called as "Constructor Overloading".

In the above program, we have called all the current class constructors by using 'this' keyword in chain fashion, this process is called as "Constructor Chaining".

**NOTE:** If we want to access current class constructor by using 'this' keyword then the respective 'this' statement must be provided as first statement, if we have not provided 'this' statement as first statement then compiler will rise an error like 'call to this must be first statement in constructor'.

## EX:

```
1) class A {  
2)     A0 {  
3)         System.out.println("A-0-arg-con");  
4)         this(10);  
5)     }  
6)     A(int i) {  
7)         System.out.println("A-int-param-con");  
8)         this(22.22f);  
9)     }  
10)    A(float f) {  
11)        System.out.println("A-float-param-con");  
12)        this(33.3333);  
13)    }  
14)    A(double d) {  
15)        System.out.println("A-double-param-con");  
16)    }  
17) }  
18) class Test {  
19)     public static void main(String args[]) {  
20)         A a = new A();  
21)     }  
22) }
```

Status: Compilation Error



**NOTE:** If we want to refer current class constructor by using 'this' keyword then the respective 'this' statement must be provided in the current class another constructor only, not in normal Java methods. If we violate this condition then compiler will rise an error like 'call to this must be first statement in constructors'.

**EX:**

```
1) class A
2) {
3)     A()
4)     {
5)         System.out.println("A-0-arg-con");
6)     }
7)     A(int i)
8)     {
9)         System.out.println("A-int-param-con");
10)    }
11)    void m1()
12)    {
13)        this(10);
14)        System.out.println("m1-A");
15)    }
16)
17) class Test
18) {
19)     public static void main(String args[])
20)     {
21)         A a=new A();
22)         a.m1();
23)     }
24)}
```

Status: Compilation Error.

**Q) Is it possible to refer more than one current class constructors by using 'this' keyword from a single current class constructor?**

No, It is not possible to refer more than one current class constructors by using 'this' keyword, because, in constructors, 'this' statement must be first statement while referring current class constructors. If we provide more than one time this(...) statement then only one this() statement is first statement among multiple this statements, in this case, compiler will rise an error like 'call to this must be first statement'.



EX:

```
1) class A
2) {
3)     A()
4) {
5)     this(10);
6)     this(22.22f);
7)     System.out.println("A-0-arg-con");
8)
9)     A(int i)
10)    {
11)        System.out.println("A-int-param-con");
12)    }
13)     A(float f)
14)    {
15)        System.out.println("A-float-param-con");
16)    }
17}
18 class Test
19{
20)    public static void main(String args[])
21)    {
22)        A a=new A();
23)    }
24}
```

Status: Compilation Error

## 4) To Return Current Class Object:

To return current class object by using 'this' keyword thn we have to use the following syntax.

return this;

EX:

```
1) class A {
2)     A getRef1() {
3)         A a = new A();
4)         return a;
5)     }
6)     A getRef2() {
7)         return this;
8)     }
```



```
9) }
10) class Test {
11)     public static void main(String args[]) {
12)         A a = new A(); // [a=abc123]
13)         System.out.println(a);
14)         System.out.println();
15)         System.out.println(a.getRef1()); // [abc123.getRef10]
16)         System.out.println(a.getRef1()); // [abc123.getRef10]
17)         System.out.println(a.getRef1()); // [abc123.getRef10]
18)         System.out.println();
19)         System.out.println(a.getRef2()); // [abc123.getRef20]
20)         System.out.println(a.getRef2()); // [abc123.getRef20]
21)         System.out.println(a.getRef2()); // [abc123.getRef20]
22)     }
23} }
```

#### OUTPUT:

A@5e3a78ad

A@50c8d62f

A@3165d118

A@138297fe

A@5e3a78ad

A@5e3a78ad

A@5e3a78ad

In the above program, for every call of getRef1() method JVM will encounter "new" keyword, JVM will create new Object for class A every time and JVM will return new object reference value every time. This approach will increase no. of objects in Java application, it will not provide Objects reusability, it will provide object duplication, it is not suggestible in application development.

In the above program, for every call of getRef2() method JVM will encounter "return this" statement, JVM will not create new Object for class A every time, JVM will return the same reference value on which we have called getRef2() method. This approach will increase Objects reusability.

## 'static' keyword:

'static' is a Java keyword, it will improve sharability in Java applications.

In Java applications, static keyword will be utilized in the following four ways.



- 
- 1) Static variables
  - 2) Static methods
  - 3) Static blocks
  - 4) Static import

## 1) Static variables:

- Static variables are normal Java variables, which will be recognized and executed exactly at the time of loading the respective class byte code to the memory.
- Static variables are normal java variables, they will share their last modified values to the future objects and to the past objects of the respective class.
- In Java applications, static variables will be accessed either by using the respective class reference variable or by using the respective class name directly.

**NOTE:** To access static variables we can use the respective class reference variable which may or may not have reference value. To access static variables it is sufficient to take a reference variable with null value.

**NOTE:** If we access any non-static variable by using a reference variable with null value then JVM will rise an exception like "java.lang.NullPointerException". If we access static variable by using a reference variable contains null value then JVM will not rise any exception.

In Java applications, static variables must be declared as class level variables only, they never be declared as local variables.

In Java applications, static variable values will be stored in method area, not in Stack memory and not in Heap Memory.

In Java applications, to access current class static variables we can use "this" keyword.

**EX:**

```
1) class A {  
2)     static int i=10;  
3)     int j = 10;  
4)     void m10 {  
5)         //static int k=30;--->error  
6)         System.out.println("m1-A");  
7)         System.out.println(this.i);  
8)     }  
9) }  
10) class Test {  
11)     public static void main(String args[]) {  
12)         A a = new A();  
13)         System.out.println(a.i);  
14)         System.out.println(A.i);  
15)         a.m10;
```



```
16)     A a1 = null;
17)     //System.out.println(a1.j);--->NullPointerException
18)     System.out.println(a1.i);
19) }
20} }
```

**OUTPUT:**

```
10
10
m1-A
10
10
```

**EX:**

```
1) class A {
2)     static int i=10;
3)     int j=10;
4) }
5) class Test {
6)     public static void main(String args[]) {
7)         A a1 = new A();
8)         System.out.println(a1.i+" "+a1.j);
9)         a1.i=a1.i+1;
10)        a1.j=a1.j+1;
11)        System.out.println(a1.i+" "+a1.j);
12)        A a2 = new A();
13)        System.out.println(a2.i+" "+a2.j);
14)        a2.i=a2.i+1;
15)        a2.j=a2.j+1;
16)        System.out.println(a1.i+" "+a1.j);
17)        System.out.println(a2.i+" "+a2.j);
18)        A a3 = new A();
19)        System.out.println(a3.i+" "+a3.j);
20)        a3.i=a3.i+1;
21)        a3.j = a3.j+1;
22)        System.out.println(a1.i+" "+a1.j);
23)        System.out.println(a2.i+" "+a2.j);
24)        System.out.println(a3.i+" "+a3.j);
25)    }
26} }
```

**OUTPUT:**

```
10 10
11 11
11 10
```



```
12 11  
12 11  
12 11  
12 10  
13 11  
13 11  
13 11
```

**NOTE:** In Java applications, Instance variable is specific to each and every object that is a separate copy of instance variables will be maintained by each and every object but static variable is common to every object that is the same copy of static variable will be maintained by all the objects of the respective class.

**Note:** In Java, for a particular, class Byte-Code will be loaded only one time but we can create any number of objects.

## 2) Static Methods:

- Static method is a normal java method, it will be recognized and executed the time when we access that method.
- Static methods will be accessed either by using reference variable or by using the respective class name directly.

**Note:** In the case of accessing static methods by using reference variables, reference variable may or may not have object reference value, it is possible to access static methods with the reference variables having 'null' value. If we access non-static method by using a reference variable contains null value then JVM will rise an exception like "java.lang.NullPointerException".  
Static methods will allow only static members of the current class, static methods will not allow non-static members of the current class directly.

**Note:** If we want to access non-static members of the current class in static methods then we have to create an object for the current class and we have to use the generated reference variable.

Static methods are not allowing 'this' keyword in its body but to access current class static methods we are able to use 'this' keyword.

**EX:**

```
1) class A {  
2)     int i = 10;  
3)     static int j = 20;  
4)     static void m1() {  
5)         System.out.println("m1-A");  
6)         System.out.println(j);
```



```
7)      //System.out.println(i);---->error
8)      //System.out.println(this.j);----->error
9)      A a = new A();
10)     System.out.println(a.i);
11) }
12) void m2() {
13)     System.out.println("m2-A");
14)     this.m1();
15) }
16)
17) class Test {
18)     public static void main(String[] args) {
19)         A a = new A();
20)         a.m1();
21)         a = null;
22)         a.m1();
23)         A.m1();
24)     }
25} }
```

## OUTPUT:

m1-A  
20  
10  
m1-A  
20  
10  
m1-A  
20  
10

**Q) Is it possible to print a line of text on command prompt without using main() method?**

Yes, it is possible to display a line of text on command prompt without using main() method, but, by using static variable and static method combination.

## EX:

```
1) class Test {
2)     static int i = m1();
3)     static int m1() {
4)         System.out.println("Welcome to durga software soultions");
5)         System.exit(0);//to terminate the application
6)         return 10;
7)     }
```



8) }

**OUTPUT:** Welcome to durga software soultions

If we provide 'Test' class name along with 'java' command on command prompt then JVM will take main class from command prompt, JVM will search for its .class file, if it is available then JVM will load main class bytecode to the memory that is Test class bytecode. At the time of loading Test class bytecode to the memory, JVM will recognize and initialize static variable, as part of initialization, JVM will execute static method. By the execution of static method, JVM will display the required message on command prompt, when JVM encounter System.exit(0) statement then JVM will terminate the application.

**NOTE:** The above question and answer are valid up to JAVA6 version, it is invalid from JAVA7 version, because, In JAVA6 version JVM will load main class bytecode to the memory irrespective of main() method availability. In JAVA7, first, JVM will check whether main() method is existed or not in main class, if main() method is available then only JVM will load main class bytecode to the memory, if main() method is not available in main class then JVM will not load main class bytecode to the memory and JVM will provide the following error message.

Error:Main Method not found in class Test,please define the main method as:

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

### 3) Static Block:

- Static Block is a set of instructions, which will be recognized and executed at the time of loading the respective class bytecode to the memory.
- Static blocks are able to allow static members of the current class directly, Static blocks are not allowing non-static members of the current class directly.

**Note:** If we want to access non-static members of the current class in static block then we must create object for the respective class and we have to use the generated reference variable.

Static blocks are not allowing 'this' keyword in its body.

**EX:**

```
1) class A {  
2)     int i=10;  
3)     static int j=20;
```



```
4) static {
5)     System.out.println("SB-A");
6)     System.out.println(i); //----->Error
7)     A a = new A();
8)     System.out.println(a.i);
9)     System.out.println(j);
10)    System.out.println(this.j);----->Error
11)
12}
13) class Test {
14)     public static void main(String[] args) {
15)         A a = new A();
16)     }
17}
```

**OUTPUT:**

SB-A  
10  
20

**Q) Is it Possible to print a line of text on command prompt with out using main() method, static variable and static method?**

Yes, it is possible to display a line of text on command prompt without using main() method, static variable, static method but by using static block.

**EX:**

```
1) class Test {
2)     static {
3)         System.out.println("Welcome to DurgaSoftware Soultions");
4)         System.exit(0); //To terminate the programme
5)     }
6}
```

**OUTPUT:**

Welcome to DurgaSoftware Soultions

If we provide main class name 'Test' along with 'java' command on command prompt then JVM will take main class name i.e Test and JVM will search for its .class file. If Test.class file is identified then JVM will load its bytecode to the memory, at the time of loading main class bytecode to the memory static block will be executed, with this, the required message will be displayed on command prompt. When JVM encounter System.exit(0) then JVM will terminate the program execution.



**NOTE:** The above question and answer are valid up to JAVA6 version, they are invalid from JAVA7 version onwards, because, in JAVA6 version JVM will load main class bytecode to the memory without checking main() method availability but in JAVA 7, first, JVM will search for main() method, if it is available then only JVM will load main class bytecode to the memory, if it is not existed then JVM will provide the following Error message.

Error: Main method not found in class Test, please define the main method as:

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

**Q) Is it possible to display a line of text on command prompt without using main() method, static variable, static method, static block?**

Yes, it is possible to display a line of text on command prompt without using main() method, a static variable, a static method and static block but by using "static Anonymous Inner class of Object class".

**EX:**

```
1) class Test {  
2)     static Object obj = new Object() {  
3)         {  
4)             System.out.println("Welcome To durga Software Solutions");  
5)             System.exit(0); //TO termiate the application.  
6)         }  
7)     };  
8) }
```

**OUTPUT:**

JAVA6:Welcome to Durga Software Soultons

JAVA7:Error:main method not found in class Test,please define the main method as:

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

## 4) static import:

- In Java, applications, if we import a particular package to the present java file then it is possible to access all the classes and interfaces of that package directly without using package name every time as fully qualified name.
- If we want to access classes and interfaces of a particular package without importing then we must use fully qualified name every time that is we have to use package name along with class names.
- A java program without import statement:



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

- A java program With import statement:  
`import java.io.*;  
BufferedReader br = new BufferedReader(new InputStreamReader(System.in));`
- In Java applications, if we want to access static members of a particular class in present java file then we must use either reference variable of the respective class or directly class name.
- In Java applications, if we want to access static members without using the respective class name and without using the respective class reference variable then we have to import static members of that respective class in the present Java file.
- To import static members of a particular class in the present java file, JDK 5.0 version has provided a new feature called as "static import".

## Syntaxes:

- 1) `import static package_Name.Class_Name_Or_Interface_Name.*;`  
It will import all the static members from the specified class or interface.
- 2) `import static package_Name.Class_Name_Or_Interface_Name.member_Name;`  
It will import only the specified member from the specified class or interface.

## EX:

```
1) import static java.lang.Thread.*;  
2) import static java.lang.System.out;  
3) class Test {  
4)     public static void main(String args[]) {  
5)         out.println(MIN_PRIORITY);  
6)         out.println(MAX_PRIORITY);  
7)         out.println(NORM_PRIORITY);  
8)     }  
9) }
```

## OUTPUT:

```
1  
10  
5
```



## Static Context / Static Flow of Execution:

In Java, Static Context will be represented in the form of the following 3 elements.

- 1) Static variables.
- 2) Static methods.
- 3) Static blocks

In Java applications, instance context will be created separately for each and every object but static context will be created for each and every class.

In Java applications, static context will be recognized and executed exactly at the time of loading the respective class bytecode to the memory.

In Java applications, when we create object for a particular class, first, JVM has to access constructor, before executing constructor, JVM has to load the respective class bytecode to the memory.

At the time of loading class bytecode to the memory, JVM has to recognize and execute static context.

EX:

```
1) class A {  
2)     static {  
3)         System.out.println("SB-A");  
4)     }  
5)     static int i = m1();  
6)     static int m1() {  
7)         System.out.println("m1-A");  
8)         return 10;  
9)     }  
10}   
11) class Test {  
12)     public static void main(String args[]) {  
13)         A a = new A();  
14)     }  
15} 
```

OUTPUT:

SB-A  
m1-A

EX:

```
1) class A {  
2)     static int i = m2();  
3)     static int m1() {  
4)         System.out.println("m1-A");  
5)     }  
6) } 
```



```
5)      return 10;
6) }
7) static {
8)     System.out.println("SB-A");
9) }
10) static int m2() {
11)     System.out.println("m2-A");
12)     return 20;
13) }
14) static int j=m1();
15) }
16) class Test {
17)     public static void main(String args[]) {
18)         A a1 = new A();
19)         A a2 = new A();
20)     }
21) }
```

**OUTPUT:**

m2-A  
SB-A  
m1-A

**EX:**

```
1) class A {
2)     static int i = m2();
3)     A() {
4)         System.out.println("A-con");
5)     }
6)     int m1() {
7)         System.out.println("m1-A");
8)         return 10;
9)     }
10)    static {
11)        System.out.println("SB-A");
12)    }
13)    int j = m1(); {
14)        System.out.println("IB-A");
15)    }
```



```
16) static int m2() {
17)     System.out.println("m2-A");
18)     return 10;
19)
20}
21) class Test {
22)     public static void main(String args[]) {
23)         A a1 = new A();
24)         A a2 = new A();
25)
26}
```

**OUTPUT:**

m2-A  
SB-A  
m1-A  
IB-A  
A-con  
m1-A  
IB-A  
A-con

## Factory Method:

Factory Method is a method, it can be used to return the same class Object where we have declared that method.

Factory Method is an idea provided by a design pattern called as "Factory Method Design Pattern".

**NOTE:** Design pattern is a System, it will provide problem definition and its solution in order to solve design problems.

**EX:**

```
1) class A {
2)     private A() {
3)         System.out.println("A-con");
4)     }
5)     void m1() {
6)         System.out.println("m1-A");
```



```
7)    }
8)    static A getRef()//Factory Method
9)    {
10)       A a = new A();
11)       return a;
12)    }
13)
14) class Test {
15)     public static void main(String args[]) {
16)         A a = A.getRef();
17)         a.m1();
18)     }
19) }
```

### OUTPUT:

A-con  
m1-A

There are 2 types of Factory Methods

- 1) Static Factory Method
- 2) Instance Factory Method

## Static Factory Method:

Static Factory Method is a static method returns the same class object.

### EX:

```
Class c = Class.forName("--);
NumberFormat nf = NumberFormat.getInstance();
DateFormat df = DateFormat.getInstance(--);
ResourceBundle rb = ResourceBundle.getBundle(--);
```

## Instance Factory Method:

If any non-static method returns the same class object then that method is called as "Instance Factory Method".

EX: Almost all the String class methods are Instance Factory methods.

```
String str = new String("DurgaSoftware Solutions");
String str1 = str.concat(" Hyderabad");
String str2 = str.trim();
```



```
String str3 = str.toUpperCase();
String str4 = str.substring(5,14);
```

## **Singleton Class:**

If any JAVA class allows to create only one Object then that class is called as "Singleton Class".

Singleton Class is an idea provided by a design pattern called as "Singleton Design Pattern".

**EX1:**

```
1) class A {
2)     static A a = null;
3)     private A() {
4)     }
5)     static A getRef() {
6)         if(a == null) {
7)             a = new A();
8)             return a;
9)         }
10)    else {
11)        return a;
12)    }
13) }
14)
15) class Test {
16)     public static void main(String args[]) {
17)         System.out.println(A.getRef());
18)         System.out.println(A.getRef());
19)         System.out.println(A.getRef());
20)     }
21) }
```

**OUTPUT:**  
A@a6eb38a  
A@a6eb38a  
A@a6eb38a

Another Alternative for Singleton Class:

```
1) class A {
2)     static A a = null;
3)     static {
4)         a = new A();
5)     }
6)     private A() {
```



```
7)    }
8)    static A getRef() {
9)        return a;
10)   }
11) }
12) class Test {
13)     public static void main(String args[]) {
14)         System.out.println(A.getRef());
15)         System.out.println(A.getRef());
16)         System.out.println(A.getRef());
17)     }
18} }
```

**OUTPUT:**

A@a6eb38a  
A@a6eb38a  
A@a6eb38a

Alternative Logic for Singleton Class:

```
1) class A {
2)     static A a = new A();
3)     private A() {
4)     }
5)     static A getRef() {
6)         return a;
7)     }
8) }
9) class Test {
10)    public static void main(String args[]) {
11)        System.out.println(A.getRef());
12)        System.out.println(A.getRef());
13)        System.out.println(A.getRef());
14)    }
15} }
```

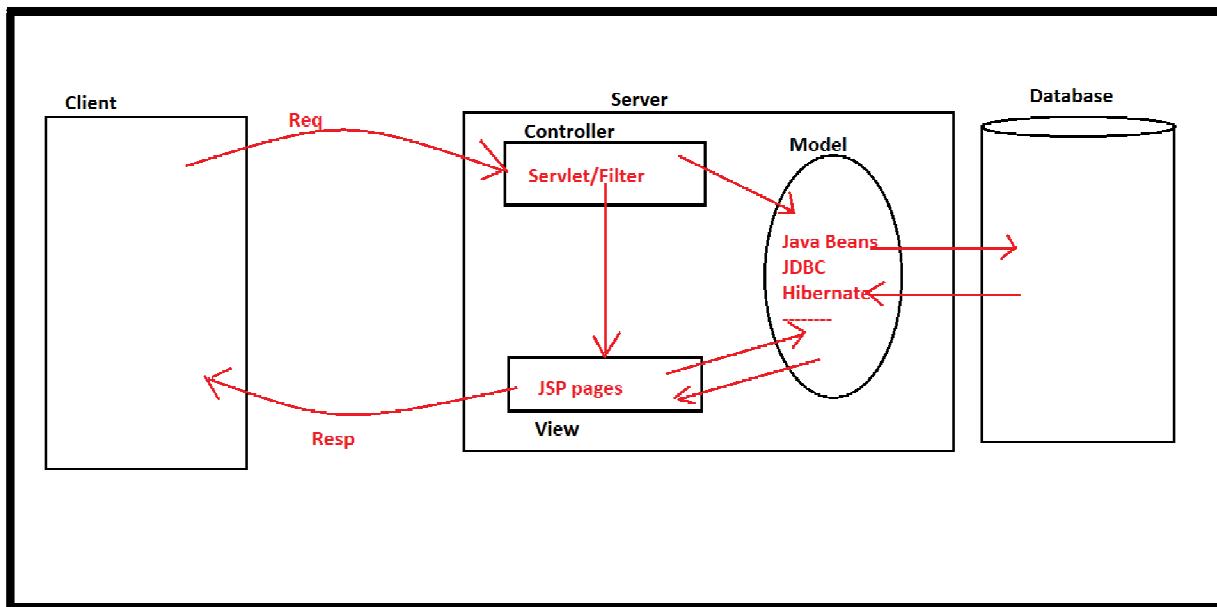
**OUTPUT:**

A@69cd2e5f  
A@69cd2e5f  
A@69cd2e5f

MVC is a design pattern, it will provide a standard structure to prepare web applications/GUI applications, where in MVC based applications we must use a servlet/filter [A Java class] as controller, a set of JSP pages/Html pages as view part and a java bean or EJB component or JDBC program or Hibernate program as Model component. As per MVC Arch rules and regulations, only one controller must be provided for application,



that is, the class which we used as controller must provide one object, therefore, controller class must be singleton class.



**EX1:** Struts is MVC based Framework, where "ActionServlet" is controller, so it is Singleton class.

**EX2:** JSF[Java Server Faces] is MVC based framework, where "FacesServlet" is controller, so it is Singleton class.

---

---

**Class.forName(--)**

Consider the following program

```
1) class A
2) {
3)     static
4)     {
5)         System.out.println("Class Loading");
6)     }
7)     A()
8)     {
9)         System.out.println("Object Creating");
10)
11}
12 class Test
13{
14    public static void main(String[] args)throws Exception
15    {
16        A a=new A();
```



```
17) }
18) }
```

When we access a constructor along with "new" keyword then JVM will perform the following actions automatically.

- 1.JVM will load the respective class bytecode to the memory.
- 2.JVM will create object for the loaded class.

As per the requirement, if we want to load class byte code to the memory without creating object then we have to use the following method from `java.lang.Class` class.

```
public static Class forName(String class_Name) throws ClassNotFoundException
```

EX: `Class c = Class.forName("A");`

When JVM encounter the above instruction, JVM will perform the following actions.

- 1) JVM will take class name from `forName()` method.
- 2) JVM will search for its .class file at current location, at java predefined library and at the locations referred by "classpath" environment variable.
- 3) If the required .class file is not available at all the above locations then JVM will rise an exception like "`java.lang.ClassNotFoundException`".
- 4) If the required .class file is available at either of the above locations then JVM will load its bytecode to the memory.
- 5) After loading class bytecode to the memory, JVM will take metadata of the loaded class like class name, super class details, implemented interfaces details, variables details, methods details,... and JVM will store them by creating `java.lang.Class` object in heap memory and JVM will return the generated Class object reference value from `Class.forName()`;

EX:

```
1) class A
2) {
3)   static
4)   {
5)     System.out.println("Class Loading");
6)   }
7)   A()
8)   {
9)     System.out.println("Object Creating");
10)  }
11) }
12) class Test
13) {
```



```
14) public static void main(String[] args) throws Exception  
15) {  
16)     Class c=Class.forName("A");  
17) }  
18}
```

## OUTPUT: Class Loading

After loading class bytecode by using Class.forName() method, if we want to create object explicitly then we have to use the following method.

`public Object newInstance()throws InstantiationException, IllegalAccessException`

EX: `Object obj=c.newInstance();`

When JVM encounter the above instruction, JVM will perform the following actions.

- 1) JVM will goto the loaded class bytecode and JVM will check whether 0-arg and non-private constructor is existed or not.
- 2) If 0-arg and non-private constructor is existed then JVM will execute that constructor and JVM will create object for the loaded class.
- 3) If the constructor parameterized constructor without 0-arg constructor then JVM will rise an exception like "java.langInstantiationException".
- 4) If the constructor is private constructor then JVM will rise an exception like "java.lang.IllegalAccessException".

Note: If the constructor is both parameterized and 0-arg then JVM will rise "java.lang.InstantiationException" only.

EX:

```
1) class A  
2) {  
3)     static  
4)     {  
5)         System.out.println("Class Loading");  
6)     }  
7)     A0  
8)     {  
9)         System.out.println("Object Creating");  
10)    }  
11) }  
12) class Test  
13){  
14) public static void main(String[] args) throws Exception  
15) {
```



```
16) Class c=Class.forName("A");
17) Object obj=c.newInstance();
18)
19}
```

## OUTPUT:

Class Loading  
Object Creating

In JDBC Appl, we are going to use Driver to map JAVA representations to Database representations and Database representations to Java representations. In Jdbc applications, we must load driver class, not to create object for Driver class, for this, we have to use "Class.forName(-)"

In general, all server side components like Servlets, JSPs, EJBs,... are executed by the server[Container] by following their lifecycle actions like loading, instantiation, initialization,..... In this context, to perform server side components loading Container will use "Class.forName(-)" method and to perform instantiation lifecycle action container will use "newInstance()" method internally.

## Final Keyword:

final is a Java Keyword it can be used to declare constant expressions.  
In java applications, 'final' keyword is able to improve security.

In Java applications, there are three ways to utilize 'final' keyword.

- 1) final variable
- 2) final method
- 3) final class

### **1) final Variable:**

final variable is a variable, it will not allow modifications on its value.

#### EX:

```
final int i=10;
i=i+10;----> Compilation Error
```

#### EX:

```
for(final int i=0;i<10;i++)
{
    System.out.println(i);
```



}

**NOTE:** In general, in bank applications, after creating an account it is possible to change the account details like account name, address details....but it is not possible to update 'accNo' value once it is created. Due to this reason, we have to declare 'accNo' variable as 'final' variable.

## 2) final Method:

- final method is a Java method, it will not allow method overriding.
- In method overriding, we have to provide same method with different implementation at both super class and at sub class, where super class method never be declared as final irrespective of sub class method final.

EX1:

```
1) class A {  
2)     final void m10 {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class B extends A {  
7)     void m10 {  
8)         System.out.println("m1-B");  
9)     }  
10}  
11) class Test {  
12)     public static void main(String[] args) {  
13)         A a = new B();  
14)         a.m10();  
15)     }  
16}
```

Status: Compilation Error.

EX2:

```
1) class A {  
2)     final void m10 {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class B extends A {  
7)     final void m10 {  
8)         System.out.println("m1-B");  
9)     }  
10}
```



```
11) class Test {  
12)     public static void main(String[] args) {  
13)         A a = new B();  
14)         a.m1();  
15)     }  
16) }
```

Status: Compilation Error

EX3:

```
1) class A {  
2)     void m1() {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class B extends A {  
7)     final void m1() {  
8)         System.out.println("m1-B");  
9)     }  
10) }  
11) class Test {  
12)     public static void main(String[] args) {  
13)         A a = new B();  
14)         a.m1();  
15)     }  
16) }
```

Status: No Compilation Error.

### 3) final Class:

- Final class is a Java class, it will not allow inheritance.
- In Java applications, super classes never be declared as final classes, but sub classes may be final.

EX:

```
1) final class A  
2) {  
3) }  
4) class B extends A  
5) {  
6) }
```

Status: Invalid



EX:

```
1) final class A  
2) {  
3) }  
4) final class B extends A  
5) {  
6) }
```

Status: Invalid

EX:

```
1) class A  
2) {  
3) }  
4) final class B extends A  
5) {  
6) }
```

Status: Valid

In Java applications to declare constant variables Java has provided a convention like to declare constants with "public static final".

EX:

In System Class:

```
public static final PrintStream out;  
public static final InputStream in;  
public static final PrintStream err;
```

In Thread Class:

```
public static final int MIN_PRIORITY=1;  
public static final int NORM_PRIORITY=5;  
public static final int MAX_PRIORITY=10;
```

EX:

```
1) class User_Status {  
2)     public static final String AVAILABLE="Available";  
3)     public static final String BUSY="Busy";  
4)     public static final String IDLE="Idle";  
5) }  
6) class Test {  
7)     public static void main(String args[]) {  
8)         System.out.println(User_Status.AVAILABLE);
```



```
9)     System.out.println(User_Status.BUSY);
10)    System.out.println(User_Status.IDLE);
11) }
12) }
```

**OUTPUT:**

Available

Busy

Idle

To declare constant variables in Java applications if we use the above convention then we are able to get the following problems.

- 1) We must declare "public static final" for each and every constant variable explicitly.
- 2) It is possible to allow multiple data types to represent one type, it will reduce typedness in Java applications.
- 3) If we access constant variables then these variables will display their values, here constant variable values may or may not reflect the actual meaning of constant variables.

To overcome all the problems, we have to go for "enum".

In case of enum,

- 1) All the constant variables are by default "public static final", no need to declare explicitly.
- 2) All the constant variables are by default the same enum type, it will improve typedness in Java applications.
- 3) All the constant variables are by default "Named Constants" that is, these constant variables are displaying their names instead of their values.

**Syntax:**

```
[Access_modifier] enum Enum_Name
{
    ----- List of constants-----
}
```

**EX:**

```
1) enum User_Status {
2)     AVAILABLE,BUSY,IDLE;
3) }
4) class Test {
```



```
5) public static void main(String args[]) {  
6)     System.out.println(User_Status.AVAILABLE);  
7)     System.out.println(User_Status.BUSY);  
8)     System.out.println(User_Status.IDLE);  
9) }  
10}
```

## OUTPUT:

Available

Busy

Idle

**NOTE:** The default super class for every enum is "java.lang.Enum" class and "Object" class is Super class to "Enum" class.

If we compile the above Java file then Compiler will translate "User\_Status" enum into "User\_Status" final class like below.

```
final class User_Status extends java.lang.Enum {  
    public static final MailStatus AVAILABLE;  
    public static final MailStatus BUSY;  
    public static final MailStatus IDLE;  
    -----  
}
```

**NOTE:** In Java, java.lang.Object class is common and default super class for all the classes. Similarly, All the Java enums are having a common and default super class that is "java.lang.Enum".

**NOTE:** In Java applications, it is possible to implement inheritance between two classes but it is not possible to implement inheritance between two "enums", because, by default, enums are final classes.

In Java applications, we can utilize enum like as classes, where we can provide normal variables, methods, constructors....

## EX

```
1) enum Apple {  
2)     A(500),B(250),C(100);  
3)     int price;  
4)     Apple(int price) {  
5)         this.price = price;  
6)     }  
7)     public int getPrice() {  
8)         return price;
```



```
9)    }
10) }
11) class Test {
12)     public static void main(String args[]) {
13)         System.out.println("A-Grade Apple :"+Apple.A.getPrice());
14)         System.out.println("B-Grade Apple :"+Apple.B.getPrice());
15)         System.out.println("C-Grade Apple :"+Apple.C.getPrice());
16)     }
17} }
```

**OUTPUT:** A-Grade Apple :500  
B-Grade Apple :250  
C-Grade Apple :100

If we compile the above program, then compiler will translate enum into the following class:

```
1) final class Apple extends Enum {
2)     public static final Apple A = new Apple(500);
3)     public static final Apple B = new Apple(250);
4)     public static final Apple C = new Apple(100);
5)     int price;
6)     Apple(int price) {
7)         this.price = price;
8)     }
9)     public int getPrice() {
10)        return price;
11)    }
12) ----
13} }
```

**EX:**

```
1) enum Book {
2)     A(500,250),B(300,150),C(200,100);
3)     int no_of_pages;
4)     int cost;
5)     Book(int no_of_pages, int cost) {
6)         this.no_of_pages = no_of_pages;
7)         this.cost = cost;
8)     }
9)     public void getBookDetails() {
10)        System.out.println(no_of_pages+"----->"+cost);
11)    }
12} }
```



```
13) class Test {  
14)     public static void main(String args[]) {  
15)         System.out.println("Durga Books Store");  
16)         System.out.println("-----");  
17)         System.out.println("No of Pages Cost");  
18)         System.out.println("-----");  
19)         Book.A.getBookDetails();  
20)         Book.B.getBookDetails();  
21)         Book.C.getBookDetails();  
22)     }  
23} }
```

## OUTPUT:

Durga Books Store

-----  
No of Pages Cost  
-----  
500----->250  
300----->150  
200----->100

If we compile the above program, then compiler will translate the enum into the following class

Translated Code for the above enum(Book):

```
1) final class Book extends Enum {  
2)     public static final Book A=new Book(500,250);  
3)     public static final Book B=new Book(300,150);  
4)     public static final Book C=new Book(200,100);  
5)     int cost;  
6)     int no_of_pages;  
7)     Book(int no_of_pages,int cost) {  
8)         this.no_of_pages = no_of_pages;  
9)         this.cost = cost;  
10)    }  
11)    public void getBookDetails() {  
12)        System.out.println(no_of_pages+"----->" +cost);  
13)    }  
14)    ---  
15) }
```

## Importance of main() method in Java:

The main intention of main() method in Java applications is,



- 
- 1) To define application logic in Java program.
  - 2) To define starting point and ending point for the applications execution.

### Syntax:

```
public static void main(String args[])
{
    ----application logic---
}
```

**Note:** Main() method is not predefined method and it is not user defined method, it is a conventional method with fixed prototype and with user defined implementation.  
In Java, JVM is implemented in such a way to access main() method automatically in order to execute application logic.

### **Q) What is the requirement to declare main() method as public?**

In Java applications, JVM has to access main() method in order to start application execution. To access main() method by JVM first main() scope must be available to JVM. In this context, to bring main() method scope to JVM, we must declare main() method as "public".

#### Case-1:

If main() method is declared as "private" then main() method will be available up to the main class only ,not to JVM.

#### Case-2:

If main() method is declared as "<default>" then main() method will be available up to the package where main class is existed, not to the JVM.

#### Case-3:

If main() method is declared as "protected" then main() method will be available up to the package where main class is existed and up to child classes available in other packages but not to the JVM.

#### Case-4:

To make available main() method to JVM, only one possibility we have to take that is "public", where public members are available throughout our system, so that, JVM can access main() method to start application execution.

**NOTE:** In Java applications, if we declare main() method without "public" then compiler will not rise any error but JVM will provide the following.

JAVA6:Main Method not public.

JAVA7:Error:Main method not found in class Test, please define main method as:

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



## Q) What is the requirement to declare main() method as "static"?

In Java applications, to start application execution JVM has to access main() method. JVM was implemented in such a way that to access main() method by using the respective main class directly.

In Java applications, only static methods are eligible to access by using their respective class name, so that, as per the JVM predefined implementation we must declare main() method as static.

**NOTE:** In Java applications, if we declare main() method without "static" then compiler will not rise any error but JVM will provide the following.

JAVA6: `java.lang.NoSuchMethodError:main`

JAVA7: `Error:Main method is not static in class Test,please define main method as:  
public static void main(String[] args)`

## Q)What is the requirement to provide "void" as return type to main() method?

In Java applications, as per Java conventions, JVM will start application execution at the starting point of main method and JVM will terminate application execution at the ending point of main() method. Due to this convention, we must start application logic at the starting point of main() method and we must terminate application logic at the ending point of main() method. To follow this Java convention we must not return any value from main() method, for this, we must provide "void" as return type.

**NOTE:** In Java applications, if we declare main() method without void return type then compiler will not rise any error but JVM will provide the following

JAVA6: `java.lang.NoSuchMethodError:main`

JAVA7: `Error:Main method must return a value of type void in class Test,please define the main method as:`

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

**NOTE:** The name of this method "main" is to show the importance of this method.

## Main Method Parameters:

### Q) What is the Requirement to provide Parameter to main() Method?

In Java applications, there are 3 ways to provide input to the Java programs.

- 1) Static Input
- 2) Dynamic Input
- 3) Command Line Input



## 1) Static Input:

Providing input to the Java program at the time of writing Java program.

EX:

```
1) class Test
2) {
3)   int i=10;
4)   int j=20;
5)   void add()
6)   {
7)     System.out.println(i+j);
8)   }
9) }
```

## 2) Dynamic Input:

Providing Input to the Java program at runtime.

```
D:\Java7>javac Add.java
D:\java7>java Add
First value : 10
Second value :20
Addition :30
```

## 3) Command Line Input:

Providing input to the Java program along with "java" command on command prompt.

EX: D:\java7>javac Add.java  
D:\java7>java Add 10 20  
Addition :30

If we provide command line input like above in Java applications then JVM will perform the following actions.

- 1) JVM will read all the command line input at the time of reading main class name from command prompt.
- 2) JVM will store all the command line inputs in the form of String[]
- 3) JVM will pass the generated String[] as parameter to main() method at the time of calling main() method.

Due to the above JVM actions, the main method is required parameters in order to store all the command line inputs in the form String[] array.



## Q)What is the requirement to provide only String data type as parameter to the main() method?

In general, from application to application or from developer to developer the types of command line input may be varied, here even developers are providing different types of command line input, still, our main() method must store all the command line input. In Java, only String data types is able to represent any type of data so that String data types is required as parameter to main() method.

To allow different types of command line input, main() method must require String dataType.

## Q)What is the requirement to provide an array as parameter to main() method?

In general, from application to application and from developer to developer number of command line inputs may be varied, here even developers are providing variable number of command line inputs our main() method parameter must store them. In Java, to store more than one value we have to take array. Due to this reason, main() method must require array type as parameter. main() method will take array type as parameter is to allow multiple number of command line input.

### EX:

```
1) class Test {  
2)     public static void main(String args[]) {  
3)         for(int i=0;i<args.length;i++) {  
4)             System.out.println(args[i]);  
5)         }  
6)     }
```

### OUTPUT:

```
D:\java7>javac Test.java  
D:\java7>java Test 10 "abc" 22.22f 34.345 'A' true  
10  
"abc"  
22.22f  
34.345  
'A'  
true
```

### EX:



```
1) class Test
2) {
3)     public static void main(String[] args) throws Exception
4)     {
5)         String val1=args[0];
6)         String val2=args[1];
7)         int fval=Integer.parseInt(val1);
8)         int sval=Integer.parseInt(val2);
9)         System.out.println("ADD :" +(fval+sval));
10)        System.out.println("SUB :" +(fval-sval));
11)        System.out.println("MUL :" +(fval*sval));
12)    }
13} }
```

#### OUTPUT:

D:\javaapps>javac Test.java

D:\javaapps>java Test 10 5

ADD :15

SUB :5

MUL :50

**NOTE:** If we declare main() method without String[] parameter then compiler will not rise any error but JVM will provide the following.

JAVA6:java.lang.NoSuchMethodError:main

JAVA7:Error:Main Method not found in class Test,please define main method as:

public static void main(String args[])

#### Q)Find the valid syntaxes of main() method from the following list?

- 1) public static void main(String[] args)--> Valid
- 2) public static void main(String[] abc)--> valid
- 3) public static void main(String args)----> Invalid
- 4) public static void main(String[][] args)-->Invalid
- 5) public static void main(String args[])---> Valid
- 6) public static void main(String []args)---> Valid
- 7) public static void main(string[] args)---> Invalid
- 8) public static void Main(String[] args)---> Invalid
- 9) public static int main(String[] args)----> Invalid
- 10) public final void main(String[] args)---> Invalid
- 11) public void main(String[] args)-----> Invalid
- 12) static void main(String[] args)-----> Invalid
- 13) static public void main(String[] args)--> Valid
- 14) public static void main(String ... args)-> Valid



## Q) Is it possible to provide more than one main() method within a single java application?

Yes, it is possible to provide more than one main() method within a single java application, but, we have to provide more than one main() method in different classes, not in a single class.

EX: File Name : D:\java7\abc.java

```
1) class A {  
2)     public static void main(String args[]) {  
3)         System.out.println("main()-A");  
4)     }  
5) }  
6) class B {  
7)     public static void main(String args[]) {  
8)         System.out.println("main()-B");  
9)     }  
10)
```

### OUTPUT:

D:\java7>javac abc.java

D:\java7>java A

main()-A

D:\java7>java B

main()-B

**NOTE:** If we compile the above abc.java file then compiler will generate two .class files [A.class,B.class]. To execute the above program, we have to give a class name along with "java" command, here which class name we are providing along with "java" command that class main() method will be executed by JVM.

**NOTE:** In the above program, it is possible to access one main() method from another main() method by passing String[] as parameter and by using the respective class name as main() method is static method.

EX: File name : D:\java7\abc.java

```
1) class A {  
2)     public static void main(String args[]) {  
3)         System.out.println("main()-A");  
4)         String[] str = {"AAA","BBB","CCC"};  
5)         B.main(str);  
6)         B.main(args);  
7)     }  
8) }  
9) class B {
```



```
10) public static void main(String args[]) {  
11)     System.out.println("main()-B");  
12) }  
13) }
```

## OUTPUT:

```
D:\java7>javac abc.java  
D:\java7>java A  
main()-A  
main()-B  
main()-B
```

## Q) Is it possible to overload main() method?

Yes, In Java, it is possible to overload main() method but it is not possible to override main() method, because, in Java applications static method overloading is possible but static method overriding is not possible.

## EX:

```
1) class Test  
2) {  
3)     public static void main(String[] args)  
4)     {  
5)         System.out.println("String[]-param-main()");  
6)     }  
7)     public static void main(int[] args)  
8)     {  
9)         System.out.println("int[]-param-main()");  
10)    }  
11)    public static void main(float[] args)  
12)    {  
13)        System.out.println("float[]-param-main()");  
14)    }  
15) }
```

OUTPUT: String[]-param-main

## Relationships in Java

As part of Java application development, we have to use entities as per the application requirements.

In Java application development, if we want to provide optimizations over memory utilization, code Reusability, Execution Time, Sharability then we have to define relationships between entities.



There are 3 types of relationships between entities.

- 1) Has-A Relationship
- 2) IS-A Relationship
- 3) USE-A Relationship

## Q)What is the difference between HAS-A Relationship and IS-A Relationship?

Has-A relationship will define associations between entities in Java applications, here associations between entities will improve communication between entities and data navigation between entities.

IS-A Relationship is able to define inheritance between entity classes, it will improve "Code Reusability" in java applications.

## Associations in JAVA:

There are four types of associations between entities

- 1) One-To-One Association
- 2) One-To-Many Association
- 3) Many-To-One Association
- 4) Many-To-Many Association

To achieve associations between entities, we have to declare either single reference or array of reference variables of an entity class in another entity class.

EX:

```
class Address {  
----  
}  
class Student {  
----  
Address[] addr;-->It will establish One-To-Many Association  
}
```

### **1) One-To-One Association:**

It is a relation between entities, where one instance of an entity should be mapped with exactly one instance of another entity.

EX: Every employee should have exactly one Account.

- ```
1) class Account {  
2)     String accNo;
```



```
3)     String accName;
4)     String accType;
5)     Account(String accNo,String accName,String accType) {
6)         this.accNo = accNo;
7)         this.accName = accName;
8)         this.accType = accType;
9)     }
10) }
11) class Employee {
12)     String eid;
13)     String ename;
14)     String eaddr;
15)     Account acc;
16)     Employee(String eid, String ename, String eaddr, Account acc) {
17)         this.eid = eid;
18)         this.ename = ename;
19)         this.eaddr = eaddr;
20)         this.acc = acc;
21)     }
22)     public void getEmployee() {
23)         System.out.println("Employee Details");
24)         System.out.println("-----");
25)         System.out.println("Employee Id : "+eid);
26)         System.out.println("Employee Name : "+ename);
27)         System.out.println("Employee Address : "+eaddr);
28)         System.out.println();
29)         System.out.println("Account Details");
30)         System.out.println("-----");
31)         System.out.println("Account Number : "+acc.accNo);
32)         System.out.println("Account Name : "+acc.accName);
33)         System.out.println("Account Type : "+acc.accType);
34)     }
35) }
36) class OneToOneEx {
37)     public static void main(String args[]) {
38)         Account acc = new Account("abc123","Durga N","Savings");
39)         Employee emp = new Employee("E-111","Durga","Hyd",acc);
40)         emp.getEmployee();
41)     }
42) }
```

## OUTPUT:

Employee Details

-----

Employee Id: E-111

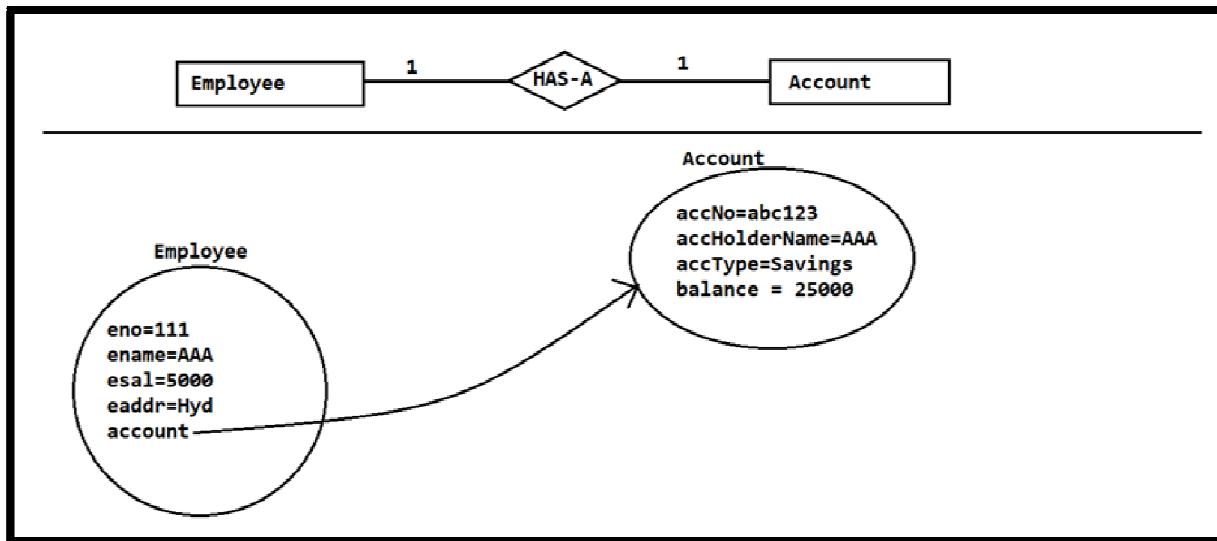


Employee Name: Durga  
Employee Address: Hyd

## Account Details

---

Account Number :abc123  
Account Name :Durga N  
Account Type :Savings



## Dependency Injection:

--> The process of injecting contained object in Container object is called as Dependency Injection.

There are three types of Dependency Injection.

1. Constructor Dependency injection.
2. Setter Method Dependency Injection.
3. Interfaces Dependency Injection.

## 1. Constructor Dependency injection:

The process of injecting dependent object to the container object through a constructor then that type of Dependency Injection is called as "Constructor Dependency Injection".

### Account.java

```
1) package com.durgasoft.entities;  
2)  
3) public class Account {  
4)     String accNo;
```



```
5) String accHolderName;
6) String accType;
7) int accBalance;
8)
9) public Account(String accNo, String accHolderName, String accType, int accBalanc
ce) {
10)     this.accNo = accNo;
11)     this.accHolderName = accHolderName;
12)     this.accType = accType;
13)     this.accBalance = accBalance;
14) }
15} }
```

## Employee.java

```
1) package com.durgasoft.entities;
2)
3) public class Employee {
4)     int eno;
5)     String ename;
6)     float esal;
7)     String eaddr;
8)
9)     Account acc;
10)    public Employee(int eno, String ename, float esal, String eaddr, Account acc) {
11)        this.eno = eno;
12)        this.ename = ename;
13)        this.esal = esal;
14)        this.eaddr = eaddr;
15)        this.acc = acc;
16)    }
17)
18)    public void getEmpDetails() {
19)        System.out.println("Employee Details");
20)        System.out.println("-----");
21)        System.out.println("Employee Number : "+eno);
22)        System.out.println("Employee Name : "+ename);
23)        System.out.println("Employee Salary : "+esal);
24)        System.out.println("Employee Address : "+eaddr);
25)        System.out.println();
26)        System.out.println("Account Details");
27)        System.out.println("-----");
28)        System.out.println("Account Number : "+acc.accNo);
29)        System.out.println("Account Holder Name : "+acc.accHolderName);
30)        System.out.println("Account Type : "+acc.accType);
```



```
31)     System.out.println("Account Balance : "+acc.accBalance);
32) }
33) }
```

## Test.java

```
1) package com.durgasoft.core;
2)
3) import com.durgasoft.entities.Employee;
4) import com.durgasoft.entities.Account;
5) public class Test {
6)
7)     public static void main(String[] args) {
8)         Account acc = new Account("abc123", "Durga", "Savings", 25000);
9)         Employee emp = new Employee(111, "Durga", 15000, "Hyd", acc);
10)        emp.getEmpDetails();
11)    }
12) }
```

## 2. Setter Method Dependency Injection:

The process of injecting the dependent object into the Container object then it is called as "Setter Method Dependency Injection".

### EX: Account.java

```
1) package com.durgasoft.entities;
2)
3) public class Account {
4)     private String accNo;
5)     private String accHolderName;
6)     private String accType;
7)     private int accBalance;
8)
9)     public String getAccNo() {
10)         return accNo;
11)     }
12)     public void setAccNo(String accNo) {
13)         this.accNo = accNo;
14)     }
15)     public String getAccHolderName() {
```



```
16)    return accHolderName;
17)
18) public void setAccHolderName(String accHolderName) {
19)     this.accHolderName = accHolderName;
20)
21) public String getAccType() {
22)     return accType;
23)
24) public void setAccType(String accType) {
25)     this.accType = accType;
26)
27) public int getAccBalance() {
28)     return accBalance;
29)
30) public void setAccBalance(int accBalance) {
31)     this.accBalance = accBalance;
32)
33}
```

## Employee.java

```
1) package com.durgasoft.entities;
2)
3) public class Employee {
4)     private int eno;
5)     private String ename;
6)     private float esal;
7)     private String eaddr;
8)
9)     private Account acc;
10)
11)    public int getEno() {
12)        return eno;
13)    }
14)    public void setEno(int eno) {
15)        this.eno = eno;
16)    }
17)    public String getEname() {
18)        return ename;
19)    }
20)    public void setEname(String ename) {
```



```
21)     this.ename = ename;
22) }
23) public float getEsal() {
24)     return esal;
25) }
26) public void setEsal(float esal) {
27)     this.esal = esal;
28) }
29) public String getEaddr() {
30)     return eaddr;
31) }
32) public void setEaddr(String eaddr) {
33)     this.eaddr = eaddr;
34) }
35) public Account getAcc() {
36)     return acc;
37) }
38) public void setAcc(Account acc) {
39)     this.acc = acc;
40) }
41)
42) public void getEmpDetails() {
43)     System.out.println("Employee Details");
44)     System.out.println("-----");
45)     System.out.println("Employee Number : "+eno);
46)     System.out.println("Employee Name : "+ename);
47)     System.out.println("Employee Salary : "+esal);
48)     System.out.println("Employee Address : "+eaddr);
49)     System.out.println();
50)
51)     System.out.println("Account Details");
52)     System.out.println("-----");
53)     System.out.println("Account Number : "+acc.getAccNo());
54)     System.out.println("Account Holder Name : "+acc.getAccHolderName());
55)     System.out.println("Account Type : "+acc.getAccType());
56)     System.out.println("Account Balance : "+acc.getAccBalance());
57) }
58} }
```

## Test.java

```
1) package com.durgasoft.test;
2)
3) import com.durgasoft.entities.Account;
4) import com.durgasoft.entities.Employee;
```



```
5)
6) public class Test {
7)
8)     public static void main(String[] args) {
9)         Account acc = new Account();
10)        acc.setAccNo("abc123");
11)        acc.setAccHolderName("Durga");
12)        acc.setAccType("Savings");
13)        acc.setAccBalance(25000);
14)
15)        Employee emp = new Employee();
16)        emp.setEno(111);
17)        emp.setEname("Durga");
18)        emp.setEsal(10000);
19)        emp.setEaddr("Hyd");
20)        emp.setAcc(acc);
21)
22)        emp.getEmpDetails();
23)
24)    }
25)
26} }
```

## 2. One-To-Many Association:

It is a relationship between entity classes, where one instance of an entity should be mapped with multiple instances of another entity.

EX: Single department has multiple employees.

```
1) class Employee {
2)     String eid;
3)     String ename;
4)     String eaddr;
5)     Employee(String eid, String ename, String eaddr) {
6)         this.eid = eid;
7)         this.ename = ename;
8)         this.eaddr = eaddr;
9)     }
10) }
11) class Department {
12)     String did;
13)     String dname;
14)     Employee[] emps;
15)     Department(String did, String dname, Employee[] emps) {
16)         this.did = did;
```



```
17)         this.dname = dname;
18)         this.emps = emps;
19)     }
20)     public void getDepartmentDetails() {
21)         System.out.println("Department Details");
22)         System.out.println("-----");
23)         System.out.println("Department Id :" + did);
24)         System.out.println("Department Name:" + dname);
25)         System.out.println();
26)         System.out.println("EID ENAME EADDR");
27)         System.out.println("-----");
28)         for(int i=0; i<emps.length; i++) {
29)             Employee e = emps[i];
30)             System.out.println(e.eid+ " " +e.ename+ " " +e.eaddr);
31)         }
32)     }
33}
34 class OneToManyEx {
35)     public static void main(String args[]) {
36)         Employee e1=new Employee("E-111","AAA","Hyd");
37)         Employee e2=new Employee("E-222","BBB","Hyd");
38)         Employee e3=new Employee("E-333","CCC","Hyd");
39)         Employee[] emps=new Employee[3];
40)         emps[0]=e1;
41)         emps[1]=e2;
42)         emps[2]=e3;
43)         Department dept=new Department("D-111","Admin",emps);
44)         dept.getDepartmentDetails();
45)
46}
```

## OUTPUT:

Department Details

-----

Department Id: D-111

Department Name: Admin

EID ENAME EADDR

-----

|       |     |     |
|-------|-----|-----|
| E-111 | AAA | Hyd |
| E-222 | BBB | Hyd |
| E-333 | CCC | Hyd |

## EX:



## Employee.java

```
1) package com.durgasoft.entities;
2)
3) public class Employee {
4)     private int eno;
5)     private String ename;
6)     private float esal;
7)     private String eaddr;
8)
9)     public int getEno() {
10)         return eno;
11)     }
12)     public void setEno(int eno) {
13)         this.eno = eno;
14)     }
15)     public String getEname() {
16)         return ename;
17)     }
18)     public void setEname(String ename) {
19)         this.ename = ename;
20)     }
21)     public float getEsal() {
22)         return esal;
23)     }
24)     public void setEsal(float esal) {
25)         this.esal = esal;
26)     }
27)     public String getEaddr() {
28)         return eaddr;
29)     }
30)     public void setEaddr(String eaddr) {
31)         this.eaddr = eaddr;
32)     }
33} }
```

## Department.java

```
1) package com.durgasoft.entities;
2)
3) public class Department {
4)     private String did;
5)     private String dname;
6)     private Employee[] emps;
```



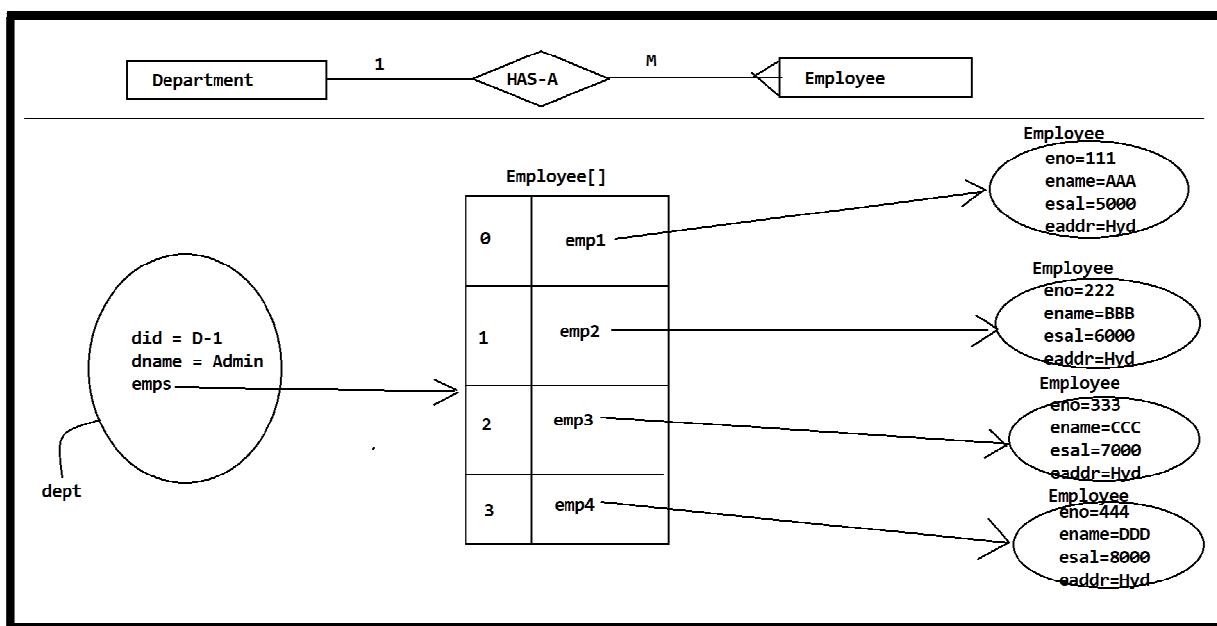
```
7)
8)     public String getDid() {
9)         return did;
10)    }
11)    public void setDid(String did) {
12)        this.did = did;
13)    }
14)    public String getDname() {
15)        return dname;
16)    }
17)    public void setDname(String dname) {
18)        this.dname = dname;
19)    }
20)    public Employee[] getEmps() {
21)        return emps;
22)    }
23)    public void setEmps(Employee[] emps) {
24)        this.emps = emps;
25)    }
26)
27)    public void getDeptDetails() {
28)        System.out.println("Department Details");
29)        System.out.println("-----");
30)        System.out.println("Department Id : "+did);
31)        System.out.println("Department Name : "+dname);
32)        System.out.println();
33)        System.out.println("ENO\tENAME\tESAL\tEADDR");
34)        System.out.println("-----");
35)        for(Employee emp: emps) {
36)            System.out.print(emp.getENO()+"\t");
37)            System.out.print(emp.getENAME()+"\t");
38)            System.out.print(emp.getESAL()+"\t");
39)            System.out.print(emp.getEADDR()+"\n");
40)        }
41)
42)    }
43} }
```

## Test.java

```
1) package com.durgasoft.test;
2)
3) import com.durgasoft.entities.Department;
4) import com.durgasoft.entities.Employee;
5)
```



```
6) public class Test {  
7)  
8)     public static void main(String[] args) {  
9)         Employee emp1 = new Employee();  
10)        emp1.setEno(111);  
11)        emp1.setEname("AAA");  
12)        emp1.setEsal(5000);  
13)        emp1.setEaddr("Hyd");  
14)  
15)        Employee emp2 = new Employee();  
16)        emp2.setEno(222);  
17)        emp2.setEname("BBB");  
18)        emp2.setEsal(6000);  
19)        emp2.setEaddr("Chennai");  
20)  
21)        Employee emp3 = new Employee();  
22)        emp3.setEno(333);  
23)        emp3.setEname("CCC");  
24)        emp3.setEsal(7000);  
25)        emp3.setEaddr("Delhi");  
26)  
27)        Employee emp4 = new Employee();  
28)        emp4.setEno(444);  
29)        emp4.setEname("DDD");  
30)        emp4.setEsal(8000);  
31)        emp4.setEaddr("Pune");  
32)  
33)        Employee[] emps = {emp1, emp2, emp3, emp4};  
34)  
35)        Department dept = new Department();  
36)        dept.setDid("D-111");  
37)        dept.setDname("Admin");  
38)        dept.setEmps(emps);  
39)        dept.getDeptDetails();  
40)  
41)    }  
42} }
```



## Many-To-One Association:

It is a relationship between entities, where multiple instances of an entity should be mapped with exactly one instance of another entity.

EX: Multiple Students have joined with a single branch.

```
1) class Branch {  
2)     String bid;  
3)     String bname;  
4)     Branch(String bid, String bname) {  
5)         this.bid = bid;  
6)         this.bname = bname;  
7)     }  
8) }  
9) class Student {  
10)    String sid;  
11)    String sname;  
12)    String saddr;  
13)    Branch branch;  
14)    Student(String sid, String sname, String saddr, Branch branch) {  
15)        this.sid = sid;  
16)        this.sname = sname;  
17)        this.saddr = saddr;  
18)        this.branch = branch;  
19)    }  
20)    public void getStudentDetails() {  
21)        System.out.println("Student Details");  
22)        System.out.println("-----");
```



```
23)         System.out.println("Student Id  :" + sid);
24)         System.out.println("Student name : " + sname);
25)         System.out.println("Student Address:" + saddr);
26)         System.out.println("Branch Id  :" + branch.bid);
27)         System.out.println("Branch Name :" + branch.bname);
28)         System.out.println();
29)     }
30) }
31) class ManyToOneEx {
32)     public static void main(String args[]) {
33)         Branch branch=new Branch("B-111","CS");
34)         Student std1=new Student("S-111","AAA","Hyd",branch);
35)         Student std2=new Student("S-222","BBB","Hyd",branch);
36)         Student std3=new Student("S-333","CCC","Hyd",branch);
37)         std1.getStudentDetails();
38)         std2.getStudentDetails();
39)         std3.getStudentDetails();
40)     }
41} }
```

## OUTPUT:

### Student Details

-----  
Student Id :S-111  
Student name:AAA  
Student Address:Hyd  
Branch Id :B-111  
Branch Name:CS

### Student Details

-----  
Student Id :S-222  
Student name:BBB  
Student Address:Hyd  
Branch Id :B-111  
Branch Name:CS

### Student Details

-----  
Student Id :S-333  
Student name:CCC  
Student Address:Hyd  
Branch Id: B-111  
Branch Name: CS



EX:

## Branch.java

```
1) package com.durgasoft.entities;
2)
3) public class Branch {
4)     private String bid;
5)     private String bname;
6)
7)     public String getBid() {
8)         return bid;
9)     }
10)    public void setBid(String bid) {
11)        this.bid = bid;
12)    }
13)    public String getBname() {
14)        return bname;
15)    }
16)    public void setBname(String bname) {
17)        this.bname = bname;
18)    }
19} }
```

## Student.java

```
1) package com.durgasoft.entities;
2)
3) public class Student {
4)     private String sid;
5)     private String sname;
6)     private String saddr;
7)     private Branch branch;
8)
9)     public String getSid() {
10)        return sid;
11)    }
12)    public void setSid(String sid) {
13)        this.sid = sid;
14} }
```



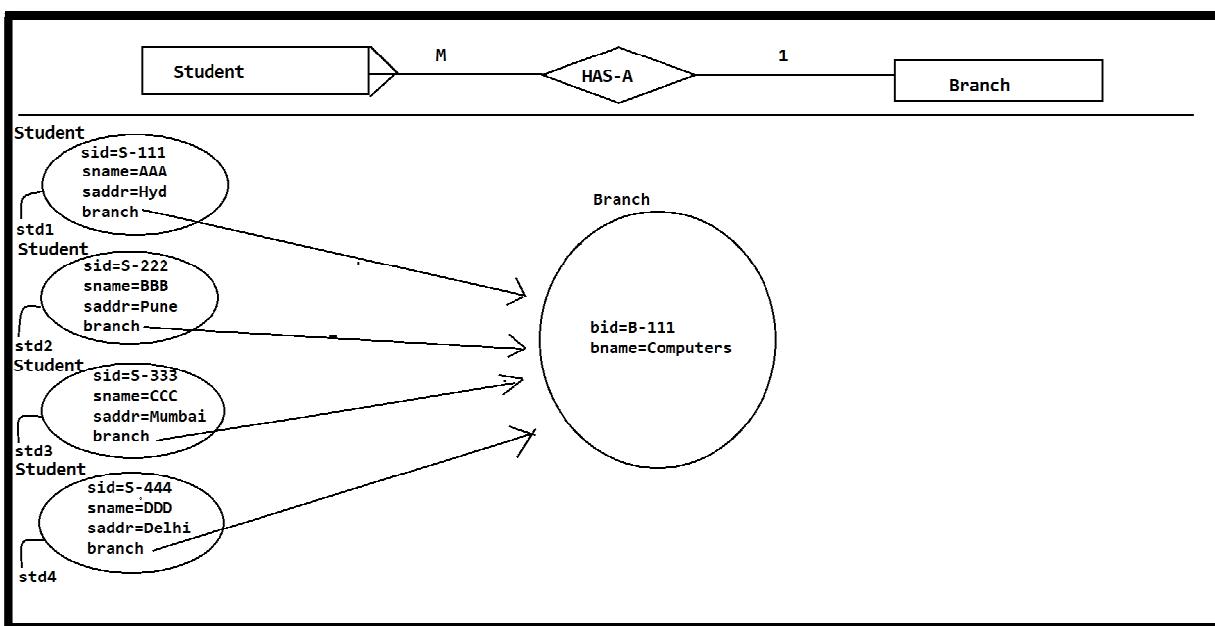
```
14) }
15) public String getSname() {
16)     return sname;
17) }
18) public void setSname(String sname) {
19)     this.sname = sname;
20) }
21) public String getSaddr() {
22)     return saddr;
23) }
24) public void setSaddr(String saddr) {
25)     this.saddr = saddr;
26) }
27) public Branch getBranch() {
28)     return branch;
29) }
30) public void setBranch(Branch branch) {
31)     this.branch = branch;
32) }
33)
34) public void getStudentDetails() {
35)     System.out.println("Student Details");
36)     System.out.println("-----");
37)     System.out.println("Student Id : "+sid);
38)     System.out.println("Student Name : "+sname);
39)     System.out.println("Student Address : "+saddr);
40)     System.out.println("Branch Id : "+branch.getBid());
41)     System.out.println("Branch Name : "+branch.getBname());
42) }
43} }
```

## Test.java

```
1) package com.durgasoft.test;
2)
3) import com.durgasoft.entities.Branch;
4) import com.durgasoft.entities.Student;
5)
6) public class Test {
7)
8)     public static void main(String[] args) {
9)         Branch branch = new Branch();
10)        branch.setBid("B-111");
11)        branch.setBname("Computers");
12)    }
```



```
13) Student std1 = new Student();
14) std1.setSid("S-111");
15) std1.setSname("AAA");
16) std1.setSaddr("Hyd");
17) std1.setBranch(branch);
18) std1.getStudentDetails();
19) System.out.println();
20)
21) Student std2 = new Student();
22) std2.setSid("S-222");
23) std2.setSname("BBB");
24) std2.setSaddr("Chennai");
25) std2.setBranch(branch);
26) std2.getStudentDetails();
27) System.out.println();
28)
29) Student std3 = new Student();
30) std3.setSid("S-333");
31) std3.setSname("CCC");
32) std3.setSaddr("Pune");
33) std3.setBranch(branch);
34) std3.getStudentDetails();
35) System.out.println();
36)
37) Student std4 = new Student();
38) std4.setSid("S-444");
39) std4.setSname("DDD");
40) std4.setSaddr("Delhi");
41) std4.setBranch(branch);
42) std4.getStudentDetails();
43) }
44) }
```



## 4. Many-To-Many Associations:

It is a relationship between entities, Where multiple instances of an entity should be mapped with multiple instances of another entity.

EX: Multiple Students Have Joined with Multiple Courses.

EX:

```
1) class Course {  
2)     String cid;  
3)     String cname;  
4)     int ccost;  
5)     Course(String cid,String cname,int ccost) {  
6)         this.cid = cid;  
7)         this.cname = cname;  
8)         this.ccost = ccost;  
9)     }  
10) }  
11) class Student {  
12)     String sid;  
13)     String sname;  
14)     String saddr;  
15)     Course[] crs;  
16)     Student(String sid,String sname,String saddr,Course[] crs) {  
17)         this.sid=sid;  
18)         this.sname=sname;  
19)         this.saddr=saddr;  
20)         this.crs=crs;  
21)     }
```



```
22)     public void getStudentDetails() {
23)         System.out.println("Student Details");
24)         System.out.println("-----");
25)         System.out.println("Student Id :"+sid);
26)         System.out.println("Student name :" +sname);
27)         System.out.println("Student Address:" +saddr);
28)         System.out.println("CID CNAME CCOST");
29)         System.out.println("-----");
30)         for(int i=0;i<crs.length;i++) {
31)             Course c=crs[i];
32)             System.out.println(c.cid+" "+c cname+" "+c.ccost);
33)         }
34)         System.out.println();
35)     }
36}
37) class ManyToManyEx {
38)     public static void main(String[] args) {
39)         Course c1=new Course("C-111","C",500);
40)         Course c2=new Course("C-222","C++",1000);
41)         Course c3=new Course("C-333","JAVA",5000);
42)         Course[] crs=new Course[3];
43)         crs[0]=c1;
44)         crs[1]=c2;
45)         crs[2]=c3;
46)         Student std1=new Student("S-111","AAA","Hyd",crs);
47)         Student std2=new Student("S-222","BBB","Hyd",crs);
48)         Student std3=new Student("S-333","CCC","Hyd",crs);
49)         std1.getStudentDetails();
50)         std2.getStudentDetails();
51)         std3.getStudentDetails();
52)     }
53} }
```

## OUTPUT:

### Student Details

-----  
Student Id :S-111  
Student name :AAA  
Student Address:Hyd

CID CNAME CCOST



---

|       |      |      |
|-------|------|------|
| C-111 | C    | 500  |
| C-222 | C++  | 1000 |
| C-333 | JAVA | 5000 |

## Student Details

---

Student Id: S-222

Student name: BBB

Student Address: Hyd

| CID   | CNAME | CCOST |
|-------|-------|-------|
| C-111 | C     | 500   |
| C-222 | C++   | 1000  |
| C-333 | JAVA  | 5000  |

## Student Details

---

Student Id: S-333

Student name: CCC

Student Address: Hyd

| CID   | CNAME | CCOST |
|-------|-------|-------|
| C-111 | C     | 500   |
| C-222 | C++   | 1000  |
| C-333 | JAVA  | 5000  |

EX:

## Course.java:

```
1) package com.durgasoft.entities;
2)
3) public class Course {
4)     private String cid;
5)     private String cname;
6)     private int ccost;
7)
8)     public String getCid() {
9)         return cid;
10)    }
11)    public void setCid(String cid) {
12)        this.cid = cid;
13)    }
14)    public String getCname() {
```



```
15)    return cname;
16) }
17) public void setCname(String cname) {
18)    this.cname = cname;
19) }
20) public int getCcost() {
21)    return ccost;
22) }
23) public void setCcost(int ccost) {
24)    this.ccost = ccost;
25) }
26} }
```

## Student.java

```
1) package com.durgasoft.entities;
2)
3) public class Student {
4)    private String sid;
5)    private String sname;
6)    private String saddr;
7)    private Course[] courses;
8)
9)    public String getSid() {
10)       return sid;
11)    }
12)    public void setSid(String sid) {
13)       this.sid = sid;
14)    }
15)    public String getSname() {
16)       return sname;
17)    }
18)    public void setSname(String sname) {
19)       this.sname = sname;
20)    }
21)    public String getSaddr() {
22)       return saddr;
23)    }
24)    public void setSaddr(String saddr) {
25)       this.saddr = saddr;
26)    }
27)    public Course[] getCourses() {
28)       return courses;
29)    }
30)    public void setCourses(Course[] courses) {
```



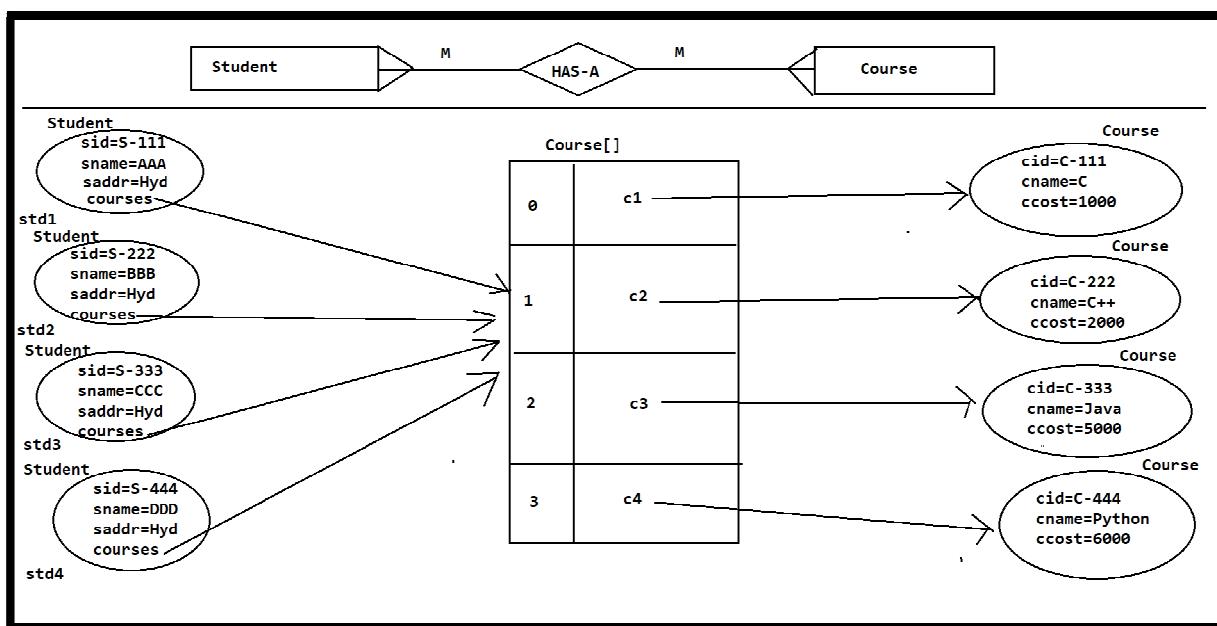
```
31)     this.courses = courses;
32) }
33) public void getStudentDetails() {
34)     System.out.println("Student Details");
35)     System.out.println("-----");
36)     System.out.println("Student Id : "+sid);
37)     System.out.println("Student Name : "+sname);
38)     System.out.println("Student Address : "+saddr);
39)     System.out.println("CID\tCNAME\tCCOST");
40)     System.out.println("-----");
41)     for(Course c: courses) {
42)         System.out.print(c.getCid()+"\t");
43)         System.out.print(c.getCname()+"\t");
44)         System.out.print(c.getCCost()+"\n");
45)     }
46)     System.out.println();
47) }
48} }
```

## Test.java

```
1) package com.durgasoft.test;
2)
3) import com.durgasoft.entities.Course;
4) import com.durgasoft.entities.Student;
5)
6) public class Test {
7)
8)     public static void main(String[] args) {
9)         Course c1 = new Course();
10)        c1.setCid("C-111");
11)        c1.setCname("C   ");
12)        c1.setCCost(1000);
13)
14)        Course c2 = new Course();
15)        c2.setCid("C-222");
16)        c2.setCname("C++  ");
17)        c2.setCCost(2000);
18)
19)        Course c3 = new Course();
20)        c3.setCid("C-333");
21)        c3.setCname("Java  ");
22)        c3.setCCost(3000);
23)
24)        Course c4 = new Course();
```



```
25) c4.setCid("C-444");
26) c4.setCname("Python");
27) c4.setCcost(1000);
28)
29) Course[] courses = {c1, c2, c3, c4};
30)
31) Student std1 = new Student();
32) std1.setSid("S-111");
33) std1.setSname("AAA");
34) std1.setSaddr("Hyd");
35) std1.setCourses(courses);
36)
37) Student std2 = new Student();
38) std2.setSid("S-222");
39) std2.setSname("BBB");
40) std2.setSaddr("Hyd");
41) std2.setCourses(courses);
42)
43) Student std3 = new Student();
44) std3.setSid("S-333");
45) std3.setSname("CCC");
46) std3.setSaddr("Hyd");
47) std3.setCourses(courses);
48)
49) Student std4 = new Student();
50) std4.setSid("S-444");
51) std4.setSname("DDD");
52) std4.setSaddr("Hyd");
53) std4.setCourses(courses);
54)
55) std1.getStudentDetails();
56) std2.getStudentDetails();
57) std3.getStudentDetails();
58) std4.getStudentDetails();
59) }
60) }
```



In java applications, Associations are existed in the following two forms.

- 1) Aggregation
- 2) Composition

## Q) What is the difference between Aggregation and Composition?

1. Where Aggregation is representing weak association that is less dependency but composition is representing strong association that is more dependency.

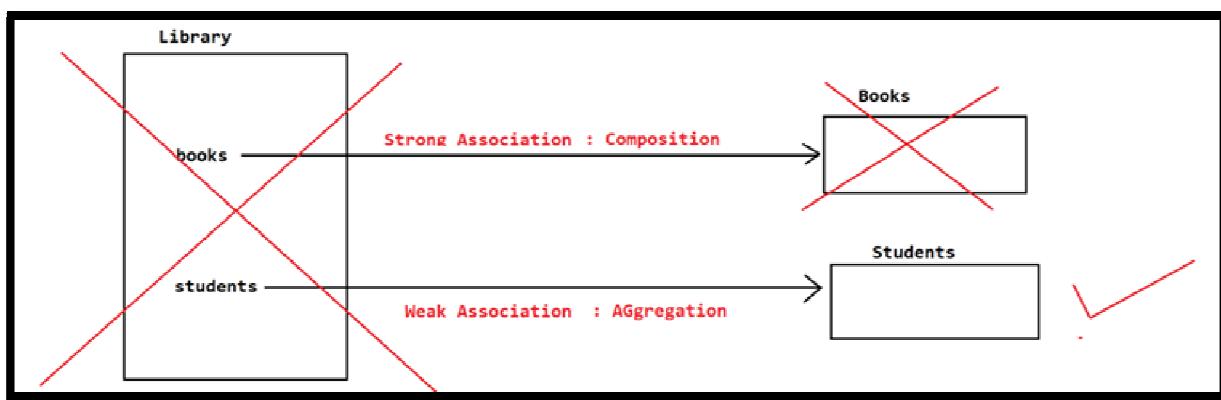
2. In case of Aggregation, if contained object is existed even without container object.  
In case of Composition, if contained object is not existed without container object.

3. In the case of Aggregation, the life of the contained Object is independent of the container Object life.

In the case of composition, the life of contained objects is depending on the container object life that is same as container object life.

### EX:

If we take an association between "Library" and "Students", "Library" and "Books". "Students" can exist without "Library", so that, the association between "Library" and "Student" is Aggregation. "Books" can not exist without "Library", so that, the association between "Library" and "Books" is composition



## Inheritance in JAVA:

The process of getting variables and methods from one class to another class is called as Inheritance.

At the basic level of Object Orientation, there are two types of inheritances.

- 1) Single Inheritance
- 2) Multiple Inheritance

## 1.Single Inheritance:

The process of getting variables and methods from only one super class to one or more number of subclasses is called as Single Inheritance.

Java is able to allow Single Inheritance.

## 2.Multiple Inheritance:

The process of getting variables and methods from more than one super class to one or more number of sub classes is called as Multiple Inheritance.

Java is not allowing Multiple Inheritance.

On the basis of Single and Multiple inheritances, there are 3 more Inheritances.

- 1) Multi-Level Inheritance
- 2) Hierarchical Inheritance
- 3) Hybrid Inheritance



## **1. Multi-Level Inheritance:**

It is a Combination of single inheritance in more than one level. Java is able to allow multi-level inheritance.

EX:

```
class A {  
}  
class B extends A {  
}  
class C extends B {  
}
```

## **2. Hierarchical Inheritance:**

It is the combination of single inheritance in a particular structure. Java is able to allow Hierarchical inheritance.

EX:

```
class A {  
}  
class B extends A {  
}  
class C extends A {  
}  
class D extends B {  
}  
class E extends B {  
}  
class F extends C {  
}  
class G extends C {  
}
```

## **3. Hybrid Inheritance:**

It is the combination of single inheritance and multiple inheritances.

Java is not allowing Hybrid Inheritance.

EX:

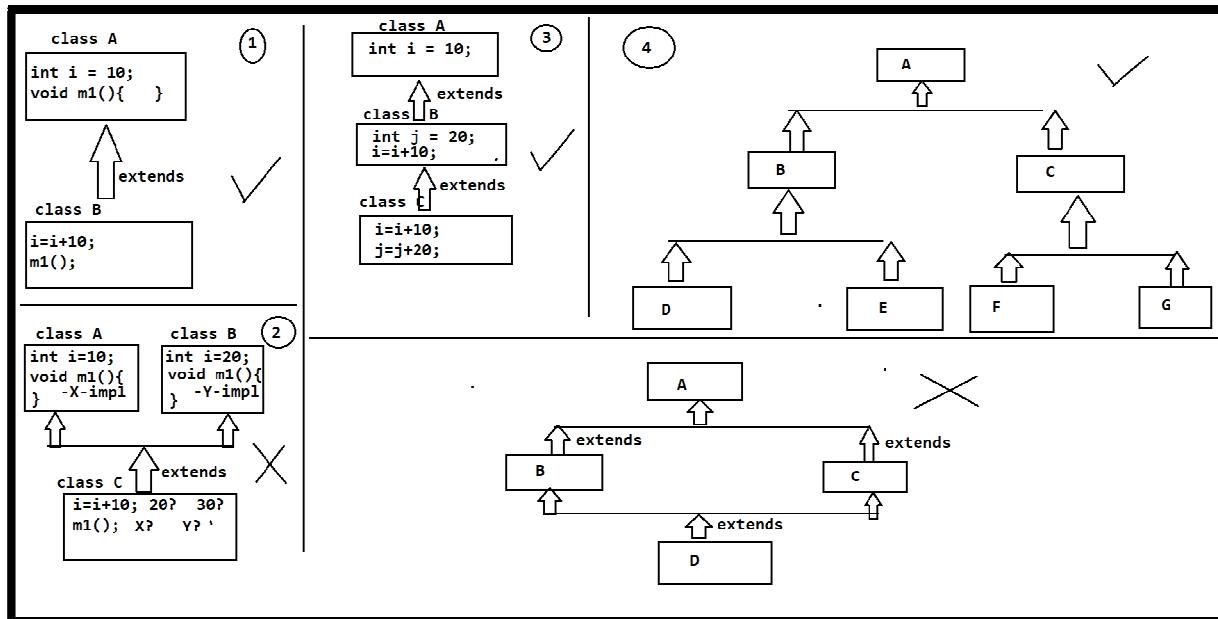
```
class A {  
}  
class B extends A {  
}  
class C extends A {  
}
```



```
}
```

```
class D extends B, C {
```

```
}
```



**EX:**

```
1) class Employee {  
2)     String eid;  
3)     String ename;  
4)     String eaddr;  
5)     public void getEmpDetails() {  
6)         System.out.println("Employee Id :" + eid);  
7)         System.out.println("Employee name :" + ename);  
8)         System.out.println("Employee Address :" + eaddr);  
9)     }  
10}   
11 class Manager extends Employee {  
12     Manager(String eid1, String ename1, String eaddr1) {  
13         eid = eid1;  
14         ename = ename1;  
15         ename = eaddr1;  
16     }  
17     public void getManagerDetails() {  
18         System.out.println("manager Details");  
19         System.out.println("-----");  
20         getEmpDetails();  
21     }  
22}
```



```
23) class Accountant extends Employee {  
24)     Accountant (String eid1, String ename1, String eaddr1) {  
25)         eid = eid1;  
26)         ename = ename1;  
27)         eaddr = eaddr1;  
28)     }  
29)     public void getAccountantDetails() {  
30)         System.out.println("Accountant Details");  
31)         System.out.println("-----");  
32)         getEmpDetails();  
33)     }  
34) }  
35) class InheritanceEx {  
36)     public static void main(String[] args) {  
37)         Manager m = new Manager("E-111","AAA","Hyd");  
38)         m.getManagerDetails();  
39)         System.out.println();  
40)         Accountant acc = new Accountant ("E-222","BBB","Hyd");  
41)         acc.getAccountantDetails();  
42)     }  
43) }
```

### On Command Prompt:

D:\java7>javac InheritanceEx.java  
D:\java7>java Inheritance Ex

### Manager Details:

Employee Id: E-111  
Employee Name: AAA  
Employee Address: Hyd

### Accountant Details:

Employee Id: E-222  
Employee Name: BBB  
Employee Address: Hyd

### Static Context in Inheritance:

In the case of inheritance, if we create an object for sub class then JVM has to execute sub class constructor, before executing sub class constructor, JVM has to check whether sub class bytecode is loaded already in the memory or not, if not, JVM has to load subclass bytecode to the memory.

In the above context, before loading sub class bytecode to the memory, first, JVM has to load the respective super class bytecode to the memory.



Therefore, in the case of inheritance, JVM will load all the classes bytecode right from super class to sub class order.

If we provide static context in both super class and subclass then JVM will recognize and execute static context of the respective classes at the time of loading the respective classes, that is from super class to sub class ordering.

EX:

```
1) class A {  
2)     static {  
3)         System.out.println("SB-A");  
4)     }  
5) }  
6) class B extends A {  
7)     static {  
8)         System.out.println("B-con");  
9)     }  
10}  
11) class C extends B {  
12)     static {  
13)         System.out.println("SB-C");  
14)     }  
15)}  
16) class Test {  
17)     public static void main(String[] args) {  
18)         C c=new C();  
19)     }  
20)}
```

OUTPUT:

SB-A  
B-con  
SB-C

EX:

```
1) class A {  
2)     static {  
3)         System.out.println("SB-A");  
4)     }  
5)     static int m10 {  
6)         System.out.println("m1-A");  
7)         return 10;  
8)     }  
9)     static int i=m10;
```



```
10)
11) class B extends A {
12)     static int j=m20;
13)     static {
14)         System.out.println("SB-B");
15)     }
16)     static int m20 {
17)         System.out.println("m2-B");
18)         return 20;
19)     }
20}
21) class C extends B {
22)     static int m30 {
23)         System.out.println("m3-C");
24)         return 30;
25)     }
26)     static int k=m30;
27)     static {
28)         System.out.println("SB-C");
29)     }
30}
31) class Test {
32)     public static void main(String[] args) {
33)         C c1=new C();
34)         C c2=new C();
35)     }
36}
```

OUTPUT: SB-A  
m1-A  
m2-B  
SB-B  
m3-C  
SB-C

## Instance Context in Inheritance:

In the case of Inheritance, if we create object for sub class then JVM has to execute sub class constructor, but before executing sub class constructor JVM has to execute 0-argument constructor in the respective super class. If we provided instance context in both super class and sub class then JVM will execute the provided instance context just before executing the respective class constructor.



EX:

```
1) class A {  
2)     A0 {  
3)         System.out.println("A-con");  
4)     }  
5) }  
6) class B extends A {  
7)     B0 {  
8)         System.out.println("B-con");  
9)     }  
10) }  
11) class C extends B {  
12)     C0 {  
13)         System.out.println("C-con");  
14)     }  
15) }  
16) class Test {  
17)     public static void main(String[] args) {  
18)         C c = new C0;  
19)     }  
20) }
```

OUTPUT: A-Con

B-Con

C-Con

EX:

```
1) class A {  
2)     A0 {  
3)         System.out.println("A-con");  
4)     }  
5) }  
6) class B extends A {  
7) }  
8) class C extends B {  
9)     C0 {  
10)         System.out.println("C-con");  
11)     }  
12) }  
13) class Test {  
14)     public static void main(String[] args) {  
15)         C c = new C0;  
16)     }  
17) }
```



**OUTPUT:** A-Con  
C-Con

**EX:**

```
1) class A {  
2)     A(int i) {  
3)         System.out.println("A-int-param-con");  
4)     }  
5) }  
6) class B extends A {  
7)     B(int i) {  
8)         System.out.println("B-int-param-con");  
9)     }  
10) }  
11) class C extends B {  
12)     C(int i) {  
13)         System.out.println("C-int-param-con");  
14)     }  
15) }  
16) class Test {  
17)     public static void main(String[] args) {  
18)         C c = new C(10);  
19)     }  
20) }
```

**Status:** Compilation Error

**Reason:** In case of Inheritance, super classes must have 0-argument constructors irrespective of the sub class constructors.

In the case of Inheritance, if we provide instance context at all the classes then JVM will recognize and execute them just before executing the respective class constructors.

**EX:**

```
1) class A {  
2)     A0 {  
3)         System.out.println("A-con");  
4)     }  
5)     int i = m10;  
6)     int m10 {  
7)         System.out.println("m1-A");  
8)         return 10;  
9)     }
```



```
10)  {
11)      System.out.println("IB-A");
12)  }
13) }
14) class B extends A {
15)     int j = m2();
16)     int m2() {
17)         System.out.println("m2-B");
18)         return 20;
19)     }
20)     {
21)         System.out.println("IB-B");
22)     }
23)     B() {
24)         System.out.println("B-Con");
25)     }
26) }
27) class C extends B {
28)     C() {
29)         System.out.println("C-con");
30)     }
31)     {
32)         System.out.println("IB-C");
33)     }
34)     int k = m3();
35)     int m3() {
36)         System.out.println("m3-C");
37)         return 30;
38)     }
39) }
40) class Test {
41)     public static void main(String args[]) {
42)         C c = new C();
43)     }
44} }
```

## OUTPUT:

m1-A  
IB-A  
A-con  
m2-B  
IB-B  
B-Con  
IB-C  
m3-C  
c-con



EX:

```
1) class A {  
2)     A() {  
3)         System.out.println("A-con");  
4)     }  
5)     static {  
6)         System.out.println("SB-A");  
7)     }  
8)     int m1() {  
9)         System.out.println("m1-A");  
10)        return 10;  
11)    }  
12)    static int m2() {  
13)        System.out.println("m2-A");  
14)        return 20;  
15)    }  
16)    {  
17)        System.out.println("IB-A");  
18)    }  
19)    static int i=m2();  
20)    int j=m1();  
21}  
22) class B extends A {  
23)    {  
24)        System.out.println("IB-B");  
25)    }  
26)    int m3(){  
27)        System.out.println("m3-B");  
28)        return 30;  
29)    }  
30)    static {  
31)        System.out.println("SB-B");  
32)    }  
33)    int k=m3();  
34)    B() {  
35)        System.out.println("B-Con");  
36)    }  
37)    static int l=m4();  
38)    static int m4(){  
39)        System.out.println("m4-B");  
40)        return 40;  
41}  
42) class C extends B {  
43)    static int m5() {  
44)        System.out.println("m5-C");
```



```
45)     return 50;
46) }
47) int m6() {
48)     System.out.println("m6-C");
49)     return 60;
50) }
51) C0 {
52)     System.out.println("C-con");
53) }
54) int m6();
55) static int n=m5() {
56)     System.out.println("IB-C");
57) }
58) static {
59)     System.out.println("SB-C");
60) }
61)
62) class Test {
63)     public static void main(String args[]) {
64)         C c1=new C0;
65)         C c2=new C0;
66)     }
67} }
```

## Super Keyword:

Super is a Java keyword, it can be used to represent super class object from sub classes.

There are three ways to utilize "super" keyword.

- 1) To refer super class variables
- 2) To refer super class constructors
- 3) To refer super class methods.

### 1.To refer super class variables:

If we want to refer super class variables by using 'super' keyword then we have to use the following syntax.

super.var\_Name;

**NOTE:** We will utilize super keyword, to access super class variables when we have same set of variables at local, at current class and at the super class.



EX:

```
1) class A {  
2)     int i=100;  
3)     int j=200;  
4) }  
5) class B extends A {  
6)     int i=10;  
7)     int j=20;  
8)     B(int i, int j) {  
9)         System.out.println(i+ " " +j);  
10)        System.out.println(this.i+ " " +this.j);  
11)        System.out.println(super.i+ " " +super.i);  
12)    }  
13) }  
14) class Test {  
15)     public static void main(String args[]) {  
16)         B b = new B(50,60);  
17)     }  
18} }
```

OUTPUT:

```
50 60  
10 20  
100 100
```

## 2. To refer super class constructors:

If we want to refer super class constructor from sub class by using 'super' keyword then we have to use the following syntax.

```
super([Param_List]);
```

**NOTE:** In general, in inheritance, JVM will execute 0-argument constructor before executing sub class constructor. In this context, instead of executing 0-argument constructor we want to execute a parameterized constructor at super class, for this, we have to use 'super' keyword.

EX:

```
1) class A {  
2)     A0 {  
3)         System.out.println("A-Con");  
4)     }  
5)     A(int i) {
```



```
6)     System.out.println("A-int-param-Con");
7) }
8) }
9) class B extends A {
10) B() {
11)     super(10);
12)     System.out.println("B-Con");
13) }
14)
15) class Test {
16)     public static void main(String args[]) {
17)         B b = new B();
18)     }
19} }
```

## OUTPUT:

A-int-param-Con  
B-Con

If we want to access super class constructor from subclass by using "super" keyword then the respective "super" statement must be provided as first statement.

If we want to access super class constructor from sub class by using "super" keyword then the respective "super" statement must be provided in the subclass constructors only, not in subclass normal Java methods.

If we violate any of the above conditions then compiler will rise an error like "call to super must be first statement in constructor".

**NOTE:** Due to the above rules and regulations, it is not possible to access more than one super class constructor from a single sub class constructor by using "super" keyword.

In the case of inheritance, when we access sub class constructor then, first, JVM will execute super class 0-arg constructor then JVM will execute sub class constructor, this flow of execution is possible in Java because of the following compiler actions over source code at the time of compilation.

a) Compiler will go to each and every class available in the source file and checks whether any requirement to provide "default constructors".

b) If any class is identified without user defined constructor explicitly then compiler will add a 0-argument constructor as default constructor.

c) After providing default constructors, compiler will go to all the constructors at each and every class and compiler will check "super" statement is provided or not by the developer explicitly to access super class constructor.



d) If any class constructor is identified with out "super" statement explicitly then compiler will append "super()" statement in the respective constructor to access a 0- argument constructor in the respective super class.

e) With the above compiler actions ,JVM will execute super class 0-arg constructor as part of executing sub class constructor explicitly.

Write the translated code:

```
1) class A {  
2)     A0 {  
3)         System.out.println("A-con");  
4)     }  
5) }  
6) class B extends A {  
7)     B(int i) {  
8)         System.out.println("B-int-param-con");  
9)     }  
10}   
11) class C extends B {  
12)     C(int i) {  
13)         System.out.println("C-int-param-con");  
14)     }  
15}   
16) class Test {  
17)     public static void main(String args[]) {  
18)         C c = new C();  
19)     }  
20} 
```

Status: Compilation Error

EX:

```
1) class A {  
2)     A0 {  
3)         System.out.println("A-con");  
4)     }  
5) }  
6) class B extends A {  
7)     B(int i) {  
8)         System.out.println("B-int-param-con");  
9)     }  
10}   
11) class C extends B {  
12)     C(int i) {  
13)         super(10);  
14)     }  
15} 
```



```
14)         System.out.println("C-int-param-con");
15)     }
16)
17) class Test {
18)     public static void main(String args[]) {
19)         C c = new C(10);
20)     }
21) }
```

**OUTPUT:**  
A-con  
B-int-param-con  
C-int-param-con

### 3.To refer super class method:

If we want to refer super class methods from sub class by using "super" keyword then we have to use the following syntax.

super.method\_Name([Param\_List]);

**NOTE:** In Java applications, when we have same method at both subclass and at super class, if we access that method at sub class then JVM will execute sub class method only, not super class method because JVM will give more priority for the local class methods. In this context, if we want to refer super class method over sub class method then we have to "super" keyword.

**EX:**

```
1) class A {
2)     void m1() {
3)         System.out.println("m1-A");
4)     }
5) }
6) class B extends A {
7)     void m2() {
8)         System.out.println("m2-B");
9)         m1();
10)        this.m1();
11)        super.m1();
12)    }
13) }
14) void m1() {
15)     System.out.println("m1-B");
16) }
17) class Test {
18)     public static void main(String args[]) {
19)         B b=new B();
```



```
20)      b.m2();
21)
22)}
```

## Class Level Type Casting:

The process of converting the data from one user defined data type to another user defined data type is called as User Defined Data type casting or Class level type Casting. If we want to perform Class Level Type Casting or User Defined data types casting then we must require either "extends" or "implements" relationship between two user defined data types.

There are two types of User defined data type casting:

- 1) UpCasting.
- 2) DownCasting.

### 1.UpCasting:

The process of converting the data from sub type to super type is called as UpCasting. To perform Upcasting, we have to assign sub class reference variable to super class reference variable.

EX:

```
1) class A {
2)
3) class B extends A {
4)
5) B b = new B();
6) A a = b;
```

If we compile the above code then compiler will check whether 'b' variable data type is compatible with 'a' variable data type or not, if not, compiler will rise an error like "InCompatible Types". If "b" variable data type id compatible to "a" variable data type then compiler will not rise any error.

**NOTE:** In Java applications, always sub class types are compatible with super class types, so that we can assign sub class reference variable to super class reference variables directly.

**NOTE:** In Java, super class types are not compatible with sub class types, so that, we cannot assign super class reference variables to sub class reference variables directly, where if we want to assign super class reference variables to sub class reference variables then we must require "cast operator" explicitly, that is called as "Explicit Type Casting".



If we execute the above code then JVM will perform the following two actions.

- 1) JVM will convert "b" variable data type [sub class type] to "a" variable data type [Super class type] implicitly.
- 2) JVM will copy the reference value of "b" variable to "a" variable.

With the upcasting, we are able to access only super class members among the availability of both super class and sub class members in sub class object.

```
1) class A {  
2)     void m10 {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class B extends A {  
7)     void m20 {  
8)         System.out.println("m2-B");  
9)     }  
10}  
11) class Test {  
12)     public static void main(String[] args) {  
13)         B b=new B();  
14)         b.m10;  
15)         b.m20;  
16)         A a=b;  
17)         a.m10;  
18)         //a.m20;---->error  
19)     }  
20} 
```

OUTPUT: m1-A  
m2-B  
m1-A

## 2.DownCasting:

The process of converting the data from super type to sub type is called as "DownCasting".

If we want to perform DownCasting then we have to use the following format.

EX:

```
1) class A {  
2)     void m10 {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class B extends A { 
```



```
7)     void m2() {
8)         System.out.println("m2-B");
9)     }
10) }
11) class Test {
12)     public static void main(String args[]) {
13)         case 1:
14)             A a = new A();
15)             B b = a;
```

Status: Compilation error, incompatible types

**Reason:** In Java, always, subclass types are compatible with super class types but super class types are not compatible with sub class types. It is possible to assign sub class reference variables directly but it is not possible to assign super class reference variable to sub class reference variables directly, if we assign then compiler will rise an error like "InCompatible Types error".

### Case 2:

```
A a=new A();
B b=(B)a;
```

Status: No Compilation error, but classCastException

**Reason:** In Java, always, it is possible to keep subclass object reference value in super class reference variable but it is not possible to keep super class object reference value in subclass reference variable. If we are trying to keep super class object reference value in sub class reference variable then JVM will rise an exception like "java.lang.classCastException".

### Case 3:

```
A a = new B();
B b = (B)a;
```

Status: No compilation error, no exception

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

### EX:

```
class A {
}
class B extends A {
}
```



---

```
class C extends B {  
}  
class D extends C {  
}
```

**Ex1:**

```
A a=new A();  
B b=a;  
Status:Compilation error,incompatible types
```

**Ex2:**

```
A a=new A();  
B b=(B)a;  
Status>No Compilation error,ClassCastException
```

**Ex3:**

```
A a=new A();  
B b=(B)a;  
Status>No Compilation error,No Exception
```

**Ex4:**

```
A a=new C();  
B b=(C)a;  
Status>No Compilation error,No Exception
```

**Ex5:**

```
A a=new B();  
B b=(C)a;  
Status>No Compilation Error;ClassCastException
```

**Ex6:**

```
A a=new C();  
C c=(D)a;  
Status>No Compilation Error,ClassCastException
```

**Ex7:**

```
B b=new D();  
C c=(D)b;  
Status:NO Compilation Error,No Exception
```

**Ex8:**

```
A a=new D();  
D d=(D)(C)(B)a;  
Status>No Compilation error,No Exception
```



## Ex9:

```
A a=new C();  
D d=(D)(C)(B)a;  
Status:No Compilation Error,ClassCastException
```

## Ex10:

```
A a=new C();  
C c=(D)(C)(B)a;  
Status: No Compilation Error, ClassCastException
```

## USES-A Relationship:

This is a relationship between entities, where one entity will use another entity up to a particular action or behavior or method.

To provide USES-A Relationship in java applications, we have to provide contained Entity class reference variable as parameter to a method in Container entity class, not as class level variable.

If we declared contained entity reference variable as class level reference variable in container entity class then that relationship is "HAS-A" relationship.

## EX:

```
1) class Account {  
2)     String accno;  
3)     String accName;  
4)     String accType;  
5)     int bal = 10000;  
6)     Account(String accNo, String accName, String accType) {  
7)         this.accNo = accNo;  
8)         this.accName = accName;  
9)         this.accType = accType;  
10)    }  
11) }  
12) class Transaction {  
13)     String tx_tid;  
14)     String tx_Type;  
15)     Transaction(String tx_id, String tx_Type) {  
16)         this.tx_Id = tx_Id;  
17)         this.tx_Type = tx_Type;  
18)    }  
19)     public void deposit(Account acc, int dep_Amt) {  
20)         int initial_Amt = acc.bal;  
21)         int total_Avl_Amt = initial_Amt+dep_Amt;  
22)         acc.bal = total_Avl_Amt;
```



```
23)         System.out.println("Transaction Details");
24)         System.out.println("-----");
25)         System.out.println("Transaction Id :" + tx_Id);
26)         System.out.println("Account Number :" + acc.accNo);
27)         System.out.println("Account Type :" + acc.accType);
28)         System.out.println("Initial Amount :" + initial_Amt);
29)         System.out.println("Deposit Amount :" + dep_Amt);
30)         System.out.println("Total Avl Amount :" + total_Avl_Amt);
31)         System.out.println("Transaction Status:SUCCESS");
32)         System.out.println("*****THANKQ,VISIT AGAIN*****");
33)     }
34)
35) class UsesAEx {
36)     public static void main(String[] args) {
37)         Account acc = new Account("abc123", "Durga", "Savings");
38)         Transaction tx = new Transaction("T-111", "Deposit");
39)         tx.deposit(acc, 5000);
40)     }
41} 
```

## OUTPUT:

Transaction Details

Transaction Id :T-111  
Account Number :abc123  
Account Type :Savings  
Initial Amount :10000  
Deposit Amount :5000  
Total Avl Amount :15000  
Transaction Status:SUCCESS

## Polymorphism:

- Polymorphism is a Greek word, where poly means many and morphism means Structures.
- If one thing is existed in more than one form then it is called as Polymorphism.
- The main advantage of Polymorphism is "Flexibility" to design Applications.

There are 2 types of Polymorphisms

- 1) Static Polymorphism or Early binding
- 2) Dynamic Polymorphism or Late Binding



## 1. Static Polymorphism:

If the Polymorphism is existed at compilation time then that Polymorphism is called as Static Polymorphism.

EX: Method Overloading

## 2. Dynamic Polymorphism:

If the Polymorphism is existed at runtime then that Polymorphism is called as Dynamic Polymorphism.

EX: Method Overriding.

### Method Overloading:

- The process of extending the existed method functionality upto some new Functionality is called as Method Overloading.
- If we declare more than one method with the same name and with the different parameter list is called as Method Overloading.
- To perform method overloading, we have to declare more than one method with different method signatures that is same method name and different parameter list.

EX:

```
1) class A {  
2)     void add(int i,int j) {  
3)         System.out.println(i+j);  
4)     }  
5)     void add(float f1,float f2) {  
6)         System.out.println(f1+f2);  
7)     }  
8)     void add(String str1,String str2) {  
9)         System.out.println(str1+str2);  
10)    }  
11) }  
12) class Test {  
13)     public static void main(String[] args) {  
14)         A a=new A();  
15)         a.add(10,20);  
16)         a.add(22.22f,33.33f);  
17)         a.add("abc","def");  
18)     }  
19) }
```

### OUTPUT:

30

55.550003



abcdef

EX:

```
1) class Employee {  
2)     void gen_Salary(int basic, float hk, float pf, int ta) {  
3)         float salary = basic+((basic*hk)/100)-((basic*pf)/100)+ta);  
4)         System.out.println("Salary :" +salary);  
5)     }  
6)     void gen_Salary(int basic, float hk, float pf, int ta, int bonus) {  
7)         float salary = basic+((basic*hk)/100)-((basic*pf)/100)+ta)+bonus;  
8)         System.out.println("Salary :" +salary);  
9)     }  
10}   
11) class Test {  
12)     public static void main(String[] args) {  
13)         Employee e = new Employee();  
14)         e.gen_Salary(20000,25.0f,12.0f,2000);  
15)         e.gen_Salary(20000,25.0f,12.0f,2000,5000);  
16)     }  
17} }
```

## Method Overriding:

- The process of replacing existed method functionality with some new functionality is called as Method Overriding.
- To perform Method Overriding, we must have inheritance relationship classes.
- In Java applications, we will override super class method with sub class method.
- In Java applications, we will override super class method with subclass method.
- If we want to override super class method with sub class method then both super class method and sub class method must have same method prototype.

## Steps to perform Method Overriding:

- 1) Declare a super class with a method which we want to override.
- 2) Declare a sub class and provide the same super class method with different implementation.
- 3) In main class, in main() method, prepare object for sub class and prepare reference variable for super class [UpCasting].
- 4) Access super class method then we will get output from sub class method.



EX:

```
1) class Loan {  
2)     public float getIR() {  
3)         return 7.0f;  
4)     }  
5) }  
6) class GoldLoan extends Loan {  
7)     public float getIR() {  
8)         return 10.5f;  
9)     }  
10) }  
11) class StudyLoan extends Loan {  
12)     public float getIR() {  
13)         return 12.0f;  
14)     }  
15) }  
16) class CraftLoan extends Loan {  
17) }  
18) class Test {  
19)     public static void main(String[] args) {  
20)         Loan gold_Loan=new GoldLoan();  
21)         System.out.println("Gold Loan IR :" +gold_Loan.getIR()+"%");  
22)         Loan study_Loan=new StudyLoan();  
23)         System.out.println("Study Loan IR :" +study_Loan.getIR()+"%");  
24)         Loan craft_Loan=new CraftLoan();  
25)         System.out.println("Craft Loan IR :" +craft_Loan.getIR()+"%");  
26)     }  
27) }
```

**NOTE:** To prove method overriding in Java, we have to access super class method but JVM will execute the respective sub class method and JVM has to provide output from the respective sub class method, not form super class method. To achieve the above requirement we must create reference variable for only super class and we must create object for sub class.

```
1) class A {  
2)     void m10 {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class B extends A {  
7)     void m10 {  
8)         System.out.println("m1-B");  
9)     }  
10) }
```



```
11) class Test {  
12)     public static void main(String args[]) {  
13)         /* A a=new A();  
14)             a.m10;  
15) Status:Here Method Overriding is not happened, because Method overriding  
16) must require sub class object, not super class object.  
17)         */  
18)         /*  
19)             B b = new B();  
20)             b.m10;  
21) Status:Here method Overriding is happened, but,to prove method  
overriding we must require super class reference variable,not sub class reference  
variable,because,subclass reference variable is able to access only sub class  
method when we have the same method in both sub class and super class.  
22)         */  
23)         A a = new B();  
24)         a.m10;  
25)     }  
26) }
```

## Rules to perform Method Overriding:

1. To override super class method with sub class then super class method must not be declared as private.

EX:

```
1) class A {  
2)     private void m10 {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class B extends A {  
7)     void m10 {  
8)         System.out.println("m1-B");  
9)     }  
10}   
11) class Test {  
12)     public static void main(String args[]) {  
13)         A a = new A();  
14)         a.m10;  
15)     }  
16) }
```



2. To override super class method with sub class method then sub class method should have the same return type of the super class method.

EX:

```
1) class A {  
2)     int m10 {  
3)         System.out.println("m1-A");  
4)         return 10;  
5)     }  
6) }  
7) class B extends A {  
8)     void m10 {  
9)         System.out.println("m1-B");  
10)    }  
11) }  
12) class Test {  
13)     public static void main(String args[]) {  
14)         A a=new B();  
15)         a.m10;  
16)     }  
17) }
```

3. To override super class method with sub class method then super class method must not be declared as final sub class method may or may not be final.

EX:

```
1) class A {  
2)     void m10 {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class B extends A {  
7)     final void m10 {  
8)         System.out.println("m1-B");  
9)     }  
10) }  
11) class Test {  
12)     public static void main(String[] args) {  
13)         A a = new B();  
14)         a.m10;  
15)     }  
16) }
```



4. To override super class method with sub class method either super class method or subclass method as static then compiler will rise an error. If we declare both super and sub class method as static in method overriding compiler will not rise any error, JVM will provide output from the super class method.

**NOTE:** If we are trying to override super class static method with sub class static method then super class static method will override subclass static method, where JVM will generate output from super class static method.

**EX:**

```
1) class A {  
2)     static void m10 {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class B extends A {  
7)     static void m10 {  
8)         System.out.println("m1-B");  
9)     }  
10}  
11) class Test {  
12)     public static void main(String args[]) {  
13)         A a = new B();  
14)         a.m10;  
15)     }  
16} }
```

5. To override super class method with subclass method, sub class method must have either same scope of the super class method or more scope when compared with super class method scope otherwise compiler will rise an error.

**EX:**

```
1) class A {  
2)     protected void m10 {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class B extends A {  
7)     public void m10 {  
8)         System.out.println("m1-B");  
9)     }  
10}  
11) class Test {  
12)     public static void main(String args[]) {  
13)         A a = new A();
```



```
14)     a.m10();
15) }
16) }
```

6. To override super class method with subclass method subclass method should have either same access privileges or weaker access privileges when compared with super class method access privileges.

## Q) What are the differences between Method Overloading and Method Overriding?

- 1) The process of extending the existed method functionality with new functionality is called as Method Overloading.  
The process of replacing existed method functionality with new functionality is called as Method Overriding.
- 2) In the case of method overloading, different method signatures must be provided to the methods  
In the case of method overriding same method prototypes must be provided to the methods.
- 3) With OR without inheritance we can perform method overloading  
With inheritance only we can perform Method overriding

### EX:

```
1) class DB_Driver {
2)     public void getDriver() {
3)         System.out.println("Type-1 Driver");
4)     }
5) }
6) class New_DB_Driver extends DB_Driver {
7)     public void getDriver() {
8)         System.out.println("Type-4-Driver");
9)     }
10)
11) class Test {
12)     public static void main(String args[]) {
13)         DB_Driver driver = new New_DB_Driver();
14)         driver.getDriver();
15)     }
16) }
```

In the above example, method overriding is implemented, in method overriding, for super class method call JVM has to execute subclass method, not super class method. In method overriding, always JVM is executing only subclass method, not super class method. In



method overriding, it is not suggestible to manage super class method body without execution, so that, we have to remove super class method body as part of code optimization. In Java applications, if we want to declare a method without body then we must declare that method as "Abstract Method".

If we want to declare abstract methods then the respective class must be abstract class.

```
1) abstract class DB_Driver {  
2)     public abstract void getDriver();  
3) }  
4) class New_DB_Driver extends DB_Driver {  
5)     public void getDriver() {  
6)         System.out.println("Type-4 Driver");  
7)     }  
8) }  
9) class Test {  
10)    public static void main(String args[]) {  
11)        DB_Driver driver = new New_DB_Driver();  
12)        driver.getDriver();  
13)    }  
14) }
```

In Java applications, if we declare any abstract class with abstract methods, then it is convention to implement all the abstract methods by taking sub class.

To access the abstract class members, we have to create object for subclass and we have to create reference variable either for abstract class or for subclass.

If we create reference variable for abstract class then we are able to access only abstract class members, we are unable to access subclass own members

If we declare reference variable for subclass then we are able to access both abstract class members and subclass members.

```
1) abstract class A {  
2)     void m1() {  
3)         System.out.println("m1-A");  
4)     }  
5)     abstract void m2();  
6)     abstract void m3();  
7) }  
8) class B extends A {  
9)     void m2() {  
10)        System.out.println("m2-B");  
11)    }  
12)    void m3() {  
13)        System.out.println("m3-B");  
14)    }  
15)    void m4() {
```



```
16)     System.out.println("m4-B");
17) }
18)
19) class Test {
20)     public static void main(String args[]) {
21)         A a = new B();
22)         a.m1();
23)         a.m2();
24)         a.m3();
25)         //a.m4();---error
26)         B b = new B();
27)         b.m1();
28)         b.m2();
29)         b.m3();
30)         b.m4();
31)     }
32) }
```

In Java applications, it is not possible to create Object from abstract classes but it is possible to provide constructors in abstract classes, because, to recognize abstract class instance variables in order to store in the sub class objects.

EX:

```
1) abstract class A {
2)     A() {
3)         System.out.println("A-Con");
4)     }
5) }
6) class B extends A {
7)     B() {
8)         System.out.println("B-Con");
9)     }
10) }
11) class Test {
12)     public static void main(String[] args) {
13)         B b = new B();
14)     }
15) }
```

In Java applications, if we declare any abstract class with abstract methods then it is mandatory to implement all the abstract methods in the respective subclass. If we implement only some of the abstract methods in the respective subclass then compiler will raise an error, where to come out from compilation error we have to declare the respective subclass as an abstract class and we have to provide implementation for the remaining abstract methods by taking another subclass in multilevel inheritance.



EX:

```
1) abstract class A {  
2)     abstract void m1();  
3)     abstract void m2();  
4)     abstract void m3();  
5) }  
6) abstract class B extends A {  
7)     void m1() {  
8)         System.out.println("m1-A");  
9)     }  
10) }  
11) class C extends B {  
12)     void m2() {  
13)         System.out.println("m2-C");  
14)     }  
15)     void m3() {  
16)         System.out.println("m3-C");  
17)     }  
18) }  
19) class Test {  
20)     public static void main(String[] args) {  
21)         A a = new C();  
22)         a.m1();  
23)         a.m2();  
24)         a.m3();  
25)     }  
26) }
```

In Java applications, if we want to declare an abstract class then it is not at all mandatory condition to have at least one abstract method, it is possible to declare abstract class without having abstract methods but if we want to declare a method as an abstract method then the respective class must be abstract class.

```
1) abstract class A {  
2)     void m1() {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class B extends A {  
7)     void m2() {  
8)         System.out.println("m2-B");  
9)     }  
10) }  
11) class Test {  
12)     public static void main(String args[]) {
```



```
13) A a = new B();  
14) a.m1();  
15) //a.m2();----->Error  
16) B b = new B();  
17) b.m1();  
18) b.m2();  
19) }  
20} }
```

In Java applications, it is possible to extend an abstract class to concrete class and from concrete class to abstract class.

```
1) class A {  
2)     void m1() {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) abstract class B extends A {  
7)     abstract void m2();  
8) }  
9) class C extends B {  
10)    void m2() {  
11)        System.out.println("m2-C");  
12)    }  
13)    void m3() {  
14)        System.out.println("m3-C");  
15)    }  
16} }  
17) class Test {  
18)     public static void main(String args[]) {  
19)         A a = new C();  
20)         a.m1();  
21)         //a.m2();---◊error  
22)         //a.m3();---◊error  
23)         B b = new C();  
24)         b.m1();  
25)         b.m2();  
26)         //b.m3();---◊error  
27)         C c = new C();  
28)         c.m1();  
29)         c.m2();  
30)         c.m3();  
31)     }  
32} }
```



**Note:** In Java applications, it is not possible to extend a class to the same class, if we do the same then compiler will rise an error like "cyclic inheritance involving".

```
class A extends A{  
}
```

Status: Compilation Error: "cyclic inheritance involving".

```
class A extends B{  
}
```

```
class B extends A{  
}
```

Status: Compilation Error: "cyclic inheritance involving".

## Interfaces:

Interface is a Java Feature, it will allow only abstract methods.

In Java applications, for interfaces, we are able to create only reference variables, we are unable to create objects.

In the case of interfaces, by default, all the variables are "public static final".

In the case of interfaces, by default, all the methods are "public and abstract".

In Java applications, constructors are possible in classes and abstract classes but constructors are not possible in interfaces.

Interfaces will provide more sharability in Java applications when compared with classes and abstract classes.

In Java applications, if we declare any interface with abstract methods then it is convention to declare an implementation class for the interface and it is convention to provide implementation for all the abstract methods in implementation class.

```
1) interface I {  
2)     void m1();  
3)     void m2();  
4)     void m3();  
5) }  
6) class A implements I {  
7)     public void m1() {  
8)         System.out.println("m1-A");  
9)     }  
10)    public void m2() {
```



```
11)     System.out.println("m2-A");
12) }
13) public void m3() {
14)     System.out.println("m3-A");
15) }
16) public void m4() {
17)     System.out.println("m4-A");
18) }
19)
20) class Test {
21)     public static void main(String args[]) {
22)         I i = new A();
23)         i.m1();
24)         i.m2();
25)         i.m3();
26)         //i.m4();---->error
27)         A a = new A();
28)         a.m1();
29)         a.m2();
30)         a.m3();
31)         a.m4();
32)     }
33} }
```

In Java applications, if we declare an interface with abstract methods then it is mandatory to provide implementation for all the abstract methods in the respective implementation class. In this context if we provide implementation for some of the abstract methods at the respective implementation class then compiler will rise an error, where to come out from the compilation error we have to declare the respective implementation class as an abstract class and we have to provide implementation for the remaining abstract methods by taking a sub class for the abstract class.

```
1) interface I {
2)     void m1();
3)     void m2();
4)     void m3();
5) }
6) abstract class A implements I {
7)     public void m1() {
8)         System.out.println("m1-A");
9)     }
10}
11) class B extends A {
12)     public void m2() {
13)         System.out.println("m2-B");
14)     }
15}
```



```
14) }
15) public void m3() {
16)     System.out.println("m3-B");
17) }
18)
19) class Test {
20)     public static void main(String args[]) {
21)         I i = new I();
22)         i.m1();
23)         i.m2();
24)         i.m3();
25)         A a = new B();
26)         a.m1();
27)         a.m2();
28)         a.m3();
29)         B b = new B();
30)         b.m1();
31)         b.m2();
32)         b.m3();
33)     }
34} }
```

In Java applications, it is not possible to extend more than one class to a single class but it is possible to extend more than one interface to a single interface.

EX:

```
1) interface I1{
2)     void m1();
3) }
4) interface I2 {
5)     void m2();
6) }
7) interface I3 extends I1, I2 {
8)     void m3();
9) }
10) class A implements I3 {
11)     public void m1() {
12)         System.out.println("m1-A");
13)     }
14)     public void m2() {
15)         System.out.println("m2-A");
16)     }
17)     public void m3() {
18)         System.out.println("m3-A");
19)     }
```



```
20)
21) class Test {
22)     public static void main(String args[]) {
23)         I1 i1 = new A();
24)         i1.m1();
25)         I2 i2 = new A();
26)         i2.m2();
27)         I3 i3 = new A();
28)         i3.m1();
29)         i3.m2();
30)         i3.m3();
31)         A a = new A();
32)         a.m1();
33)         a.m2();
34)         a.m3();
35)     }
36)}
```

In Java applications, it is possible to implement more than one interface into a single implementation class.

```
1) interface I1 {
2)     void m1();
3)
4) interface I2 {
5)     void m2();
6)
7) interface I3 {
8)     void m3();
9)
10) class A implements I1, I2, I3 {
11)     public void m1() {
12)         System.out.println("m1-A");
13)     }
14)     public void m2() {
15)         System.out.println("m2-A");
16)     }
17)     public void m3() {
18)         System.out.println("m3-A");
19)     }
20)
21) class Test {
22)     public static void main(String args[]) {
23)         I1 i1 = new A();
24)         i1.m1();
```



```
25) I2 i2 = new A();  
26) i2.m2();  
27) I3 i3=new A();  
28) i3.m3();  
29) A a = new A();  
30) a.m1();  
31) a.m2();  
32) a.m3();  
33) }  
34) }
```

**Q) Find Valid Syntaxes between classes, abstract classes and Interfaces from the following list of syntaxes?**

- 1) class extends class --->Valid
- 2) class extends class,class--->InValid
- 3) class extends abstract class--->Valid
- 4) class extends abstract class,class--->InValid
- 5) class extends abstract class,abstract class--->InValid
- 6) class extends interface--->InValid
- 7) class implements interface--->Valid
- 8) class implements interface,interface--->Valid
- 9) class implements interface extends class--->InValid
- 10) class implements interface extends abstract class--->InValid
- 11) class extends class implements interface--->Valid
- 12) class extends abstract class implements interface--->Valid
- 13) abstract class extends class--->Valid
- 14) abstract class extends abstract class--->Valid
- 15) abstract class extends class,class--->InValid
- 16) abstract class extends abstract class,abstract class--->InValid
- 17) abstract class extends class,abstract class--->InValid
- 18) abstract class extends interface--->InValid
- 19) abstract class implements interface--->Valid
- 20) abstract class implements interface,interface--->Valid
- 21) abstract class extends class implements interface--->Valid
- 22) abstract class extends abstract class implements interface--->Valid
- 23) abstract class implements interface extends class-->InValid
- 24) abstract class implements interface extends abstract class-->InValid
- 25) interface extends interface -->Valid
- 26) interface extends interface,interface -->Valid
- 27) interface extends class -->InValid
- 28) interface extends abstract class -->InValid
- 29) interface implements interface -->InValid



### Marker Interfaces:

Marker Interface is an Interface, it will not include any abstract method and it will provide some abilities to the objects at runtime of our Java application.

EX: `java.io.Serializable , java.lang.Cloneable`

### java.io.Serializable:

The process of separating the data from an object is called as **Serialization**.

The process of reconstructing an Object on the basis of data is called as **Deserialization**.

**Serialization & Deserialization**

Where `java.io.Serializable` interface is a marker interface, it was not declared any method but it will make eligible any object for **Serialization** and **Deserialization**.

### java.lang.Cloneable:

The process of generating duplicate object is called as **Object Cloning**.

In java applications, by default, all objects are not eligible for Object cloning, only the objects which implements `java.lang.Cloneable` interface are eligible for Object cloning.

## JAVA 8 Features over Interfaces

- Default Methods in Interfaces
- Static Methods in Interfaces
- Functional Interfaces

### Default Methods in Interfaces:

In general, if we declare abstract methods in an interface then we have to implement all that interface methods in more number of classes with variable implementation part.

In the above context, if we require any method implementation common to every implementation class with fixed implementation then we have to implement that method in the interface as **default method**.

To declare default methods in interfaces we have to use "default" keyword in method syntax like access modifier.

EX:

```
1) interface I {  
2)     default void m1() {  
3)         System.out.println("m1-A");  
4)     }  
5) }  
6) class A implements I {}  
7) class Test {  
8)     public static void main(String args[]) {  
9)         I i = new A();
```



```
10)     i.m10;
11) }
12) }
```

**NOTE:** It is possible to provide more than one default methods within a single interface.

**EX:**

```
interface I {
    default void m10 {
        -----
    }
    default void m20 {
        -----
    }
}
```

1. In JAVA8, it is possible to override default methods in the implementation classes.

```
1) interface I {
2)     default void m10 {
3)         System.out.println("m1-A");
4)     }
5) }
6) class A implements I {
7)     public void m10 {
8)         System.out.println("m1-A");
9)     }
10)
11) class Test {
12)     public static void main(String args[]) {
13)         I i = new A();
14)         i.m10();
15)     }
16) }
```

## Static Methods in Interfaces:

Up to JAVA7 version, static methods are not possible in interfaces but from JAVA8 version static methods are possible in interfaces in order to improve sharability.

If we declare static methods in the interfaces then it is not required to declare any implementation class to access that static method, we can use directly interface name to access static method.



**NOTE:** If we declare static methods in an interface then they will not be available to the respective implementation classes, we have to access static methods by using only interface names not even by using interface reference variable

**EX:**

```
1) interface I {  
2)     static void m1() {  
3)         System.out.println("m1-1");  
4)     }  
5) }  
6) class Test {  
7)     public static void main(String args[]) {  
8)         I.m1();  
9)     }  
10}
```

**Note:** In JAVA8 version, interfaces will allow concrete methods along with either "static" keyword or "default" keyword.

## Functional Interface:

If any Java interface allows only one abstract method then it is called as "Functional Interface".

To make any interface as Functional Interface then we have to use the following annotation just above of the interface.

@FunctionalInterface

**EX:** java.lang.Runnable  
java.lang.Comparable

**NOTE:** In Functional Interfaces we have to provide only one abstract method but we can provide any number of default methods and any number of static methods.

```
1) @FunctionalInterface  
2) interface I {  
3)     void m1();  
4)     //void m2();---->error  
5)     default void m3() {  
6)         System.out.println("m3-I");  
7)     }  
8)     static void m4() {  
9)         System.out.println("m4-I");  
10} }  
11) class A implements I {  
12)     public void m1() {  
13)         System.out.println("m1-A");  
14)     }
```



```
14)     }
15)
16) class Test {
17)     public static void main(String args[]) {
18)         I i=new A();
19)         i.m1();
20)         i.m3();
21)         //i.m4();--->error
22)         I.m4();
23)         //A.m4();--->error
24)     }
25} }
```

## Instanceof Operator:

It is a boolean operator, it can be used to check whether the specified reference variable is representing the specified class object or not that is compatible or not.

ref\_Var instanceof Class\_Name

where ref\_Var and Class\_Name must be related otherwise compiler will rise an error like "incompatible types error".

If ref\_Var class is same as the specified Class\_Name then instanceof operator will return "true".

If ref\_Var class is subclass to the specified Class\_Name then instanceof operator will return "true".

If ref\_Var class is super class to the specified Class\_Name then instanceof operator will return "false".

EX:

```
1) class A {
2)
3) class B extends A {
4)
5) class C {
6)
7) class Test {
8)     public static void main(String args[]) {
9)         A a = new A();
10)        B b = new B();
11)        System.out.println(a instanceof A);
12)        System.out.println(a instanceof B);
```



```
13)     System.out.println(b instanceof A);
14) //System.out.println(a instanceof C);
15)
16} }
```

## Object Cloning:

The process of creating duplicate object for an existed object is called as Object Cloning.

If we want to perform Object Cloning in Java application then we have to use the following steps.

- 1) Declare an User defined Class.
- 2) Implement `java.lang.Cloneable` interface in order to make eligible any object for cloning.
- 3) Override `Object` class `clone()` method in user defined class.  
`public Object clone() throws CloneNotSupportedException`
- 4) In Main class, in `main()` method, access `clone()` method over the respective object.

EX:

```
1) class Student implements Cloneable {
2)     String sid;
3)     String sname;
4)     String saddr;
5)     Student(String sid, String sname, String saddr) {
6)         this.sid = sid;
7)         this.sname = sname;
8)         this.saddr = saddr;
9)     }
10)    public Object clone() throws CloneNotSupportedException {
11)        return super.clone();
12)    }
13)    public String toString() {
14)        System.out.println("Student details");
15)        System.out.println("-----");
16)        System.out.println("Student Id :" +sid);
17)        System.out.println("Student name:" +sname);
18)        System.out.println("Student Address:" +saddr);
19)        return "";
20)    }
21}
22 class Test {
23)     public static void main(String args[]) {
24)         Student std1 = new Student("S-111", "Durga", "Hyd");
```



```
25)     System.out.println("Student Details Before Cloning");
26)     System.out.println(std1);
27)
28)     Student std2 = (Student)std1.clone();
29)     System.out.println();
30)     System.out.println("Student Details After cloning");
31)     System.out.println(std2);
32) }
33} }
```

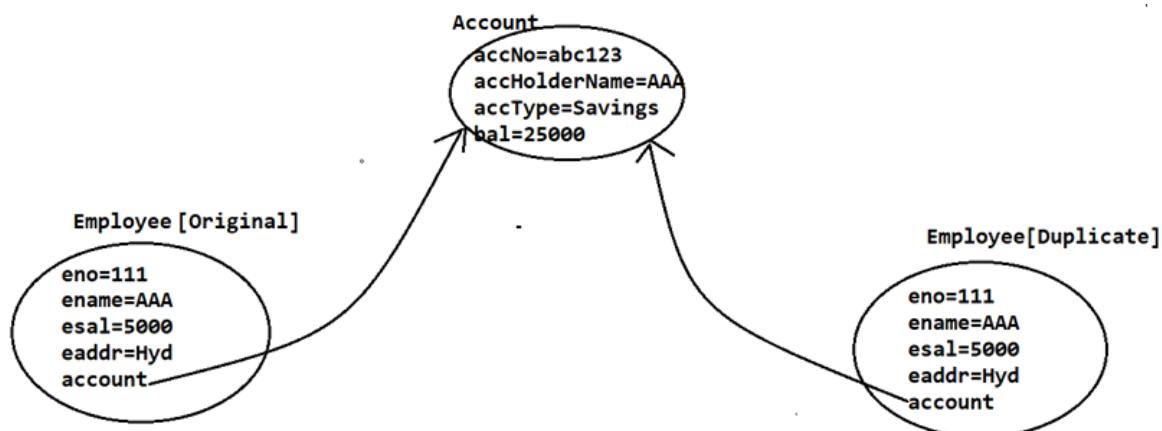
There are 2 types clonings in Java:

- Shallow Cloning/Shallow Copy
- Deep Cloning/Deep Copy

## Shallow Cloning/Shallow Copy:

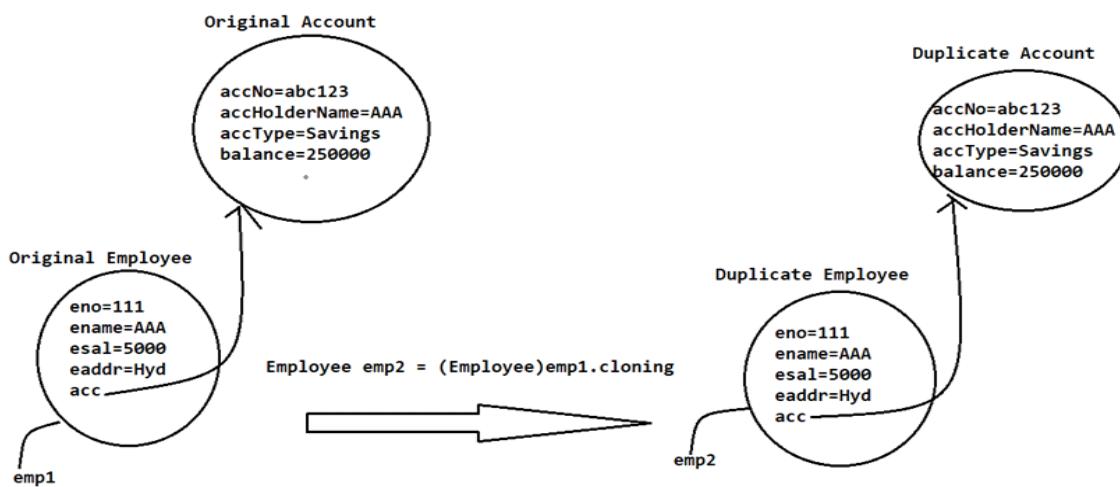
In this cloning mechanism, while cloning an object if any associated object is encountered then JVM will not duplicate associated object along with data duplication, where duplicated object is also refer the same associated object which was referred by original object.

**NOTE:** Shallow Cloning is default cloning mechanism in Java.



## Deep Cloning/Deep Copy:

In this cloning mechanism, while cloning an object if JVM encounter any associated object then JVM will duplicate associated object also along with data duplication. In this cloning mechanism, both original object and cloned object are having their own duplicated associated objects copy, both are not referring a single associated object.



EX:

```
1) class Account {  
2)     String accNo;  
3)     String accName;  
4)     String accType;  
5)     Account(String accNo, String accName, String accType) {  
6)         this.accNo = accNo;  
7)         this.accName = accName;  
8)         this.accType = accType;  
9)     }  
10) }  
11) class Employee implements Cloneable {  
12)     String eid;  
13)     String ename;  
14)     String eaddr;  
15)     Account acc;  
16)     Employee(String eid, String ename, String eaddr, Account acc) {  
17)         this.eid = eid;  
18)         this.ename = ename;  
19)         this.eaddr = eaddr;  
20)         this.acc = acc;  
21)     }  
22)     public Object clone() throws CloneNotSupportedException {  
23)         return super.clone();  
24)     }  
25)     public String toString() {  
26)         System.out.println("Employee Details");  
27)         System.out.println("-----");  
28)         System.out.println("Employee Id :" +eid);  
29)         System.out.println("Employee Address :" +eaddr);  
30)         System.out.println("Employee Name :" +ename);  
31)     }
```



```
31)         System.out.println("Account Deatails");
32)         System.out.println("-----");
33)         System.out.println("Account Number:"+acc.accNo);
34)         System.out.println("Account Name :" +acc.accName);
35)         System.out.println("Account Type :" +acc.accType);
36)         System.out.println("Account Reference: " +acc);
37)         return "";
38)     }
39)
40) class Test {
41)     public static void main(String args[]) throws Exception {
42)         Account acc = new Account("abc123","Durga","Savings");
43)         Employee emp1 = new Employee("E-111","Durga","Hyd",acc);
44)         System.out.println("Employee Details Before Cloning");
45)         System.out.println(emp1);
46)         Employee emp2 = (Employee)emp1.clone();
47)         System.out.println("Employee Details After Cloning");
48)         System.out.println(emp2);
49)     }
50} 
```

To prepare example for deep cloning provide the following clone() method in employee class in the above example of (shallow Cloning)

```
public Object clone()throws CloneNotSupportedException{
Account acc1=new Account(acc.accNo,acc.accName,acc.accType);
Employee emp=new Employee(eid,ename,eaddr,acc1);
return emp;
```

## Adapter Classes:

In general, in Java applications when we implements an interface in any class then we must implement all the abstract methods of that interface with or without the requirement, it may increase unnecessary methods implementation in Java applications.

To overcome the above problem we have to use “Adapter classes”.



```
interface I{
    void m1();
    -----
    void m50();
}

abstract class M implements I {
    void m1(){   }
    -----
    abstract void m25();
    -----
    void m50(){   }
}

class C1 implements I{
    extends M
    void m1(){ --- }
    void m25(){ --- }
}
class C2 implements I{
    extends M
    void m2(){ --- }
    void m25(){ --- }
}
-----
```

Adapter Class  
Generic Class

```
class C10 implements I{
    extends M
    void m10(){ --- }
    void m25(){ --- }
}
```

```
interface I{
    void m1();
    -----
    void m50();
}

abstract class M implements I {
    void m1(){   }
    -----
    abstract void m25();
    -----
    void m50(){   }
}

class C1 implements I{
    extends M
    void m1(){ --- }
    void m25(){ --- }
}
class C2 implements I{
    extends M
    void m2(){ --- }
    void m25(){ --- }
}
-----
```

Adapter Class  
Generic Class

```
class C10 implements I{
    extends M
    void m10(){ --- }
    void m25(){ --- }
}
```



```
interface I{
    void m1();
    -----
    void m50();
}

abstract class M implements I {
    void m1(){ }
    -----
    abstract void m25();
    -----
    void m50(){ }
}

class C1 implements I{
    void m1(){ --- }
    void m25(){ --- }
}

class C2 implements I{
    void m2(){ --- }
    void m25(){ --- }
}

class C10 implements I{
    void m10(){ --- }
    void m25(){ --- }
}
```

Adapter Class  
Generic Class

extends M

extends M

extends M

```
interface I{
    void m1();
    -----
    void m50();
}
}

abstract class A implements I{
    void m1(){ }
    -----
    abstract void m25();
    -----
    void m50(){ }
}

class C1 implements I{
    void m1(){ --- }
    void m25(){ --- }
}

class C2 implements I{
    void m2(){ --- }
    void m25(){ --- }
}

class C10 implements I{
    void m10(){ --- }
    void m25(){ --- }
}
```

Adapter Class

extends A

extends A

extends A

The main intention of Adapter classes is to reduce unnecessary methods implementations while implementing interfaces in implementation classes.



EX:

**WindowAdapter**  
**MouseAdapter**  
**KeyAdapter**  
**GenericServlet**  
**ServletRequestWrapper**  
**ServletResponseWrapper**  
**HttpServletRequestWrapper**  
**HttpServletResponseWrapper**

1

1

```
interface I{
    void m1();
    -----
    void m50();
}

abstract class M implements I {
    void m1(){ }
    -----
    abstract void m25();
    -----
    void m50(){ }
}

class C1 implements I{
    extends M
    void m1(){ --- }
    void m25(){ --- }
}

class C2 implements I{
    extends M
    void m2(){ --- }
    void m25(){ --- }
}

class C10 implements I{
    extends M
    void m10(){ --- }
    void m25(){ --- }
}
```



# Inner Classes



# Inner Classes

Declaring a class inside a class is called as Inner class.

Syntax:

```
class Outer_Class
{
    class Inner_Class
    {
    }
}
```

In Java applications, Inner classes are able to provide the following advantages.

- 1) Modularity
- 2) Abstraction
- 3) Security
- 4) Sharability
- 5) Reusability

### **1) Modularity:**

- In Java application, we are able to improve we modularity by using packages.
- If we want to divide our implementation in a class in the form of modules then we have to use Inner classes.

**EX:** class Account {  
 class SavingsAccount{  
 ----  
 }  
 class CurrentAccount {  
 ----  
 }  
}

### **2) Abstraction:**

If we declare any variable or method in an inner class then that variable and method are having scope up to that inner class only, it is not available to other inner classes, so that, Inner classes are able to improve Abstraction.

**EX:** class A {  
 class B {  
 int i = 10;  
 }  
 class C {



```
void m10 {  
    i = i+10; --> Error  
}  
}  
}
```

## 3) Security:

It is not possible to declare a class as private, but, it is possible to declare an inner class as private, so that, inner classes are able to improve Security.

**EX:** `private class A {  
 }  
Status: Invalid`

**EX:** `class A {  
 private class B {  
 ---  
 }  
}Status: Valid`

## 4) Sharability:

It is not possible to declare a class as Static, but, it is possible to declare an inner class as static, so that, inner classes are able to improve Sharability.

**EX:** `static class A {  
 }  
Status: Invalid`

**EX:** `class A {  
 static class B {  
 ---  
 }  
}Status: Valid`

## 5) Reusability:

- In Java, inheritance feature will improve Code Reusability.
- In java applications, we can extend one inner class to another class, so that, inner classes are able to improve Code Reusability.



```
EX: class A {  
    class B {  
        ---  
    }  
    class C extends B {  
        ---  
    }  
}
```

Status: Valid

There are four types of inner classes in java.

- Member Inner class
- Static Inner class
- Method Local Inner class
- Anonymous Inner class

**Note:** If we compile java file contains inner classes then Compiler will generate a separate .class for outer class and a separate .class file will be generated for inner class.

Outer\_Class.class ---> for Outer class

Outer\_Class\$Inner\_Class.class---> For Inner class

## □ Member Inner Class:

Declaring a normal class [Non static class] inside a class is called as member inner class.

```
EX: class Outer {  
    class Inner {  
        ---  
    }  
}
```

If we want to access the members of inner class then we have to create object for the inner class, for this, we have to use the following syntax.

```
Outer.Inner ref_Var = new Outer().new Inner();
```

- By using outer class reference variable we are able to access only outer class members, we are unable to access inner class members.
- By using Inner class reference variable we are Able to access only inner class members we are unable to access outer class members.
- By default, all outer class members are available to inner class, so we can access outer class members inside the inner class, but, Inner class members are not available to outer classes, we are unable to access inner class members in outer classes.



- Member inner classes are not allowing static declarations directly, but, static keyword is allowed along with final keyword.

**EX:**

```
1) class A
2) {
3)     void m1()
4)     {
5)         System.out.println("m1-A");
6)     }
7)     class B
8)     {
9)         //static int i=10;--> Error
10)        static final int j=20;
11)        void m2()
12)        {
13)            System.out.println("m2-B");
14)            System.out.println(j);
15)        }
16)        void m3()
17)        {
18)            System.out.println("m3-B");
19)        }
20)    }
21)
22) class Test {
23)     public static void main(String[] args)
24)     {
25)         A a=new A();
26)         a.m1();
27)         //a.m2();--> Error
28)         A.B ab=new A().new B();
29)         ab.m2();
30)         ab.m3();
31)         //ab.m1();--> Error
32)     }
33)}
```

In Member inner classes, we can extend one inner class to another inner class but both the inner classes must be provided with in the same outer class.



EX:

```
1) class A
2) {
3)     class B
4)     {
5)         void m1()
6)         {
7)             System.out.println("m1-B");
8)         }
9)         void m2()
10)        {
11)            System.out.println("m2-B");
12)        }
13)    }
14)    class C extends B
15)    {
16)        void m3()
17)        {
18)            System.out.println("m3-C");
19)        }
20)        void m4()
21)        {
22)            System.out.println("m4-C");
23)        }
24)    }
25)
26) class Test
27) {
28)     public static void main(String[] args)
29)     {
30)         A.B ab=new A().new C();
31)         ab.m1();
32)         ab.m2();
33)         //ab.m3();--> Error
34)         A.C ac=new A().new C();
35)         ac.m1();
36)         ac.m2();
37)         ac.m3();
38)         ac.m4();
39)     }
40) }
```

We cannot extend one inner class to another inner class which are available in two different outer classes.



EX:

```
class A {  
    class B {  
        ---  
    }  
}  
class C {  
    class D extends A.B {  
        ---  
    }  
}
```

**Status: Invalid.**

We can extend an inner class from an outer class.

EX:

```
class A {  
    ---  
}  
class B {  
    class C extends A {  
        ---  
    }  
}
```

**Status: Valid**

We can extend the immediate outer class to its inner class.

EX:

```
class A {  
    class B extends A {  
        ---  
    }  
}
```

**Q) Is it possible to write an interface inside a class?**

Yes, it is possible to provide an interface inside a class, but, the respective implementation class must be provided with in the same outer class.



EX:

```
1) class A
2) {
3)     interface I
4)     {
5)         void m1();
6)         void m2();
7)         void m3();
8)     }
9)     class B implements I
10)    {
11)        public void m1()
12)        {
13)            System.out.println("m1-B");
14)        }
15)        public void m2()
16)        {
17)            System.out.println("m2-B");
18)        }
19)        public void m3()
20)        {
21)            System.out.println("m3-B");
22)        }
23)    }
24)
25) class Test
26) {
27)     public static void main(String[] args)
28)     {
29)         A.I ai=new A().new B();
30)         ai.m1();
31)         ai.m2();
32)         ai.m3();
33)     }
34)}
```

EX:

```
1) class A
2) {
3)     abstract class B
4)     {
5)         void m1()
6)         {
7)             System.out.println("m1-B");
8)         }
}
```



```
9)    abstract void m2();
10)   abstract void m3();
11) }
12) class C extends B
13) {
14)   void m2()
15)   {
16)     System.out.println("m2-C");
17)   }
18)   void m3()
19)   {
20)     System.out.println("m3-C");
21)   }
22) }
23}
24) class Test
25) {
26)   public static void main(String[] args)
27)   {
28)     A.B ab=new A().new C();
29)     ab.m1();
30)     ab.m2();
31)     ab.m3();
32)   }
33} }
```

EX:

```
1) package com.durgasoft.core;
2)
3) abstract class A{
4)   class B{
5)     void m1(){
6)       System.out.println("m1-B");
7)     }
8)     void m2(){
9)       System.out.println("m2-B");
10)    }
11)    void m3(){
12)      System.out.println("m3-B");
13)    }
14)  }
15}
16) class C extends A{
17)   // class B{ }
18} }
```



```
19) class Test{  
20)   public static void main(String[] args){  
21)     A.B ab = new C().new B();  
22)     ab.m1();  
23)     ab.m2();  
24)     ab.m3();  
25)   }  
26} }
```

EX:

```
1) package com.durgasoftware.core;  
2)  
3) abstract class A{  
4)   abstract class B{  
5)     void m1(){  
6)       System.out.println("m1-B");  
7)     }  
8)     abstract void m2();  
9)     abstract void m3();  
10)   }  
11)   class C extends B{  
12)     @Override  
13)     void m2() {  
14)       System.out.println("m2-C");  
15)     }  
16)  
17)     @Override  
18)     void m3() {  
19)       System.out.println("m3-C");  
20)     }  
21)   }  
22} }  
23) class D extends A{  
24)  
25} }  
26) class Test{  
27)   public static void main(String[] args){  
28)     A.B ab = new D().new C();  
29)     ab.m1();  
30)     ab.m2();  
31)     ab.m3();  
32)   }  
33} }
```



Ex:

```
1) package com.durgasoft.core;
2)
3) abstract class A{
4)     interface I{
5)         void m1();
6)         void m2();
7)         void m3();
8)     }
9)     class B implements I{
10)        @Override
11)        public void m1() {
12)            System.out.println("m1-B");
13)        }
14)
15)        @Override
16)        public void m2() {
17)            System.out.println("m2-B");
18)        }
19)
20)        @Override
21)        public void m3() {
22)            System.out.println("m3-B");
23)        }
24)    }
25)
26) class C extends A{
27)
28)}
29) class Test{
30)     public static void main(String[] args){
31)         A.I ai = new C().new B();
32)         ai.m1();
33)         ai.m2();
34)         ai.m3();
35)     }
36)}
```



## □ Static Inner Class:

Declaring a static class inside a class is called as Static inner class.

### Syntax:

```
class Outer
{
    static class Inner
    {
        ----
    }
}
```

If we want to access the members of static inner class we have to create object for static inner class, for this we have to use the following syntax.

```
Outer.Inner ref_Var = new Outer.Inner();
```

- Static inner classes are able to allow only static members of the outer class, it will not allow non static members of the outer class.
- In general, inner classes are not allowing static keyword directly, but, static inner class is able to allow static declarations directly.

### EX:

```
1) class A
2) {
3)     static class B
4)     {
5)         void m1()
6)         {
7)             System.out.println("m1-B");
8)         }
9)         void m2()
10)    {
11)        System.out.println("m2-B");
12)    }
13)        static void m3()
14)        {
15)            System.out.println("m3-B");
16)        }
17)    }
18)
19) class Test
20) {
21)     public static void main(String[] args)
```



```
22) {
23)     A.B ab=new A.B();
24)     ab.m1();
25)     ab.m2();
26)     A.B.m3();
27) }
28}
```

## **Q) Is it possible to write a class inside an interface?**

Yes, it is possible to write a class inside an interface, If we declare a class inside an interface then that class is converted as a static inner class, we can access members of this inner class like static inner class members.

**EX:**

```
1) interface I
2) {
3)     class A// static class A
4)     {
5)         void m1()
6)         {
7)             System.out.println("m1-A");
8)         }
9)         void m2()
10)        {
11)             System.out.println("m2-A");
12)        }
13)    }
14)
15) class Test
16) {
17)     public static void main(String[] args)
18)     {
19)         I.A ia=new I.A();
20)         ia.m1();
21)         ia.m2();
22)     }
23)}
```

**Note:** We can write abstract class inside an interface, but, the respective sub class must be declared in the same interface, here the declared sub class is converted as static inner class.



EX:

```
1) interface I
2) {
3)     abstract class A
4)     {
5)         void m1()
6)         {
7)             System.out.println("m1-A");
8)         }
9)         abstract void m2();
10)        abstract void m3();
11)    }
12)    class B extends A
13)    {
14)        void m2()
15)        {
16)            System.out.println("m2-B");
17)        }
18)        void m3()
19)        {
20)            System.out.println("m3-B");
21)        }
22)    }
23)
24) class Test
25) {
26)     public static void main(String[] args)
27)     {
28)         I.A ia=new I.B();
29)         ia.m1();
30)         ia.m2();
31)         ia.m3();
32)     }
33) }
```

**NOTE:** We can write an interface inside an interface, but, the respective implementation class must be provided with in the same outer interface.

EX:

```
1) interface I1
2) {
3)     interface I2
4)     {
5)         void m1();
6)         void m2();
```



```
7)     void m3();
8)   }
9) class A implements I2
10) {
11)   public void m1()
12)   {
13)     System.out.println("m1-A");
14)   }
15)   public void m2()
16)   {
17)     System.out.println("m2-A");
18)   }
19)   public void m3()
20)   {
21)     System.out.println("m3-A");
22)   }
23) }
24)
25) class Test
26) {
27)   public static void main(String[] args)
28)   {
29)     I1.I2 i12=new I1.A();
30)     i12.m1();
31)     i12.m2();
32)     i12.m3();
33)   }
34} }
```

## □ Method Local Inner Class:

Declaring class inside a method is called as Method Inner class.

If we declare a class inside a method then the scope of that is available up to the respective method only, we have to create object for that inner class in the respective method only, we have to access the members of that inner class inside the respective method.

EX:

```
1) class A
2) {
3)   void m1()
4)   {
5)     class B
6)     {
7)       void m2()
```



```
8)     {
9)         System.out.println("m2-B");
10)    }
11)   void m3()
12)   {
13)       System.out.println("m3-B");
14)   }
15) //B
16) B b=new B();
17) b.m2();
18) b.m3();
19) //m10
20) //A
21) class Test
22) {
23)     public static void main(String[] args)
24)     {
25)         A a=new A();
26)         a.m1();
27)     }
28} 
```

## □ Anonymous Inner Classes:

Anonymous inner classes are nameless inner classes, these are used to provide implementation for abstract classes and interfaces.

**Note:** For abstract class we may take sub classes and for interfaces we may take implementation classes to provide implementations , but, here sub classes of the abstract class and implementation class of the interface are able to allow their own members and these sub classes and implementation classes objects are having their identity, not interface identity and abstract class identity.

In java applications, if we want to provide implementations for only abstract class members or interface members and if we want to create objects with interface identity and with abstract class identity then we have to use Anonymous inner classes.

### Syntax:

```
abstract class Name / interface Name
{
    ---
}
class Outer {
    Name ref_Var = new Name()
```



```
{  
    ---implementation----  
};  
}
```

EX:

```
1) abstract class A  
2) {  
3)     void m1()  
4)     {  
5)         System.out.println("m1-A");  
6)     }  
7)     abstract void m2();  
8)     abstract void m3();  
9) }  
10) class Outer  
11){  
12)     A a=new A()  
13)     {  
14)         void m2()  
15)         {  
16)             System.out.println("m2-AIC");  
17)         }  
18)         void m3()  
19)         {  
20)             System.out.println("m3-AIC");  
21)         }  
22)         void m4()  
23)         {  
24)             System.out.println("m4-AIC");  
25)         }  
26)     };  
27}  
28) class Test  
29){  
30)     public static void main(String[] args)  
31)     {  
32)         Outer o=new Outer();  
33)         o.a.m1();  
34)         o.a.m2();  
35)         o.a.m3();  
36)         //o.a.m4();--> Error  
37)     }  
38}
```



EX:

```
1) interface I
2) {
3)     void m1();
4)     void m2();
5)     void m3();
6) }
7) class Outer
8) {
9)     I i=new I()
10)    {
11)        public void m1()
12)        {
13)            System.out.println("m1-AIC");
14)        }
15)        public void m2()
16)        {
17)            System.out.println("m2-AIC");
18)        }
19)        public void m3()
20)        {
21)            System.out.println("m3-AIC");
22)        }
23)        public void m4()
24)        {
25)            System.out.println("m4-AIC");
26)        }
27)    };
28}
29) class Test
30) {
31)     public static void main(String[] args)
32)     {
33)         Outer o=new Outer();
34)         o.i.m1();
35)         o.i.m2();
36)         o.i.m3();
37)         //o.i.m4();--> Error
38)     }
39)}
```

In general, we will use **Anonymous Inner classes** when we have requirement to pass any interface or abstract class references as parameters to the methods.



EX:

```
1) interface I
2) {
3)     void m1();
4)     void m2();
5)     void m3();
6) }
7) class A
8) {
9)     void meth(I i)
10)    {
11)        i.m1();
12)        i.m2();
13)        i.m3();
14)    }
15) }
16) class Test
17) {
18)     public static void main(String[] args)
19)     {
20)         A a=new A();
21)         a.meth(new I()
22)         {
23)             public void m1()
24)             {
25)                 System.out.println("m1-AIC");
26)             }
27)             public void m2()
28)             {
29)                 System.out.println("m2-AIC");
30)             }
31)             public void m3()
32)             {
33)                 System.out.println("m3-AIC");
34)             }
35)             public void m4()
36)             {
37)                 System.out.println("m4-AIC");
38)             }
39)         });
40)     }
41) }
```



# WRAPPER CLASSES



# Wrapper Classes

Collection is an object, it able to store a group of other objects.

In java applications, Collection objects are able to store only objects, they will not store primitive data.

JAVA has provided 8 no of wrapper classes w.r.t the 8 number of primitive data types.

| <u>Primitive DTs</u> | <u>Wrapper Classes</u> |
|----------------------|------------------------|
| 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: Java has provided all the wrapper classes in the form of immutable classes.

## Conversions from Primitive Type To Object Type:

### A) By Using Parameterized Constructors From Wrapper Classes

public XXX(xxx value)  
XXX ----> Wrapper classes  
xxx ----> Primitive types.

EX: int i = 10;  
      Integer in = new Integer(i);  
      System.out.println(i+ " "+in);  
OUTPUT: 10 10

### B) By Using Valueof(-) Method Provided By Wrapper Classes:

public static XXX valueOf(xxx value)  
XXX ----> Wrapper classes  
xxx ----> primitive data types  
EX: int i = 10;  
      Integer in = Intreger.valueOf(i);  
      System.out.println(i+ " "+in);  
OUTPUT: 10 10



## C) By Using Auto-Boxing Approach:

Auto-Boxing approach was provided by JDK5.0 version, in this approach no need to use pre defined methods and constructor, simply, we have to assign primitive variable to wrapper class reference variable.

EX: int i = 10;

```
Integer in = i;  
System.out.println(i+" "+in);  
OUTPUT: 10 10
```

## 2) Conversions From Object Type To Primitive Types:

### a) By Using xxxValue() Method From Wrapper Classes:

public xxx xxxValue()  
xxx ----> primitive data types

EX: Integer in = new Integer(10);  
int i = in.intValue();  
System.out.println(in+" "+i);  
OUTPUT: 10 10

### b) By using Auto-Unboxing:

Auto-Unboxing was provided by JDK 5.0 version, in this approach no need to use any predefined methods, simply assign wrapper class reference variables to the respective primitive variables.

EX: Integer in = new Integer(10);  
int i = in;  
System.out.println(in+" "+i);  
OUTPUT: 10 10

## 3) Conversions from String Type to Object Type:

### • By Using String Parameterized Constructors From Wrapper Classes:

public XXX(String value)

XXX----> Wrapper classes

EX: String data = "10";  
Integer in = new Integer(data);  
System.out.println(data+" "+in);  
OUTPUT: 10 10



## **b) By Using Static valueOf(-) Method From Wrapper Classes:**

```
public static XXX valueOf(String data)
```

EX: String data = "10";  
    Integer in = Integer.valueOf(data);  
    System.out.println(data+ " "+in);  
OUTPUT: 10 10

## **4) Conversions From Object Type To String Type:**

### **a) By using toString() Method from Wrapper Classes:**

```
public String toString()
```

EX: Integer in = new Integer(10);  
    String data = in.toString();  
    System.out.println(in+ " "+data);  
OUTPUT: 10 10

### **b) By Using '+' Concatenation Operator:**

If we concatenate any reference variable with "" by using '+' operator then JVM will access `toString()` method over the provided reference variable.

EX: Integer in = new Integer(10);  
    String data = ""+in;  
    System.out.println(in+ " "+data);  
OUTPUT: 10 10

## **5. Conversions From Primitive Data Types To String Data Types:**

- By using Static `toString()` Method from Wrapper Classes:**

```
public static String toString(xxx value)
```

xxx----> Primitive types

EX: int i = 10;  
    String data = Integer.toString(i);  
    System.out.println(i+ " "+data);  
OUTPUT: 10 10

- By using '+' Concatenation Operator:**

If we concatenate any primitive variable with "" by using '+' operator then the resultant value will be generated in String data type.

EX: int i = 10;  
    String data = ""+i;  
    System.out.println(i+ " "+data);  
OUTPUT: 10 10



## 6) Conversions From String Type To Primitive Type:

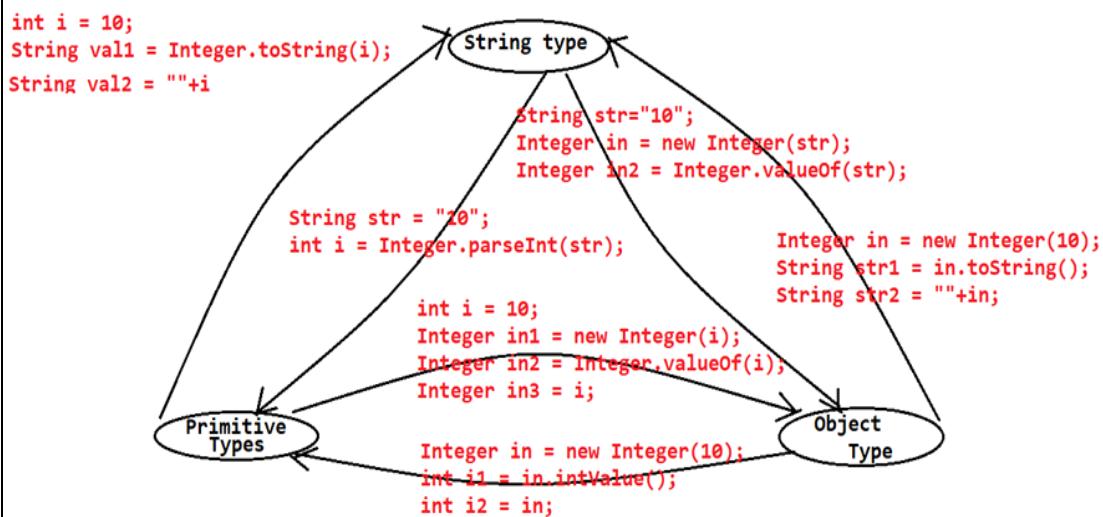
### a) By using parseXXX() Method from Wrapper Classes:

```
public static xxx parseXxx(String data)
```

EX: String data = "10";

```
int i = Integer.parseInt(data);  
System.out.println(data+" "+i);
```

OUTPUT: 10 10





# PACKAGES



# PACKAGES

Package is the collection of related classes and interfaces as a single unit.

Package is a folder contains .class files representing related classes and interfaces

In Java applications, packages are able to provide the following advantages.

- Modularity
- Abstraction
- Security
- Sharability
- Reusability

### **1) Modularity:**

- Dividing a requirement set into no of pieces, providing solutions to the individual parts and combining all individual solutions to a single is called as "Modularization".
- In Java applications, Module is a folder contains .class files representing related classes and interfaces as a single unit.
- Due to the above module definition, we can conclude that Packages are able to improve modularity.

### **2) Abstraction:**

If we declare classes and interfaces in a module or in a package then that classes and interfaces are not visible in outside of the package by default, so that, packages are able to improve Abstraction.

### **3) Security:**

In Java applications, packages are able to provide Abstraction and Encapsulation, where both are able to improve Security.

### **4) Sharability:**

In Java applications, if we declare packages one time then we are able to share that package content to any number of applications or modules at a time.

### **5) Reusability**

In Java applications, if we declare a package one time then we are able to reuse any number of times within a single application or in more than one application.

There are 2 types of packages in Java .

- Predefined Packages
- User defined Packages



## 1) Predefined Packages:

These packages are provided by JAVA programming language along with Java software.

### EX 1: java.lang:

- This package is default package in java applications, no need to import this package to the present java file, when we save java file with .java extension then automatically java.lang package is imported internally.
- This package is able to provide all fundamental classes and interfaces which are required to prepare basic java applications.
- java.lang package includes the following classes and interfaces to prepare java applications.

**String, StringBuffer, StringBuilder,.....**

**Object, System, Class,.....**

**Exception , ArithmeticException, NullPointerException,...**

**Thread, Runnable, Cloneable, Comparable.....**

**Integer, Byte, Short, Float, Long,.....**

### EX2: java.io:

This package is able to provide predefined classes and interfaces in order to perform Input and Output operations in Java.

This package includes the following classes and interfaces in java applications.

**InputStream, ByteArrayInputStream, FileInputStream,.....**

**OutputStream, ByteArrayOutputStream, FileOutputStream,.....**

**Reader, CharArrayReader, FileReader, .....**

**Writer , CharArrayWriter, FileWriter,.....**

**Serializable, Externalizable,.....**

### EX3: java.util:

This package is able to provide all predefined classes and interfaces which are representing data structures .

This package is able to provide classes and interfaces like below.

**Collection, List, Set,Queue**

**ArrayList, Vector, Stack, LinkedList.**

**HashSet, LinkedHashSet, SortedSet, NavigableSet, TreeSet**

**Queue, PriorityQueue, BlockingQueue, LinkedBlockingQueue,.....**

**Map, HashMap, LinkedHashMap, IdentityHashMap, WeakHashMap,.....**



## EX 4: java.awt:

This package is able to provide predefined classes and interfaces representing all GUI components in order to prepare GUI applications.

This package has provided the following classes and interfaces to prepare GUI applications.

Component, Label, TextField, TextArea, Button, CheckBox, List, Choice, Frame,.....

## EX 5: javax.swing:

This package is able to provide predefined library to prepare GUI applications.

## Q) What are the differences between AWT [java.awt] and SWING [javax.swing]?

1. AWT provided GUI components are platform dependent GUI components.  
SWING provided GUI components are Platform Independent GUI Components.

2. AWT provided GUI components are heavy weight GUI components.  
SWING GUI components are light weight GUI components.

3. AWT provided GUI COmponents are basic GUI components.

EX: TextField, TextArea, Label, Button,....

SWING provided GUI components are most advanced GUI components.

EX: JColorChooser, JFileChooser, JTable, JTree,....

4. AWT provided GUI components are able to reduce application performance.

SWING provided GUI components are able to improve application performance.

javax.swing package has provided the following predefined classes and interfaces to design GUI applications.

JComponent, JTextField, JPasswordField, JButton, JCheckBox, JRadioButton, JFrame, JColorChooser, JFileChooser,.....

## EX 6: java.net:

If we prepare any java application without using Client-Server Arch then that java application is called as Standalone Application.

If we prepare any java application on the basis of Client-Server Arch then that java application is called as Distributed application.

To prepare distributed applications, JAVA has provided the following distributed technologies.



- Socket Programming
- RMI [Remote Method Invocation]
- CORBA [Common Object Request Broker Arch/Agent]
- EJBs [Enterprise Java Beans]
- Web Services

**java.net package has provided predefined library to prepare distributed applications by using Socket Programming.**

Socket  
ServerSocket  
URL, URI, URN  
URLConnection  
---  
---

### **EX 7: java.rmi:**

**java.rmi package is able to provide predefined library to prepare Distributed applications on the basis RMI .**

Remote  
RemoteException  
Naming  
---  
---

### **EX 8: java.sql:**

**The process of interacting with database from java application is called as JDBC [Java Database Connectivity].**

**To prepare JDBC applications, java has provided predefined library in the form of java.sql package.**

**java.sql package has provided the following classes and interfaces to prepare JDBC applications.**

**Driver, DriverManager, Connection, Statement, PreparedStatement, CallableStatement, ResultSet, ResultSetMetaData, DatabaseMetaData, ....**



## **2) User Defined Packages:**

These packages are defined by the developers as per their application requirements.

Syntax: package package\_Name;

If we want to use package declaration statement in java applications then we have to use the following two conditions.

- Package Declaration Statement must be first statement.
- Package name must be unique, it must not be sharable and it must not be duplicated.

## **Q) Is It Possible To Provide More Than One Package Declaration Statement Within A Single Java File?**

- No, it is not possible to provide more than one package declaration statement within a single java file, because, package declaration statement must be provided as first statement in java files.
- To provide package names in java applications, JAVA has provided a convention like to include our company domain name in reverse.

EX:

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

If we declare classes and interfaces in a package and if we want to use these classes and interfaces in the present java file then we have to import the respective package to the present java file.

To import packages to the present java file we have to use the following syntaxes.

**import package\_Name.\*;**

--> It able to import all the classes and interfaces of the specified package.

EX: **import java.util.\*;**

**import package\_Name.member\_Name;**

It able to import only the specified member from the specified package.

EX: **import java.util.ArrayList;**

**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.



## **Q) Is It Possible To Use Classes And Interfaces of A Particular Package without Importing The Respective Package?**

Yes, it is possible to use classes and interfaces of a particular package without importing that package, but by using fully qualified names of the respective classes and interfaces.

**Note:** Specifying class name or interface name along with package name is called as "Fully Qualified Name"

**EX:** `java.io.BufferedReader`  
`java.util.ArrayList`  
---  
---

### **A Java program with import statement:**

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

### **A Java program without import statement:**

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

## **Application-1:**

D:\abc  
Employee.java

C:\xyz  
com  
|---durgasoft  
|---core  
|----Employee.class

D:\javaapps\ packages  
Test.java  
Test.class



## Employee.java

```
1) package com.durgasoft.core;
2) public class Employee
3) {
4)     String eid;
5)     String ename;
6)     float esal;
7)     String eaddr;
8)     public Employee(String eid, String ename, float esal, String eaddr)
9)     {
10)         this.eid=eid;
11)         this.ename=ename;
12)         this.esal=esal;
13)         this.eaddr=eaddr;
14)     }
15)     public void getEmpDetails()
16)     {
17)         System.out.println("Employee Details");
18)         System.out.println("-----");
19)         System.out.println("Employee Id :"+eid);
20)         System.out.println("Employee Name :"+ename);
21)         System.out.println("Employee Salary :"+esal);
22)         System.out.println("Employee Address:"+eaddr);
23)     }
24) }
```

On Command Prompt:

D:\abc>javac -d C:\xyz Employee.java

## Test.java

```
1) import com.durgasoft.core.*;
2) public class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         Employee emp=new Employee("E-111", "Durga", 50000.0f, "Hyderabad");
7)         emp.getEmpDetails();
8)     }
9) }
```

On Command Prompt:

D:\javaapps\packages>set classpath=C:\xyz\;

D:\javaapps\packages>javac Test.java

D:\javaapps\packages>java Test



---

--- Employee Details----

**classpath:** To specify the location where package is existed.

**import com.durgasoft.core.\* :** To specify in which package Employee class is existed.

## **JAR Files in JAVA:**

In java applications development, moving .class files from one machine to another machine in the network, uploading .class files directly to the websites,... are not suggestible. Always, it is suggestible to prepare JAR files for the .class files in order to move in the network and to upload applications in the websites.

To create JAR files we have to use the following Command.

**jar -cvf File\_Name.jar \*.\***

**EX:**

D:\javaapps>jar -cvf durgaapp.jar \*.\*

To extract JAR file we have to use the following command on command prompt.

**jar -xvf file\_Name.jar**

**EX:**

D:\javaapps>jar -xvf durgaapp.jar

If we want to access packages or classes or interfaces from JAR file then we have to set classpath path environment variable to JAR file directly.

## **Application-2:**

**D:\abc**

Account.java

**C:\xyz**

com

|---durgasoft

|----icici

|-----accounts

|-----Account.class

**C:\xyz**

durga\_icici.jar

Move durga\_icici.jar from C:\xyz location to D:\xyz location



D:\xyz

durga\_icici.jar

D:\javaapps\packages

Test.java

Test.class

Account.java

```
1) package com.durgasoft.icici.accounts;
2) public class Account
3) {
4)     String accNo;
5)     String accName;
6)     String accType;
7)     String accBranch;
8)     String accBank;
9)     public Account(String accNo, String accName, String accType, String accBranch,
String accBank)
10)    {
11)        this.accNo=accNo;
12)        this.accName=accName;
13)        this.accType=accType;
14)        this.accBranch=accBranch;
15)        this.accBank=accBank;
16)    }
17)    public void getAccountDetails()
18)    {
19)        System.out.println("Account Details");
20)        System.out.println("-----");
21)        System.out.println("Account Number :"+accNo);
22)        System.out.println("Account Name :"+accName);
23)        System.out.println("Account Type :"+accType);
24)        System.out.println("Account Branch :"+accBranch);
25)        System.out.println("Account Bank :"+accBank);
26)    }
27} }
```

On Command Priompt:

D:\abc>javac -d C:\xyz Account.java

C:\xyz>jar -cvf durga\_icici.jar \*.\*

Move durga\_icici.jar from C:\xyz location to D:\xyz location.



## Test.java

```
1) import com.durgasoft.icici.accounts.*;
2) public class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         Account acc=new Account("abc123", "Durga", "Savings", "S R Nagar", "ICICI");
7)         acc.getAccountDetails();
8)     }
9) }
```

On Command Prompt:

D:\javaapps\packages>set classpath=D:\xyz\durga\_icici.jar;..;

D:\javaapps\packages>javac Test.java

D:\javaapps\packages>java Test

---- Account Details-----

## MANIFEST File in JAR:

MANIFEST.MF file is created by "jar" command at the time of creating JAR file under "META-INF" folder and it will provide metadata about the jar file in the form of Key-Value pairs.

## Executable Jar Files:

If we prepare any JAR file with main class and main() method then that jar file is called as "Executable Jar" file.

To prepare Executable JAR file we have to use the following steps.

- Prepare Java application as per the requirement and Compile Java application:

EX: D:\javaapps\packages\ Test.java

```
1) import java.awt.*;
2) class LogoFrame extends Frame
3) {
4)     LogoFrame()
5)     {
6)         this.setVisible(true);
7)         this.setSize(900,300);
8)         this.setBackground(Color.green);
9)         this.setTitle("Logo Frame");
10)    }
11)    public void paint(Graphics g)
```



```
12) {
13)     Font f=new Font("arial", Font.BOLD, 40);
14)     g.setFont(f);
15)     this.setForeground(Color.red);
16)     String logo="DURGA SOFTWARE SOLUTIONS";
17)     g.drawString(logo, 100,150);
18) }
19)
20) class Test
21) {
22)     public static void main(String[] args)
23)     {
24)         LogoFrame lf=new LogoFrame();
25)     }
26)}
```

On Command Prompt D:\javaapps\packages>javac Test.java

- Prepare a text file with "Main-Class" attribute with Main class name:  
D:\javaapps\packages\ abc.txt  
Main-Class: Test

**Note:** The main intention to specify "Main-Class" attribute in text file is to send Main-Class attribute information to MANIFEST.MF file.

- Create JAR file with the MANIFEST.MF file, it must include Main-Class attributes which we specified in text file.  
On Command Prompt D:\javaapps\packages>jar -cvfm durgaapp.jar abc.txt \*.\*
- Execute JAR File  
On Command prompt D:\javaapps\packages>java -jar durgaapp.jar

## Internal Flow:

- JVM will recognize -jar option and JVM will take jar file name from command prompt.
- JVM will search for jar file at current location, if it is available then JVM will go to MANIFEST.MF file which is available META-INF folder.
- In MANIFEST.MF file, JVM will search for Main-Class attribute, if it is available then JVM will take its value that is Main Class name.
- After getting Main Class name from MANIFEST.MF file JVM will execute Main Class and generate output.

If we want to execute the above application through batch file, first, we have to prepare batch file with the required execution command then double click on that batch.



---

**durga.bat [Take Text file and save with .bat extension]**

**java -jar durgaapp.jar**

**NOTE:** We must save both jar file and bat file at the same location.



# STRING MANIPULATIONS



# String Manipulations

In C and C++ applications, to perform String operations, C and C++ programming languages have provided some predefined library in the form of the functions.

In Java, to perform String operations JAVA has provided the following predefined classes.

- `java.lang.String`
- `java.lang.StringBuffer`
- `java.lang.StringBuilder`
- `java.util.StringTokenizer`

## **Q) What is the difference between *String* and *StringBuffer*?**

- String class objects are immutable objects, where Immutable objects are not allowing modifications over their content, if we are trying to perform modifications over immutable object content then operations are allowed , but, the resultant data will not be stored back in original object, where the resultant data will be stored by creating new object.
- StringBuffer objects are mutable objects, they are able to allow modifications on their content.

## **Q) What are the differences between *StringBuffer* and *StringBuilder*?**

- StringBuffer class was introduced in JDK1.0 version.  
StringBuilder class was introduced in JDK5.0 version.
- StringBuffer is synchronized.  
StringBuilder is not synchronized.
- Almost all the methods are synchronized in StringBuffer.  
No method is synchronized in StringBuilder.
- StringBuffer is able to allow only one thread to access data.  
StringBuilder is able to allow more than one thread to access data.
- StringBuffer is following sequential execution.  
StringBuilder is following parallel execution.
- StringBuffer will increase execution time.  
StringBuilder will reduce execution time.
- StringBuffer will reduce application performance.  
StringBuilder will increase application performance.



- **StringBuffer** is able to give guarantee for Data Consistency.  
**StringBuilder** is unable to give guarantee for data consistency.
- **StringBuffer** is threadsafe.  
**StringBuilder** is not thread safe.

## **Q) What is String tokenization and how it is possible to perform String Tokenization in Java Applications?**

- The process of dividing a string into no of tokens is called as String Tokenization.
- To perform String Tokenization JAVA has provided a predefined class in the form of `java.util.StringTokenizer`.

To perform String Tokenization in java applications we have to use the following steps.

- **Create StringTokenizer Class Object:**

To create `StringTokenizer` class object we have to use the following constructor.

```
public StringTokenizer(String data)
```

**EX:** `StringTokenizer st=new StringTokenizer("Durga Software Solutions");`

When JVM encounter the above instruction, JVM will take the provided String, JVM will divide the provided String into no of tokens and JVM will store the generated tokens by creating object for `StringTokenizer` class.

**NOTE:** When `StringTokenizer` class object is created with the no of tokens then a pointer or cursor will be created before the first token.

To get no of tokens which are existed in `StringTokenizer` class object we have to use the following method.

```
public int countTokens()
```

- **Retrieve Tokens from StringTokenizer Object:**

To retrieve tokens from `StringTokenizer` object we have to use the following steps for each and every token.

- Check whether more tokens are available or not from the current cursor position by using the following method. `public boolean hasMoreTokens()`
  - It will return true value if at least next token is existed.
  - It will return false value if no more tokens are existed.
- 
- If atleast next token is available then read next token and move cursor to next position by using the following method.  
`public String nextToken()`

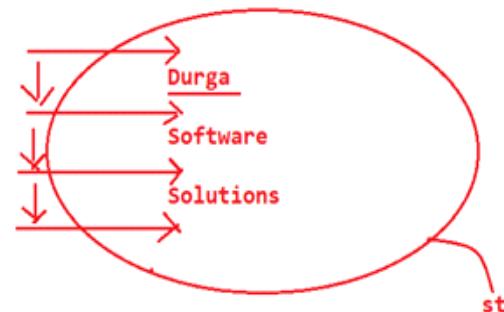


EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         StringTokenizer st=new StringTokenizer("Durga Software Solutions");
7)         System.out.println("No Of Tokens:"+st.countTokens());
8)         while(st.hasMoreTokens())
9)         {
10)             System.out.println(st.nextToken());
11)         }
12)     }
13} }
```

```
StringTokenizer st = new StringTokenizer("Durga Software Solutions");
int count = st.countTokens()

while(st.hasMoreTokens())
{
    Sopln(st.nextToken());
}
```



## String Class Library:

### Constructor:

- public String()

It able to create an empty String object.

EX: String str = new String();

- public String(String str)

This constructor can be used to create String class object with the specified data.

EX: String str = new String("Durga Software Solutions");

System.out.println(str);

Output: Durga Software Solutions



## Q) What is the difference between the following 2 Statements

- a) String str = new String("Durga Software Solutions");
- b) String str = "Durga Software Solutions";

- To create String class object if we use first statement then one String object will be created at heap memory as per "new" keyword and another String object will be created at String Constant Pool Area inside method Area as per "".
- To create String object if we use second statement then String object will be created at String Constant Pool Area only in Method Area.

**Note:** The objects which are created in String Constant Pool Area are not eligible for Garbage Collection. The objects which are created in Heap memory are eligible for Garbage Collection.

## 3) public String(byte[] b)

This constructor can be used to create String object with the String equivalent of the specified byte[].

**EX:** byte[] b = {65, 66, 67, 68, 69, 70};

```
String str = new String(b);  
System.out.println(str);
```

**Output:** ABCDEF

## 4) public String(byte[] b, int start\_index, int no\_Of\_Chars)

This constructor can be used to create String class object with the String equivalent of the specified byte[] starts from the specified start index and up to the specified no of elements.

**EX:** byte[] b = {65, 66, 67, 68, 69, 70};

```
String str = new String(b, 2, 3);  
System.out.println(str);
```

**Output:** CDE

## 5) public String(char[] ch)

This constructor can be used to create String object with the String equivalent of the specified char[].

**EX:** char[] ch = {'A', 'B', 'C', 'D', 'E', 'F'};

```
String str = new String(ch);  
System.out.println(str);
```

**Output:** ABCDEF



## 6) **public String(char[] ch, int start\_Index,int no\_Of\_Chars)**

It will create String object with the String equivalent of the specified char[] starts from the specified start index and up to the specified no of element.

**EX:** `char[] ch = {'A','B','C','D','E','F'};`  
`String str = new String(ch,2,3);`  
`System.out.println(str);`  
**Output:** CDE

**Note:** Constructors 3 and 4 are used to convert data from byte[] to String and constructors 5 and 6 are used to convert data from char[] to String.

## Methods:

- **public int length()**

This method will return an integer value representing the no of characters existed in the String including spaces.

**EX:** `String str = new String("Durga Software Solutions");`  
`System.out.println(str.length());`  
**Output:** 24

- **public String concat(String str)**

This method will add the specified String to String object content in immutable manner.

**EX:**

```
String str1 = new String("Durga ");
String str2 = str1.concat("Software ");
String str3 = str2.concat("Solutions");
System.out.println(str1);
System.out.println(str2);
System.out.println(str3);
```

**Output:** Durga  
Durga Software  
Durga Software Solutions

**EX:**

```
String str = "Durga".concat("Software ").concat("Solutions");
System.out.println(str);
```

**Output:** Durga Software Solutions



- **public boolean equals(Object obj)**

This method will check whether the two String objects content is same or not. if two string objects content is same then equals() method will return "true" value otherwise equals() method will return "false" value.

**Q) What is the difference between == Operator and equals(-) Method?**

- '==' is basically a comparison operator, it will check whether the provided operand values are same or not, where operands may be normal primitive variables or object reference variables.
- Initially equals() method was defined in java.lang.Object class , it was implemented in such a way that to provide comparison between two object reference values.
- String class has overridden Object class equals() method in such a way that to provide comparison between two String object contents

**EX:**

```
1) class A
2) {
3) }
4) class Test
5) {
6)     public static void main(String[] args)
7)     {
8)         A a1=new A();
9)         A a2=new A();
10)
11)        int i=10;
12)        int j=10;
13)
14)        String str1=new String("abc");
15)        String str2=new String("abc");
16)
17)        System.out.println(i == j); // true
18)        System.out.println(a1 == a2); // false
19)        System.out.println(str1 == str2); // false
20)        System.out.println(a1.equals(a2)); // false
21)        System.out.println(str1.equals(str2)); // true
22)    }
23} }
```



## public boolean equalsIgnoreCase(String str)

In String class, equals(-) method will perform case sensitive comparison between two String object contents, but, equalsIgnoreCase(-) method will perform case insensitive comparison between two String objects content.

EX:

```
String str1 = new String("abc");
String str2 = new String("ABC");
System.out.println(str1.equals(str2));
System.out.println(str1.equalsIgnoreCase(str2));
```

Output:

```
false
true
```

## • public int compareTo(String str)

This method will check whether two string objects contents are existed in dictionary order or not.

## str1.compareTo(str2)

- If str1 come first when compared with str2 in dictionary order then compareTo() method will return -ve value.
- If str2 come first when compared with str1 in dictionary order then compareTo() method will return +ve value.
- If str1 and str2 are available at the same position in dictionary order then compareTo() method will return 0 value.

EX:

```
String str1 = new String("abc");
String str2 = new String("def");
String str3 = new String("abc");
System.out.println(str1.compareTo(str2));
System.out.println(str2.compareTo(str3));
System.out.println(str3.compareTo(str1));
```

Output: -3

```
3
0
```



- **public boolean startsWith(String str)**

It will check whether the String starts with the specified String or not.

- **public boolean endsWith(String str)**

It will check whether the String ends with the specified String or not.

- **public boolean contains(String str)**

It will check whether String contains the specified String or not.

EX:

```
String str=new String("Durga Software Solutions");
System.out.println(str.startsWith("Durga"));
System.out.println(str.endsWith("Solutions"));
System.out.println(str.contains("Software"));
```

**OUTPUT:**

```
true
true
true
```

- **public String replace(char old\_Char, char new\_Char)**

This method will replace the specified old char with the specified new char.

- **public char charAt(int index)**

It will return a character which is existed at the specified index value.

- **public int indexOf(String str)**

It will return an index value where the first occurrence of the specified String.

- **public int lastIndexOf(String str)**

It will return an index value where the last occurrence of the specified String.

EX:

```
String str=new String("Durga Software Solutions");
System.out.println(str);
System.out.println(str.replace('S', 's'));
System.out.println(str.charAt(6));
System.out.println(str.indexOf("So"));
System.out.println(str.lastIndexOf("So"));
```



**Output:** Durga Software Solutions

Durga software solutions

S

6

15

- **public String substring(int start\_index)**

This method return a substring starts from the specified index value.

- **public String substring (int start\_Index, int end\_Index)**

This method will return a substring starts from the specified start index and up to the specified end index.

**EX:**

```
String str = new String("Durga Software Solutions");
System.out.println(str.substring(6));
System.out.println(str.substring(6,14));
```

**Output:**

Software Solutions  
Software

- **public byte[] getBytes()**

This method will convert data from String to byte[].

- **public char[] toCharArray()**

This method will convert data from String to char[].

**EX:**

```
1) String str = new String("Durga Software Solutions");
2) byte[] b = str.getBytes();
3) for(int i=0; i<b.length; i++)
4) {
5)   System.out.print(b[i]+ " ");
6) }
7) System.out.println();
8) char[] ch = str.toCharArray();
9) for(int i=0; i<ch.length; i++)
10) {
11)   System.out.print(ch[i]+ " ");
12) }
```



## public String[] split(String str)

It will divide the String into no pieces on the basis of the provided String.

EX:

```
String str = new String("Durga Software Solutions");
String[] s = str.split(" ");
for(int i=0; i<s.length; i++)
{
    System.out.println(s[i]);
}
```

- public String trim()

This method will remove pre-spaces and post-spaces of a particular String.

- public String toLowerCase()

It will convert String into lower case letters.

- public String toUpperCase()

It will convert String into Upper case letters.

EX: String str = new String(" Durga Software Solutions ");
System.out.println(str);
System.out.println(str.trim());
String str1 = new String("Durga Software Solutions");
System.out.println(str1.toLowerCase());
System.out.println(str1.toUpperCase());

## StringBuffer:

### Constructors:

- public StringBuffer()

This constructor can be used to create an empty StringBuffer object with 16 elements as initial capacity.

EX: StringBuffer sb=new StringBuffer();
System.out.println(sb.capacity());
Output: 16



- **public StringBuffer(int capacity)**

This constructor can be used to create an empty StringBuffer object with the specified capacity value.

**EX:** `StringBuffer sb=new StringBuffer(10);`

`System.out.println(sb.capacity());`

**Output:** 10

- **public StringBuffer(String str)**

This constructor can be used to create StringBuffer object with the specified data.

`New_capacity = initial_Capacity+data_Length;`

**EX:** `StringBuffer sb = new StringBuffer("abc");`

`System.out.println(sb);`

`System.out.println(sb.capacity());`

**Output:** abc

19

## Methods:

- **public StringBuffer append(String data)**

This method will append the specified data to the StringBuffer object content.

**EX:** `StringBuffer sb1 = new StringBuffer("Durga ");`

`StringBuffer sb2 = sb1.append("Software ");`

`StringBuffer sb3 = sb2.append("Solutions");`

`System.out.println(sb1);`

`System.out.println(sb2);`

`System.out.println(sb3);`

**Output:** Durga Software Solutions

Durga Software Solutions

Durga Software Solutions

- **public void ensureCapacity(int capacity)**

This method can be used to set a particular capacity value to StringBuffer object. If the provided capacity value is less than 16 then StringBuffer object will take 16 as capacity value. If the provided capacity value is greater than 16 and less than 34 then StringBuffer will take 34 as capacity value. If the provided capacity value is greater than 34 then StringBuffer object will take the specified value as capacity value.



**EX:** `StringBuffer sb = new StringBuffer();  
sb.ensureCapacity(10);  
System.out.println(sb.capacity());`

**Output:** 16

- **public StringBuffer reverse()**

This method will reverse the content of StringBuffer object.

**EX:** `StringBuffer sb = new StringBuffer("Durga Software Solutions");  
System.out.println(sb);  
System.out.println(sb.reverse());`

### **public StringBuffer delete(int start index, int end index)**

This method can be used to delete a string from StringBuffer object content starts from the specified start index and up to the specified end index.

**EX:**

```
StringBuffer sb = new StringBuffer("Durga Software Solutions");  
System.out.println(sb);  
System.out.println(sb.delete(6,14));
```

**Output:**

Durga Software Solutions  
Durga Solutions

- **public StringBuffer insert(int index, String str)**

This method can be used to insert the specified String at the specified index in StringBuffer object.

**EX:** `StringBuffer sb = new StringBuffer("Durga Solutions");  
System.out.println(sb);  
System.out.println(sb.insert(6,"Software"));`

**Output:**

Durga Solutions  
Durga Software Solutions



# Exception Handling



# Exception Handling

## Q) What is the difference between Error and Exception?

Error is a problem in java applications.

There are two types of errors in java.

- Compile time Errors
- Runtime Errors

### • Compile Time Errors:

These are problems identified by the compiler at compilation time.

In general, there are 3 types of compile time errors.

- Lexical Errors: Mistakes in keywords,..

EX: int i=10; ----> Valid

nit i=10; ----> Invalid

- Syntax Errors: Gramatical Mistakes or syntactical mistakes.

EX: int i=10;----> Valid

i int 10 ; --> Invalid

- Semantic Errors: Providing operators in between incompatible types.

EX: int i=10;

int j=20;

int k=i+j; ----> Valid

EX: int i=10;

boolean b=true;

char c= i+b;----> Invalid.

**NOTE:** Some other errors are generated by Compiler when we violate JAVA rules and regulations in java applications.

**EX:** Unreachable Statements, variable might not have been initialization, possible loss of precision,....

### • Runtime Errors:

These are the problems for which we are unable to provide solutions programmatically and these errors are not identified by the compilers and these errors are occurred at runtime.



**EX:** Insufficient Main Memory.

Unavailability of IO Components.

JVM Internal Problems.

## 2) Exception:

Exception is a problem, for which we are able to provide solutions programmatically.

**EX:** ArithmeticException

NullPointerException

ArrayIndexOutOfBoundsException

----

----

**Exception:** Exception is an unexpected event occurred in java applications at runtime , which may be provided by users while entering dynamic input in java applications, provided by the Database Engines while executing sql queries in Jdbc applications, provided by Network while establish connection between local machine and remote machine,.....causes abnormal termination to the java applications.

There are two types of terminations in java applications.

- **Smooth Termination**

Terminating program at end is called as Smooth termination.

- **Abnormal Termination**

Terminating program in the middle is called as Abnormal Termination.

In general, Exceptions are providing abnormal terminations, these abnormal terminations may crash local operating systems, these abnormal terminations may provide hanged out situations to the network,.....

To overcome the above problems we have to handle exceptions properly, to handle exceptions properly we have to use "Exception Handling Mechanism".

Java is a Robust programming language, because,

- Java is having very good memory management system in the form of Heap Memory Management system, it is a dynamic memory management system, it allocates and deallocates memory for the objects at runtime.
- Java is having very good exception handling mechanisms, JAVA has provided very good predefined library to represent and handle almost all the frequently generated exceptions.

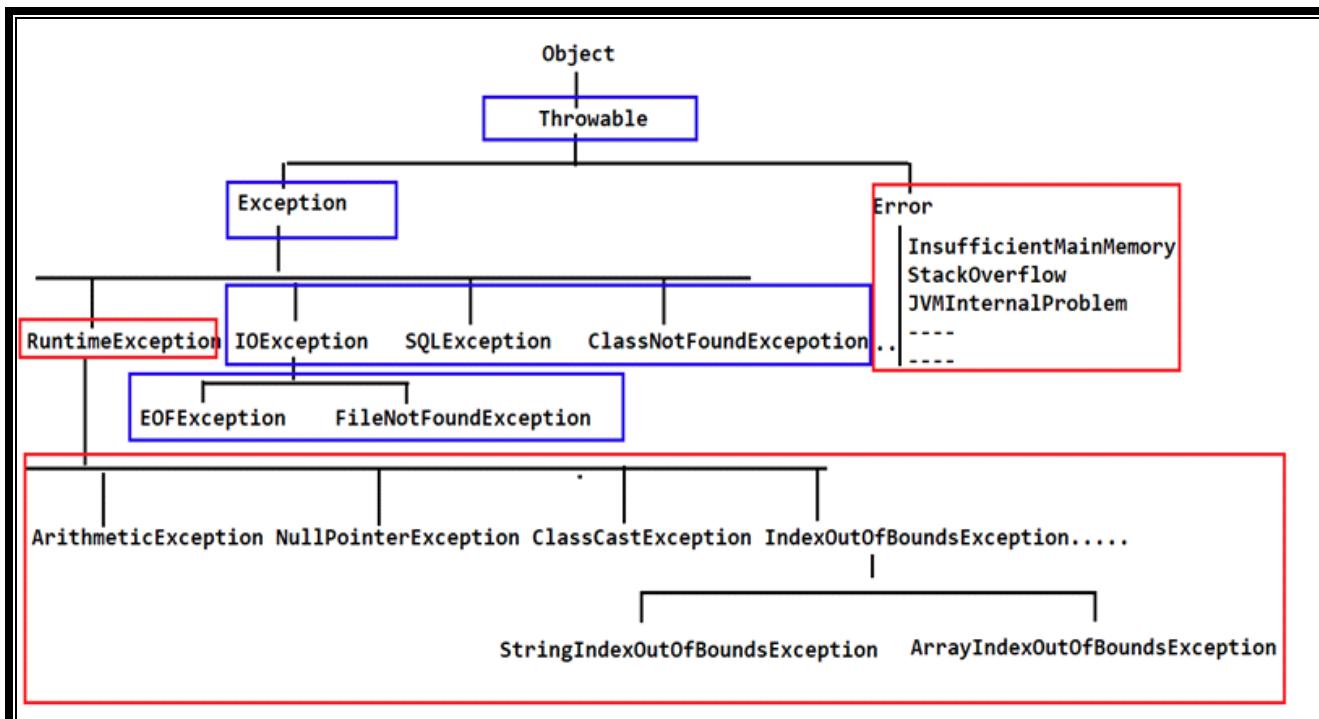


There are two types of exceptions in Java.

- Predefined Exceptions
- User defined Exceptions

## • Predefined Exceptions

These Exceptions are defined by JAVA programming language and provided along Java software.



There are two types predefined Exceptions.

- Checked Exceptions
- Unchecked Exceptions

## Q) What is the difference between *Checked Exceptions* and *Unchecked Exceptions*?

- Checked Exception is an exception recognized at compilation time, but, not occurred at compilation time.
- Unchecked Exceptions are not recognized at compilation time, these exceptions are recognized at runtime by JVM.

**Note:** In Exceptions Arch, **RuntimeException** and its sub classes and **Error** and its sub classes are the examples for Unchecked Exceptions and all the remaining classes are the examples for Checked Exceptions.



There are two types of Checked Exceptions

- Pure Checked Exceptions
- Partially Checked Exceptions

## **Q) What is the difference between *Pure Checked Exception* and *Partially Checked Exceptions*?**

If any checked exception is having only checked exceptions as sub classes then this Exception is called as Pure Checked Exception.

EX: IOException

If any Checked Exception contains at least one sub class as unchecked exception then that Checked Exception is called as Partially Checked Exception.

EX: Exception, Throwable

## **Predefined Exceptions Overview:**

### **• ArithmeticException**

In Java applications, when we have a situation like a number is divided by zero then JVM will rise Arithmetic Exception.

EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int i=10;
6)         int j=0;
7)         float f=i/j;
8)         System.out.println(f);
9)     }
10) }
```

If we run the above program then JVM will provide ArithmeticException with the following Exception message.

Exception in thread "main" java.lang.ArithmaticException: / by zero  
at Text.main(Test.java:7)

The above exception message is devided into the following three parts.

- Exception Name: java.lang.ArithmaticException
- Exception Descrioptioun: / by zero
- Exception Location : Test.java:7



- **NullPointerException:**

In java applications, when we access any instance variable and instance methods by using a reference variable contains null value then JVM will rise NullPointerException.

**EX:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         java.util.Date d=null;
6)         System.out.println(d.toString());
7)     }
8) }
```

If we run the above code then JVM will provide the following exception details.

- Exception Name: java.lang.NullPointerException
- Exception Description: --- No description----
- Exception Location: Test.java:6

- **ArrayIndexOutOfBoundsException:**

In Java applications, when we insert an element to an array at a particular index value and when we are trying to access an element from an array at a particular index value and if the specified index value is in outside of the arrays size then JVM will rise an exception like "ArrayIndexOutOfBoundsException".

**EX:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         int [] a={1,2,3,4,5};
6)         System.out.println(a[10]);
7)     }
8) }
```

If we run the above program then JVM will rise an exception with the following details.

Exception Name: java.lang.ArrayIndexOutOfBoundsException

Exception Description: 10

Exception Location: Test.java: 6



- **ClassCastException:**

In Java applications, we are able to keep sub class object reference value in super class reference variable, but, we are unable to keep super class object reference value in sub class reference variable. If we are trying to keep super class object reference value in sub class reference variable then JVM will rise an exception like "java.lang.ClassCastException".

EX:

```
1) class A
2) {
3) }
4) class B extends A
5) {
6) }
7) class Test
8) {
9)     public static void main(String[] args)
10)    {
11)        A a=new A();
12)        B b=(B)a;
13)    }
14) }
```

- If we run the above code then JVM will rise an exception with the following details.
- Exception Name: java.lang.ClassCastException
- Exception Description: A cannot be cast to B
- Exception location: Test.java: 12

- **ClassNotFoundException:**

In Java applications, if we want to load a particular class byte code to the memory without creating object then we will use the following method from java.lang.Class class.

```
public static Class.forName(String class_Name) throws ClassNotFoundException
```

EX: Class c=Class.forName("A");

When JVM encounter the above instruction, JVM will search for A.class file at current location, at java predefined library and at the locations referred by "classpath" environment variable, if the required A.class file is not identified at all the locations then JVM will rise ClassNotFoundException.



EX:

```
1) class A
2) {
3)     static
4)     {
5)         System.out.println("Class Loading");
6)     }
7) }
8) class Test
9) {
10)    public static void main(String[] args) throws Exception
11)    {
12)        Class c=Class.forName("AAA");
13)    }
14) }
```

If we run this code the JVM will provide the following exception details.

- Exception Name: java.lang.ClassNotFoundException
- Exception Description: AAA
- Exception Location: Test.java: 12
- InstantiationException 5.IllegalArgumentException

In java applications, if we load class byte code by using "Class.forName(-)" method then to create object explicitly we have to use the following method from java.lang.Class class.

`public Object newInstance()`

EX: `Object obj = c.newInstance()`

When JVM encounter the above code then JVM will search for 0-arg constructor and non-private constructor in the loaded class. If 0-arg constructor is not available, if parameterized constructor is existed then JVM will rise "java.lang.instantiationException". If non-private constructor is not available, if private constructor is available then JVM will rise "java.lang.IllegalAccessException".

EX1:

```
1) class A
2) {
3)     static
4)     {
5)         System.out.println("Class Loading");
6)     }
7)     A(int i)
```



```
8)  {
9)      System.out.println("Object Creating");
10) }
11) }
12) class Test
13) {
14)     public static void main(String[] args) throws Exception
15)     {
16)         Class c=Class.forName("A");
17)         Object obj=c.newInstance();
18)     }
19) }
```

If run the above code then JVM will provide an exception with the following details.

- Exception Name : java.lang.InstantiationException
- Exception Description: A
- Exception Location:Test.java: 17

EX:

```
1) class A
2) {
3)     static
4)     {
5)         System.out.println("Class Loading");
6)     }
7)     private A()
8)     {
9)         System.out.println("Object Creating");
10)    }
11) }
12) class Test
13) {
14)     public static void main(String[] args) throws Exception
15)     {
16)         Class c=Class.forName("A");
17)         Object obj=c.newInstance();
18)     }
19) }
```

If we run the above code then JVM will rise an exception with the following details

- Exception Name : java.lang.IllegalAccessException
- Exception Description: Test class cannot access the members of class A with the modifier "private".
- Exception Location" Test.java: 17



## **'throw' keyword:**

'throw' is a java keyword, it can be used to rise exceptions intentionally as per the developers application requirement.

### **Syntax:**

```
throw new Exception_Name("----Exception Description----");
```

### **EX:**

```
1) class Test
2) {
3)     public static void main(String[] args) throws Exception
4)     {
5)         String accNo=args[0];
6)         String accName=args[1];
7)         int pin_Num=Integer.parseInt(args[2]);
8)         String accType=args[3];
9)         System.out.println("Account Details");
10)        System.out.println("-----");
11)        System.out.println("Account Number :"+accNo);
12)        System.out.println("Account Name :"+accName);
13)        System.out.println("Account Type :"+accType);
14)        System.out.println("Account PIN Number:"+pin_Num);
15)        if(pin_Num>=1000 && pin_Num<=9999)
16)        {
17)            System.out.println("valid PIN Number");
18)        }
19)        else
20)        {
21)            throw new RuntimeException("Invalid PIN Number, enter Valid 4 digit PIN
Number");
22)        }
23)    }
24} }
```

```
D:\javaapps>javac Test.java
D:\javaapps>java Test abc123 AAA 1234 Ssavings
--- Account details without Exception----
```

```
D:\javaapps>java Test abc123 AAA 123 Savings
--- Account details are displayed with Exception----
```

---Account details----

Exception in thread "main" java.lang.RuntimeException: Invalid PIN Number, enter valid 4 digit PIN number at Test.main(Test.java:17)



There are two ways to handle exceptions.

- By using 'throws' keyword.
- By using try-catch-finally block.

## • **'throws' keyword:**

- It is a Java keyword, it can be used to bypass the generated exception from the present method or constructor to the caller method (or) constructor.
- In Java applications, 'throws' keyword will be used in method declarations, not in method body.
- In Java applications, "throws" keyword allows an exception class name, it should be either same as the generated exception or super class to the generated exception. It should not be subclass to the generated Exception.
- 'throws' keyword allows more than one exception in method prototypes.
- In Java applications, "throws" keyword will be utilized mainly for checked exceptions.

**EX:** void m1() throws RuntimeException {  
    throw new ArithmeticException();  
}

Status:Valid

**EX:** void m1() throws FileNotFoundException {  
    throw new IOException();  
}

Status:Invalid

**EX:** Void m1() throws NullPointerException, ClassNotFoundException {  
}

Status:Valid

**EX:** Void m1() throws IOException, FileNotFoundException  
{  
}

Status: Valid, Not Suggestible

If we specify any super exception class along with throws keyword, then it is not necessary to specify any of its child exception classes along with "throws" keyword.



**NOTE:** In any Java method, if we call some other method which is bypassing an exception by using “throws” keyword, then we must handle that exception either by using “throws” keyword in the present method [Caller Method] prototype or by using “try-catch-finally” in the body of the present method [Caller method].

**EX:**

```
Void m1() throws Exception {
```

```
----
```

```
}
```

```
Void m2() {
```

```
    try {
```

```
        m1();
```

```
    }
```

```
    catch(Exception e) {
```

```
        e.printStackTrace();
```

```
    }
```

```
}
```

**EX:**

```
void m1() throws Exception {
```

```
----
```

```
}
```

```
void m2() throws Exception {
```

```
    m1();
```

```
}
```

**EX:**

```
1) import java.io.*;  
2) class A {  
3)     void add() throws Exception {  
4)         concat();  
5)     }  
6)     void concat() throws IOException {  
7)         throw new IOException();  
8)     }  
9) }  
10) class Test {  
11)     public static void main(String args[]) throws Throwable {  
12)         A a = new A();  
13)         a.add();  
14)     }  
15) }
```



## Internal Flow:

If we execute the above program, then JVM will recognize throw keyword in concat() method and JVM will rise an exception in concat() method, due to throws keyword in concat() method prototype, Exception will be bypassed to concat() method call that is in add(), due to throws keyword in add() method Exception will be bypassed to add() method call , that is , in main() method, Due to 'throws' keyword in main() method Exception will be bypassed to main() method call , that is, to JVM, where JVM will activate "Default Exception Handler" , where Default Exception Handler will display exception details.

## Q) What are the differences between “throw” and “throws” Keywords?

- 'throw' keyword can be used to rise the exceptions intentionally as per the application requirement.
- 'throws' keyword is able to bypass the exception from the present method to the caller method.
- 'throw' keyword will be utilized in method body.
- 'throws' keyword will be used in method declarations or in method prototype (or) in method header part.
- 'throw' keyword allows only one exception class name.
- 'throws' keyword allows more than one exception class name.

## try-catch-finally:

In Java application “throws” keyword is not really an exception handler, because “throws” keyword will bypass the exception handling responsibility from present method to the caller method.

If we want to handle the exceptions, the locations where exceptions are generated then we have to use “try-catch-finally”.

### Syntax:

```
try {  
    ----  
}  
catch(Exception_Name e) {  
    ----  
}  
finally {  
    ----  
}
```

Where the purpose of try block is to include a set of exceptions, which may rise an exception.



If JVM identify any exception inside "try" block then JVM will bypass flow of execution to "catch" block by skipping all the remaining instructions in try block and by passing the generated Exception object reference as parameter.

If no exception is identified in "try" block then JVM will execute completely "try" block, at the end of try block, JVM will bypass flow of execution to "finally" block directly.

The main purpose of catch block is to catch the exception from try block and to display exception details on command prompt.

To display exception details on command prompt, we have to use the following three approaches.

- `e.printStackTrace()`
- `System.out.println(e);`
- `System.out.println(e.getMessage());`

### • **e.printStackTrace();**

It will display the exception details like Exception Name, Exception Description and Exception Location.

### • **System.out.println(e);**

If we pass Exception object reference variable as parameter to `System.out.println()` method then JVM will access Exception class `toString()` method internally, it will display the exception details like Exception name, Exception description.

### • **System.out.println(e.getMessage());**

Where `getMessage()` method will return a String contains the exception details like only Description of the exception.

EX:

```
1) class Test {  
2)     public static void main(String args[]) {  
3)         try {  
4)             throw new ArithmeticException("My Arithmetic Exception");  
5)         }  
6)         catch(ArithmeticException e) {  
7)             e.printStackTrace();  
8)             System.out.println();  
9)             System.out.println(e);  
10)            System.out.println();  
11)            System.out.println(e.getMessage());  
12)        }  
13)    finally {
```



```
14)    }
15)  }
16) }
```

## Output:

```
java.lang.ArithmaticException:My Arithmetic Exception
at Test.main(Test.java:7)
```

```
java.lang.ArithmaticException:My Arithmetic Exception
```

My Arithmetic Exception

Where the main purpose of finally block is to include some Java code , it will be executed irrespective of getting exception in "try" block and irrespective of executing "catch" block.

## Q) What is the difference between "final","finally" and "finalize" in JAVA?

- "final" is a keyword it can be used to declare constant expressions.  
There are three ways to use final keyword in java applications.

- a) **final variable:** It will not allow modifications over its value.
- b) **final methods:** It will not allow method overriding.
- c) **final class:** It will not allow inheritance, that is, sub classes.

- **finally block:** It is part of try-catch-finally syntax, it will include some instructions, which must be executed by JVM irrespective of getting exception from try block and irrespective of executing catch block.
- **finalize ():** It is a method in java.lang.Object class, it will be executed just before destroying objects in order to give final notification to the user about to destroy objects.

## Q) Find the output from the following programs?

```
1) class Test {
2)     public static void main(String args[]) {
3)         System.out.println("Before Try");
4)         try {
5)             System.out.println("Inside Try");
6)         }
7)         catch(Exception e) {
8)             System.out.println("Inside Catch");
9)         }
10)        finally {
```



```
11)         System.out.println("Inside Finally");
12)     }
13)         System.out.println("After Finally");
14)     }
15} }
```

## Output:

Before try  
Inside try  
Inside finally  
After finally

## EX:

```
1) class Test {
2)     public static void main(String args[]) {
3)         System.out.println("Before Try");
4)         try {
5)             System.out.println("Before Exception in try");
6)             float f = 100/0;
7)             System.out.println("After Exception in try");
8)         }
9)         catch(Exception e) {
10)             System.out.println("Inside Catch");
11)         }
12)         finally {
13)             System.out.println("Inside Finally");
14)         }
15)         System.out.println("After Finally");
16)     }
17} }
```

## Output:

Before try  
Before exception in try  
Inside catch  
Inside finally  
After finally



## **Q) Find The Output From The Following Program?**

```
1) class A {  
2)     int m10 {  
3)         try {  
4)             return 10;  
5)         }  
6)         catch(Exception e) {  
7)             return 20;  
8)         }  
9)         finally {  
10)            return 30;  
11)        }  
12)    }  
13)}  
14) class Test {  
15)     public static void main(String args[]) {  
16)         A a = new A();  
17)         int val = a.m10;  
18)         System.out.println(val);  
19)     }  
20)}
```

### **Output:**

30

**NOTE:** finally block provided return statement is the finally return statement for the method

## **Q) Is it possible to provide "try" Block without "catch" Block?**

Yes, it is possible to provide try block without catch block but by using "finally" Block.

### **Syntax:**

```
try {  
}  
finally {  
}
```



## EX:

```
1) class Test {  
2)     public static void main(String args[]) {  
3)         System.out.println("Before try");  
4)         try {  
5)             System.out.println("Before Exception inside try");  
6)             int i = 100;  
7)             int j=0;  
8)             float f = i/j;  
9)             System.out.println("After Exception inside try");  
10}        }  
11}        finally {  
12)            System.out.println("Inside finally");  
13}        }  
14)        System.out.println("After Finally");  
15}    }  
16}
```

Status: No Compilation Error.

## Output:

Before try

Before exception inside try

Inside finally

Exception in thread "main" java.lang.ArithmaticException:/by zero  
at Test.main(Test.java:11)

**REASON:** When JVM encounter exception in try Block, JVM will search for catch Block, if no catch block is identified, then JVM will terminate the program abnormally after executing finally block.

## Q) Is it possible to provide "try" Block without "finally" Block?

Yes, it is possible to provide "try" block without "finally" block but by using "catch" block.

## Syntax:

```
try {  
    -----  
    -----  
}  
catch(Exception e) {  
    -----  
    -----  
}
```



## Q) Is it possible to provide try-catch-finally

- a) inside try block,
- b) inside catch block and
- c) inside finally block

Yes, it is possible to provide try-catch-finally inside try block, inside catch block and inside finally block.

### Syntax-1:

```
try {  
    try {  
    }  
    catch(Exception e) {  
    }  
    finally {  
    }  
}  
catch(Exception e) {  
}  
finally {  
}
```

### Syntax-2:

```
try {  
}  
catch(Exception e) {  
    try {  
    }  
    catch(Exception e)  
    {  
    }  
    finally {  
    }  
}  
finally {  
}
```

### Syntax-3:

```
try {  
}  
catch(Exception e) {  
}  
finally {  
    try {  
    }
```



```
}
```

```
catch(Exception e) {
```

```
}
```

```
finally {
```

```
}
```

```
}
```

## **Q) Is it possible to provide more than one catch Block for a Single try Block?**

Yes, it is possible to provide more than one catch block for a single try block but with the following conditions.

- If no inheritance relation existed between exception class names which are specified along with catch blocks then it is possible to provide all the catch blocks in any order. If inheritance relation is existed between exception class names then we have to arrange all the catch blocks as per Exception classes inheritance increasing order.
- In general, specifying an exception class along with a catch block is not giving any guarantee to rise the same exception in the corresponding try block, but if we specify any pure checked exception along with any catch block then the corresponding "try" block must rise the same pure checked exception.

### **Ex 1:**

```
try {
```

```
}
```

```
catch(ArithmeticException e) {
```

```
}
```

```
catch(ClassCastException e) {
```

```
}
```

```
catch(NullPointerException e) {
```

```
}
```

**Status:Valid Combination**

### **Ex 2:**

```
try {
```

```
}
```

```
catch(NullPointerException e) {
```

```
}
```

```
catch(ArithmeticException e) {
```

```
}
```

```
catch(ClassCastException e) {
```

```
}
```

**status:Valid Combination**



**Ex 3:**

```
try {  
}  
catch(ArithmeticException e) {  
}  
catch(RuntimeException e) {  
}  
catch(Exception e) {  
}
```

Status:Valid

**Ex 4:**

```
try {  
}  
catch(Exception e) {  
}  
catch(RuntimeException e) {  
}  
catch(ArithmeticException e) {  
}
```

status: Invalid

**Ex 5:**

```
try {  
    throws new ArithmeticException("My Exception");  
}  
catch(ArithmeticException e) {  
}  
catch(IOException e) {  
}  
catch(NullPointerException e) {  
}
```

Status:Invalid

**Ex 6:**

```
try {  
    throw new IOException("My Exception");  
}  
catch(ArithmeticException e) {  
}  
catch(IOException e) {  
}
```



```
catch(NullPointerException e) {  
}
```

status:Valid

## **Custom Exceptions/User Defined Exceptions:**

Custom Exceptions are the exceptions, which would be defined by the developers as per their application requirements.

If we want to define user defined exceptions then we have to use the following steps:

- **Define User defined Exception Class:**

To declare user-defined Exception class, we have to take an user-defined class, which must be extended from java.lang.Exception class.

```
Class MyException extends Exception  
{  
}
```

- **Declare a String Parametrized Constructor in User-Defined Exception Class and Access String Parametrized Super Class Constructor by using "super" Keyword:**

```
class MyException extends Exception {  
    MyException(String err_Msg) {  
        super(err_Msg);  
    }  
}
```

## **Create and Rise Exception in Java Application as per the Application Requirement:**

```
1) try {  
2)     throw new MyException("My Custom Exception");  
3) }  
4) catch(MyException me) {  
5)     me.printStackTrace();  
6) }  
7)  
8) class InsufficientFundsException extends Exception  
9) {  
10)    InsufficientFundsException(String err_Msg)  
11)    {  
12)        super(err_Msg);  
13)    }  
14} 
```



```
15) class Account
16) {
17)     String accNo;
18)     String accName;
19)     String accType;
20)     int balance;
21)
22)     Account(String accNo, String accName, String accType, int balance)
23)     {
24)         this.accNo=accNo;
25)         this.accName=accName;
26)         this.accType=accType;
27)         this.balance=balance;
28)     }
29) }
30) class Transaction
31) {
32)     public void withdraw(Account acc, int wd_Amt)
33)     {
34)         try
35)         {
36)             System.out.println("Transaction Details");
37)             System.out.println("-----");
38)             System.out.println("Account Number :" + acc.accNo);
39)             System.out.println("Account Name   :" + acc.accName);
40)             System.out.println("Account Type  :" + acc.accType);
41)             System.out.println("Transaction Type :WITHDRAW");
42)             System.out.println("Withdraw Amount :" + wd_Amt);
43)             if(acc.balance>wd_Amt)
44)             {
45)                 acc.balance=acc.balance-wd_Amt;
46)                 System.out.println("Total Balance  :" + acc.balance);
47)                 System.out.println("Transaction Status:SUCCESS");
48)             }
49)             else
50)             {
51)                 System.out.println("Total Balance  :" + acc.balance);
52)                 System.out.println("Transaction Status:FAILURE");
53)                 throw new InsufficientFundsException("Funds are not Sufficient in
your Account, please enter valid Withdraw Amout");
54)             }
55)         }
56)         catch (InsufficientFundsException e)
57)         {
58)             System.out.println(e.getMessage());
```



```
59)    }
60)    finally
61)    {
62)        System.out.println("*****ThanQ, Visit Again*****");
63)    }
64) }
65)
66) class Test
67) {
68)     public static void main(String[] args)
69)     {
70)         Account acc1=new Account("abc123", "Durga", "Savings", 10000);
71)         Transaction tx1=new Transaction();
72)         tx1.withdraw(acc1, 5000);
73)         System.out.println();
74)         Account acc2=new Account("xyz123", "Anil", "Savings", 10000);
75)         Transaction tx2=new Transaction();
76)         tx2.withdraw(acc2, 15000);
77)     }
78} }
```

## JAVA 7 Features in Exception Handling:

- Multi Catch block
- try-with-Resources/Automatic Resources Management/Auto close able Resources

### **• Multi Catch Block:**

Consider the below Syntax

```
try {
}
catch(Exception e) {
```

If we specify "Exception" class along with catch block then it able to catch and handle all the exceptions which are either same as Exception or child classes to Exception, this approach will not handling exceptions individually, it will handle all the exceptions in the common way.

If we want to handle to the Exceptions separately then we have to use multiple catch blocks for a single try block.

```
try {
} catch(ArithmaticException e) {
} catch(NullPointerException e) {
} catch(ClassCastException e) {
```



}

If we use this approach then number of catch blocks are increased.

In Java applications, if we want to handle all the exceptions separately and by using a single catch block then we have to use "JAVA7" provided multi- catch block.

### Syntax:

```
try {  
}  
}  
catch(Exception1 | Exception2 |....|Exception-n e) {  
}
```

Where Exception1, Exception2....must not have inheritance relation otherwise Compilation error will be raised.

### EX:

```
1) class Test {  
2)     public static void main(String args[]) {  
3)         try {  
4)             /* int a = 10;  
5)             int b = 0;  
6)             float c = a/b;  
7)             */  
8)             /*java.util.Date d = null;  
9)             System.out.println(d.toString());  
10)            */  
11)             int[] a = {1, 2, 3, 4, 5};  
12)             System.out.println(a[10]);  
13)         }  
14)         catch(ArithmeticException | NullPointerException |  
15)             ArrayIndexOutOfBoundsException e) {  
15)             e.printStackTrace();  
16)         }  
17} }
```

## try-With-Resources/Auto Closeable Resources:

In general, in Java applications, we may use the resources like Files, Streams, Database Connections....as per the application requirements.

If we want to manage the resources along with try-catch-finally in Java applications then we have to use the following conventions.

- Declare all the resources before "try" block.
- Create the resources inside "try" block.
- Close the resources inside finally block.



---

The main intention to declare the resources before "try" block is to make available resources variables to "catch" block and to "finally" block to use.

If we want to close the resources in "finally" block then we have to use close() methods, which are throwing some exceptions like IOException, SQLException depending on the resource by using "throws" keyword, to handle these exceptions we have to use "try-catch-finally" inside "finally" block.

```
1) //Declare the resources
2) File f = null;
3) BufferedReader br = null;
4) Connection con=null;
5) try {
6)     //create the resources
7)     f = new File("abc.txt");
8)     br = new BufferedReader(new InputStreamReader(System.in));
9)     con = DriverManager.getConnection("jdbc:odbc:nag", "system", "durga");
10)    ----
11)    ----
12)
13) catch(Exception e) {
14)
15) finally {
16) //close the resources
17) try {
18)     f.close();
19)     br.close();
20)     con.close();
21) } catch(Exception e) {
22)     e.printStackTrace();
23) }
24)}
```

To manage the resources in Java applications, if we use the above convention then developers have to use close() methods explicitly, Developers have to provide try-catch-finally inside "finally" block, this convention will increase number of instructions in Java applications.

To overcome all the above problems, JAVA 7 version has provided a new Feature in the form of "Try-With-Resources" or "Auto Closeable Resources".

In the case of "Try-With-Resources", just we have to declare and create the resources along with "try"[not inside try block, not before try block] and no need to close these resources inside the finally block, why because, JVM will close all the resources automatically when flow of execution is coming out from "try" block.



In the case of "Try-With-Resources", it is not required to close the resources explicitly, it is not required to use close() methods in finally block explicitly, so that it is not required to use "finally" block in try-catch-finally syntax.

### Syntax:

```
try(Resource1; Resource2;.....Resource-n) {  
    -----  
    -----  
}  
catch(Exception e) {  
    -----  
    -----  
}
```

Where all the specified Resources classes or interfaces must implement "java.io.AutoCloseable" interface.

Where if we declare resources as AutoCloseable resources along with "try" then the resources reference variables are converted as "final" variables.

### EX:

```
1) try(File f = new File("abc.txt");  
2) BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
3) Connection con = DriverManager.getConnection("jdbc:odbc:nag", "system", "durga  
");  
4) {  
5)     -----  
6)     -----  
7) }  
8) catch(Exception e) {  
9)     e.printStackTrace();  
10} }
```

**NOTE:** In Java, all the predefined Stream classes, File class, Connection interface are extends/implemented "java.io.AutoCloseable" interface predefined.

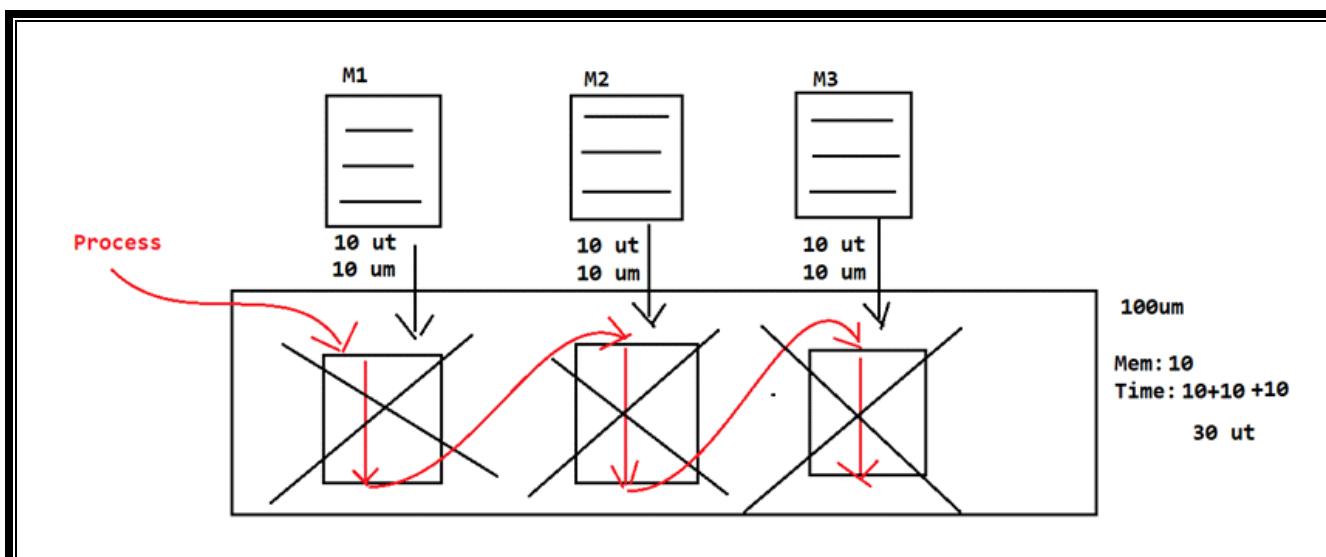


# Multi Threading



## Q) What is the difference between Process, Procedure and Processor?

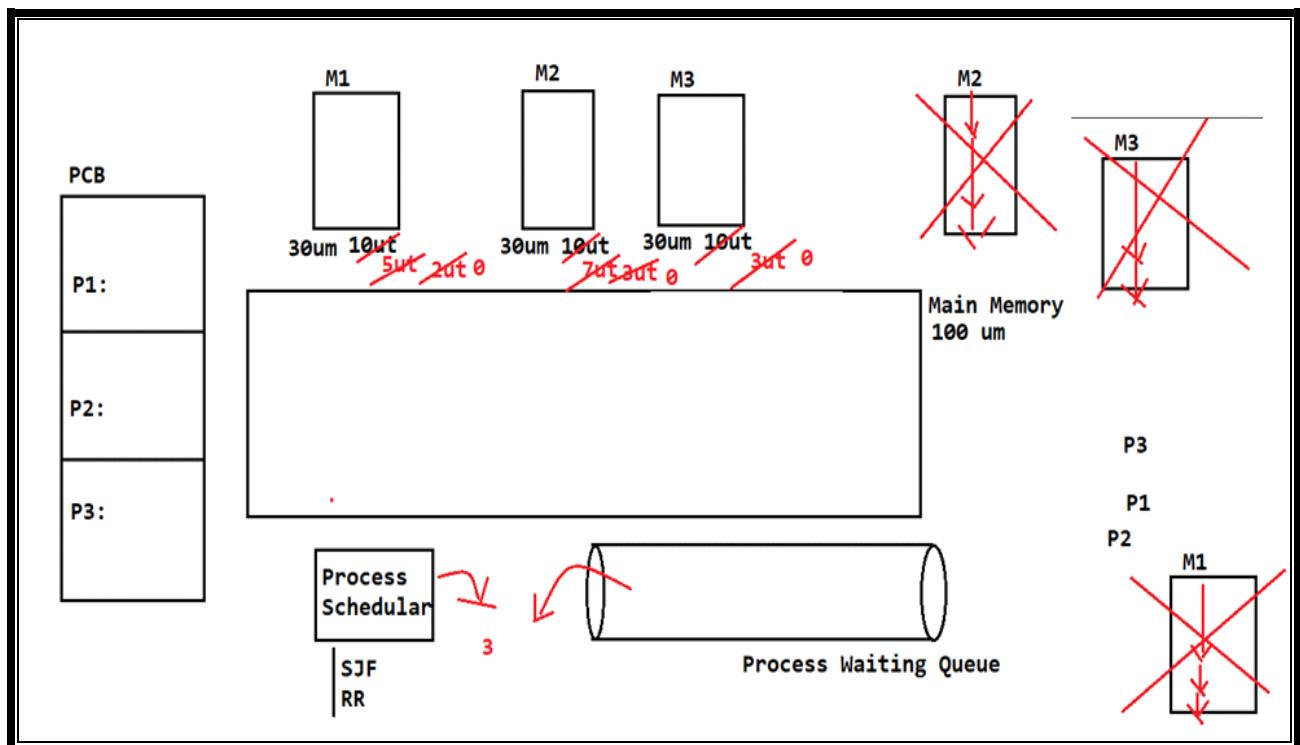
- Process is a flow of execution to perform a particular task.
- Procedure is a set of instructions to represent a particular task.
- Processor is an H/W component to generate no of processes in order to execute applications.
- At starting point of the computers we have Single Process Mechanism or Single Tasking to execute applications.
- In Single Process Mechanism, System is able to allow only one task at a time to load into the memory even our system main memory is capable to manage all the tasks.



- In Single Process mechanism, System is able to allow only one process to execute all the tasks which are available in our application, it will follow sequential kind of execution, it will increase application execution time and it will reduce application performance.
- To overcome the above problems we have to use Multi Process Mechanism or multi tasking. In Multi tasking system is able to allow to load more than one task at a time in main memory and it able to allow more than one process to execute application, it will follow parallel execution, it will reduce application execution time and it will improve application performance.



To execute applications by using Multi Tasking or Multi Process Mechanism we have to use the following components.



- **Main Memory:** To load all the tasks.
- **Process Waiting Queue:** To keep track of all process
- **Process Context Block:** To manage status of all the processes execution.
- **Process Scheduler:** It will take process from Process Waiting Queue and it will assign time stamps to each and every process in order to execute.

In the above multi processing system, controlling is switched from one process context to another process context called as "Context Switching".

There are two types of Context Switching's.

- **Heavy Weight Context Switching:**

It is the context switching between two heavy weight components, it will take more memory and more execution time, it will reduce application performance.

EX: Cotext switching between two Processes.

- **Light Weight Context Switching:**

It is the context switching between two light weight components, it will take less memory and less execution time and it will improve application performance.

EX: Context switching between two threads.



## **Q) What is the difference between Process and Thread?**

Process is heavy weight, to handle it System has to consume more memory and more execution time, it will reduce application performance.

Thread is light weight, to handle it system has to consume less memory and less execution time, it will improve application performance.

There are two thread models to execute applications.

- Single Thread Model
- Multi Thread Model

### **• Single Thread Model:**

It will allow only one thread to execute application, it will follow sequential execution, it will increase application execution time and it will reduce application performance.

### **• Multi Thread Model:**

It will allow more than one thread to execute application, it will follow parallel execution, it will reduce application execution time and it will improve application performance.

Java is following Multi Thread Model to execute applications and it will provide very good environment to create and execute more than one Thread at a time.

In java applications, to create Threads JAVA has provided the following predefined library in the form of `java.lang` package.

## **Q) What is Thread and in how many ways we are able to create Threads in Java?**

- Thread is a flow of execution to perform a particular task.
- As per the predefined library provided by JAVA, there are two ways to create threads in java applications.

### **• Extending Thread Class:**

In this approach, we have to declare a class, it must be extended from `java.lang.Thread` class.

```
class MyThread extends Thread  
{  
    --implementation---  
}
```



## • Implementing Runnable Interface:

In this approach, we have to declare a class, it must implement `java.lang.Runnable` Interface.

```
class MyThread implements Runnable
{
    ---implementation---
}
```

## Threads Design in Java:

There are two approaches to create threads in java applications.

- Extending Thread Class
- Implementing Runnable Interface

### 1) Extending Thread class

- Declare a user defined class.
- Extend `java.lang.Thread` class to user defined class
- Override `Thread` class `run()` method in user defined thread class with the implementation representing a particular task which we want to perform by creating a thread.
- In main class, in `main()` method, create object for user defined class.
- Access `Thread` class provided `start()` method on user defined thread class object reference variable.

The main intention of `start()` method is to create new thread and to access `run()` method by passing the generated thread.

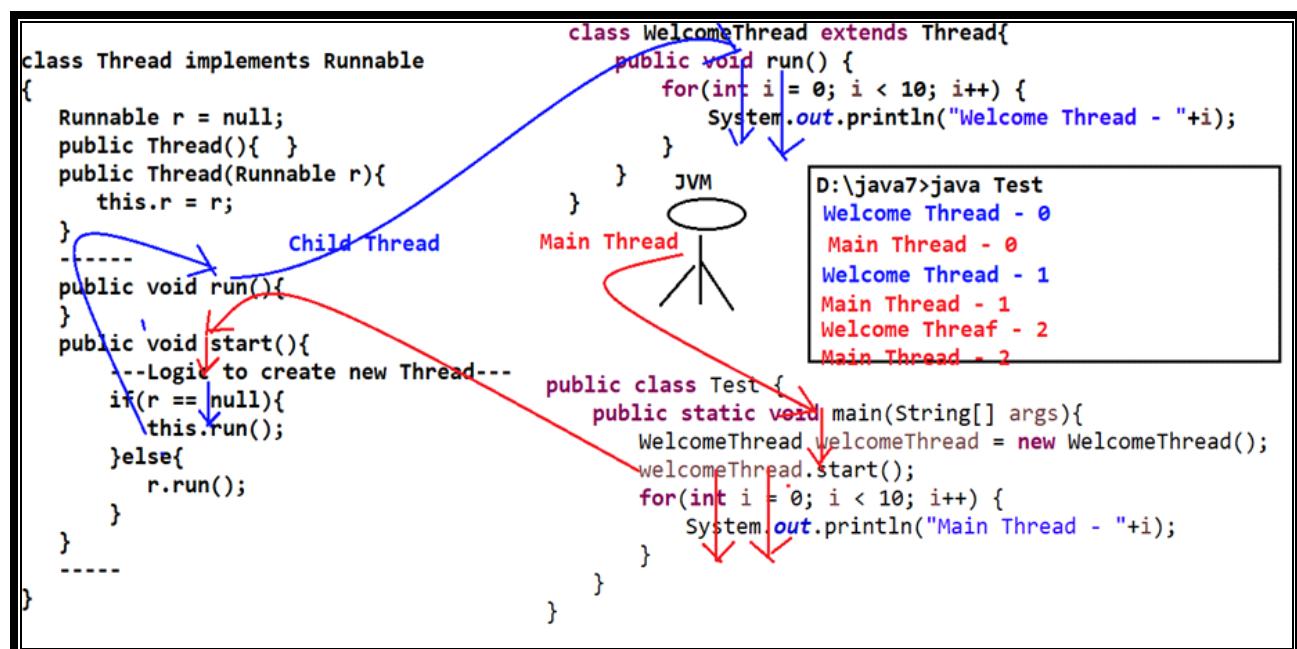
```
public void start()
```

#### EX:

```
1) class MyThread extends Thread
2) {
3)     public void run()
4)     {
5)         for(int i=0;i<10;i++)
6)         {
7)             System.out.println("User Thread :" +i);
8)         }
9)     }
10)
11) class Test
12) {
13)     public static void main(String[] args)
```



```
14) {
15)     MyThread mt=new MyThread();
16)     mt.start();
17)     for(int i=0;i<10;i++)
18)     {
19)         System.out.println("Main Thread :"+i);
20)     }
21) }
22}
```



Q) In Java applications, to create threads we have already first approach [Extending thread class] then what is the requirement to go for Second Approach [implementing Runnable interface]?

In java applications, to create threads if we use first approach then we have to declare an user defined class and it must be extended from `java.lang.Thread` class, in this context, it is not possible to extend other classes , if we extend any other class like `Frame`,.. along with `Thread` class then it will represent Multiple Inheritance, it is not possible in Java.

```
class MyClass extends Frame, Thread {
    ...
}
```

To overcome the above problem, we have to use second approach to create Thread, that is, implementing Runnable Interface.



```
class MyClass extends Frame implements Runnable{  
---  
}
```

## 2) Implementing Runnable Interface:

- Declare a user defined class.
- Implement `java.lang.Runnable` interface.
- Provide implementation part in `run()` method which we want to execute by creating a thread.
- In main class, in `main()` method, create a thread and access user defined thread class `run()` method.

To perform the above step we have to use the following cases.

```
class MyThread implements Runnable {  
    public void run() {  
        ---  
    }  
}
```

### Case-1:

```
MyThread mt = new MyThread();  
mt.start();
```

Status: Compilation Error.

Reason: `start()` method was not declared in `MyThread` class and in its super class `java.lang.Object` class,  
`start()` method is existed in `java.lang.Thread` class.

### Case-2:

```
MyThread mt = new MyThread();  
mt.run();
```

Status: No Compilation Error, but, only Main thread access `MyThread` class `run()` method like a normal Java method, no multi threading environment.

### Case-3:

```
MyThread mt = new MyThread();  
Thread t = new Thread();  
t.start();
```

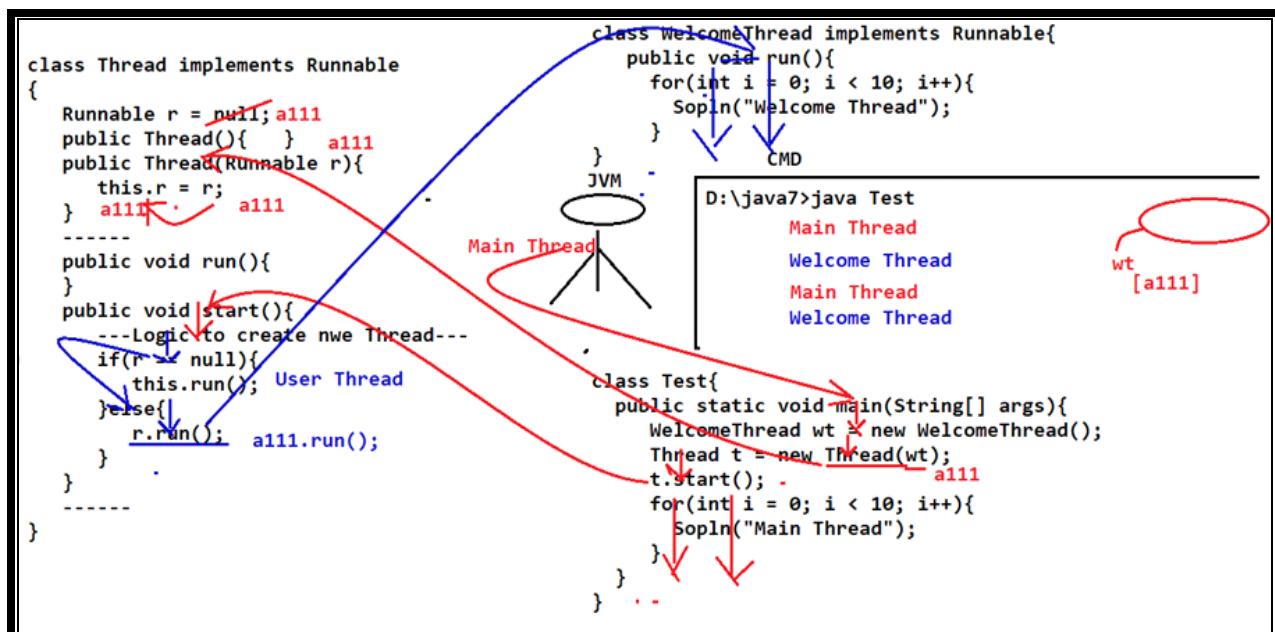
Status: No Compilation Error, `start()` method creates new thread and it access `Thread` class `run()` method, not `MyThread` class `run()` method.



## Case-4:

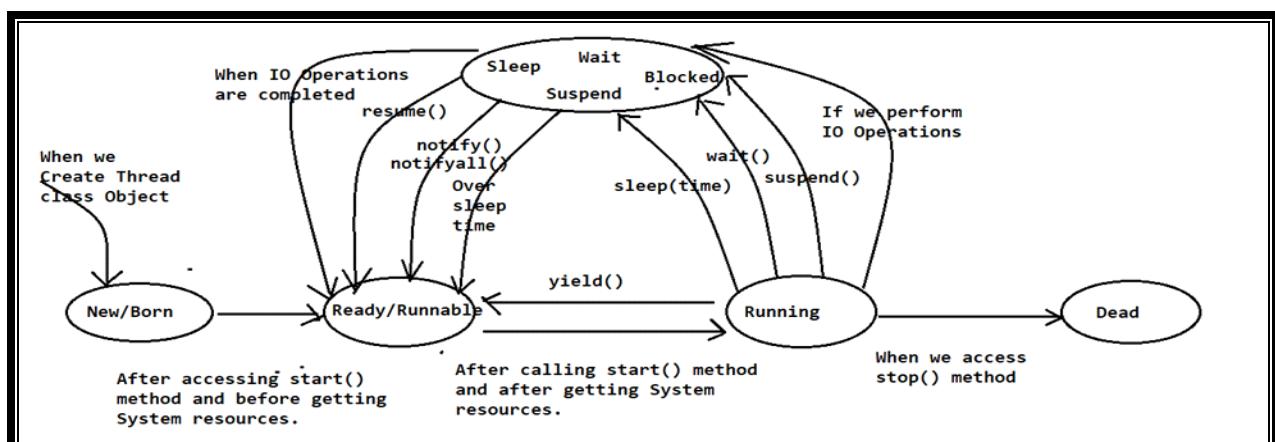
```
MyThread mt = new MyThread();
Thread t = new Thread(mt);
t.start();
```

Status: No Compilation Error, start() method creates new thread and it will bypass new thread to MyThread class run() method.



## Thread Lifecycle:

- The collective information of a thread right from its starting point to ending point is called as "Thread Life Cycle".
- In Java applications, Threads are able to have the following states as part of their lifecycle.





- **New/Born State:**

When we create Thread class object in java applications then Thread will come to New/Born state.

- **Ready/Runnable State:**

When we access start() method Thread Scheduler has to assign system resources like memory and time, here before assigning system resources and after calling start() method is called as Ready/Runnable state.

- **Running State:**

In Java applications, after calling start() method and after getting system resources like memory and execution time is called as "Running State".

**NOTE:** We can send a thread from Running state to Runnable state directly by accessing yield() method, but, it is not supported by Windows operating system, because, it will perform its functionality on the basis of Threads priority values, priority based operations are not supported by windows operating system.

- **Dead/Destroy State:**

In Java applications, when we access stop() method over Running thread then that thread will come to Dead/Destroy state.

- **Blocked State:**

In Java applications, we are able to keep a thread in Blocked state from Running state in the following situations.

- a) When we access sleep(--) method with a particular sleep time.
- b) When we access wait() method.
- c) When we access suspend() method.
- d) When we perform IO Operations.

In Java applications, we are able to bring a thread from Blocked state to Ready / Runnable state in the following situations.

- a) When sleep time is over.
- b) If any other thread access notify() / notifyAll() methods.
- c) If any other thread access resume() method.
- d) When IO Operations are completed.



## Thread Class Library:

### Constructors:

- **public Thread()**

This constructor can be used to create thread class object with the following properties.

Thread Name: Thread-0

Thread Priority: 5

Thread group Name : main

**EX:** `Thread t = new Thread();  
System.out.println(t);`

**Output:** `Thread[Thread-0, 5, main]`

- **public Thread(String name)**

This constructor can be used to create Thread class object with the specified name.

**EX:** `Thread t = new Thread("Core Java");  
System.out.println(t);`

**Output:** `Thread[Core Java,5,main]`

- **public Thread(Runnable r)**

This constructor can be used to create Thread class object with the specified Runnable reference.

**EX:** `Runnable r = new Thread();  
Thread t = new Thread(r);  
System.out.println(t);`

**Output:** `Thread[Thread-1,5,main]`



- **public Thread(Runnable r, String name)**

This constructor can be used to create Thread class object with the specified Runnable reference and with the specified name.

**EX:** `Runnable r = new Thread();  
Thread t = new Thread(r, "Core Java");  
System.out.println(t);`

**Output:** `Thread[Core Java,5,main]`

- **public Thread(ThreadGroup tg, Runnable r)**

This constructor can be used to create Thread class object with the specified ThreadGroup name and with the specified thread name.

**NOTE:** To provide ThreadGroup name we have to use a predefined class like `java.lang.ThreadGroup`, to create ThreadGroup class object we have to use the following Constructor.

`public ThreadGroup(String name)`

**EX:** `ThreadGroup tg = new ThreadGroup("Java");  
Runnable r = new Thread();  
Thread t = new Thread(tg, r)  
System.out.println(t);`

**Output:** `Thread[Thread-1,5,Java]`

- **public Thread(ThreadGroup tg, String name)**

This constructor can be used to create Thread class object with the specified ThreadGroup name and with the specified Thread name.

**EX:** `ThreadGroup tg = new ThreadGroup("Java");  
Thread t = new Thread(tg, "Core Java");  
System.out.println(t);`

**Output:** `Thread[Core Java,5,Java]`



## public Thread(ThreadGroup tg, Runnable r, String name)

This constructor can be used to create Thread class object with the specified ThreadGroup name, with the Runnable reference and with the thread name.

**EX:** ThreadGroup tg = new ThreadGroup("Java");  
Runnable r = new Thread();  
Thread t = new Thread(tg, r, "Core Java");

**Output:** Thread[Core Java, 5, Java]

## Methods:

- **public void setName(String name)**  
It can be used to set a particular name to the Thread explicitly.
- **public String getName()**  
It can be used to get thread name explicitly.

**EX:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         Thread t=new Thread();
6)         System.out.println(t.getName());
7)         t.setName("Core Java");
8)         System.out.println(t.getName());
9)     }
10) }
```

## public void setPriority(int priority)

It can be used to set a particular priority value to the Thread, but, here the priority value must be provided in the range from 1 to 10, if we provide any other value then JVM will rise an exception like `java.lang.IllegalArgumentException`.

To represent Thread priority values, `java.lang.Thread` class has provided the following constants.

```
public static final int MIN_PRIORITY=1;
public static final int NORM_PRIORITY=5;
public static final int MAX_PRIORITY=10;
```



- **public int getPriority()**

It can be used to get priority value of the Thread.

**EX:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         Thread t=new Thread();
6)         System.out.println(t.getPriority());
7)         t.setPriority(7);
8)         System.out.println(t.getPriority());
9)         t.setPriority(Thread.MAX_PRIORITY-2);
10)        System.out.println(t.getPriority());
11)        //t.setPriority(15);-->IllegalArgumentException
12)    }
13) }
```

### **public static int activeCount()**

It will return the no of threads which are in active.

**EX:**

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         Thread t=new Thread();
6)         t.start();
7)         System.out.println(Thread.activeCount());
8)     }
9) }
```

### **public boolean isAlive()**

This method can be used to check whether a thread is in live or not.

**EX:**

```
1) class Test {
2)     public static void main(String[] args) {
3)         Thread t=new Thread();
4)         System.out.println(t.isAlive());
5)         t.start();
6)         System.out.println(t.isAlive());
7)     }
8) }
```



## public static thread currentThread()

It can be used to get Thread object reference which is in active at present.

EX:

```
1) class MyThread extends Thread
2) {
3)     public void run()
4)     {
5)         for(int i=0;i<10;i++)
6)         {
7)             System.out.println(Thread.currentThread().getName());
8)         }
9)     }
10)
11) class Test
12) {
13)     public static void main(String[] args)
14)     {
15)         MyThread mt1=new MyThread();
16)         MyThread mt2=new MyThread();
17)         MyThread mt3=new MyThread();
18)
19)         mt1.setName("AAA");
20)         mt2.setName("BBB");
21)         mt3.setName("CCC");
22)
23)         mt1.start();
24)         mt2.start();
25)         mt3.start();
26)     }
27) }
```

## public static void sleep(long time) throws InterruptedException

- This method can be used to keep a running thread into sleeping state up to the specified sleep time.
- In general, we will use sleep() method in run() method of user defined thread class, where to handle InterruptedException we must use try-catch-finally syntax only, we must not use "throws" keyword in
- run() method prototype, because, we are overriding Thread class or Runnable interface predefined run() method.



- **public void join()throws InterruptedException**

This method will pause a thread to complete a thread on which we accessed join() method , after completion of the respective thread, paused thread will continue its execution part automatically.

## **Daemon Threads**

These threads are running internally to provide services to some other thread and it will be terminated along with the threads which are taking services.

To make a thread as daemon thread we have to use the following method.

### **public void setDaemon(boolean b)**

If 'b' value is true then thread will be daemon thread.

If 'b' value is false then thread will not be daemon thread.

**EX:** mt.setDaemon(true);

To check whether a thread is daemon thread or not we have to use the following method.

### **public boolean isDaemon()**

**EX:** In Java, Garbage Collector is a thread running internally inside JVM and it will provide Garbage Collection services to JVM and it will be terminated along with JVM automatically.

## **Synchronization:**

In Java applications, if we execute more than one thread on a single data item then there may be a chance to get data inconsistency, it may generate wrong results in java applications.

In java applications, to provide data consistency in the above situation we have to use "Synchronization".

"Synchronization" is a mechanism, it able to allow only one thread at a time , it will not allow more than one at a time, it able to allow other threads after completion of the present thread.

In java applications, synchronization is going on the basis of Locking mechanisms, If we send multiple threads at a time to synchronized area then Lock Manager will assign lock to a thread which is having highest priority, once a thread gets lock from Lock manager then that thread is eligible to enter in synchronized area, once a thread is available in



synchronized area then Lock Manager will not assign lock to other threads, when Thread completes its execution in synchronized area then that thread has to submit lock back to Lock Manager, once Lock is given back to Lock manager then Lock Manager will assign that lock to another thread which is having next priority.

In java applications, to provide synchronization JAVA has provided a keyword in the form of "synchronized".

In java applications, we are able to achieve "synchronization" in the following two ways.

- synchronized method
- synchronized block

## Synchronized method:

It is a normal java method, it will allow only one thread at a time to execute instructions, it will not allow more than one thread at a time, it will allow other threads after completion of the present thread execution.

EX:

```
1) class A
2) {
3)     synchronized void m10
4)     {
5)         for(int i=0;i<10;i++)
6)         {
7)             String thread_Name=Thread.currentThread().getName();
8)             System.out.println(thread_Name);
9)         }
10)    }
11) }
12) class MyThread1 extends Thread
13) {
14)     A a;
15)     MyThread1(A a)
16)     {
17)         this.a=a;
18)     }
19)     public void run()
20)     {
21)         a.m10();
22)     }
23) }
24) class MyThread2 extends Thread
25) {
26)     A a;
```



```
27) MyThread2(A a)
28) {
29)     this.a=a;
30) }
31) public void run()
32) {
33)     a.m1();
34) }
35)
36) class MyThread3 extends Thread
37) {
38)     A a;
39)     MyThread3(A a)
40)     {
41)         this.a=a;
42)     }
43)     public void run()
44)     {
45)         a.m1();
46)     }
47)
48) class Test
49) {
50)     public static void main(String[] args)
51)     {
52)         A a=new A();
53)         MyThread1 mt1=new MyThread1(a);
54)         MyThread2 mt2=new MyThread2(a);
55)         MyThread3 mt3=new MyThread3(a);
56)
57)         mt1.setName("AAA");
58)         mt2.setName("BBB");
59)         mt3.setName("CCC");
60)
61)         mt1.start();
62)         mt2.start();
63)         mt3.start();
64)     }
65) }
```



**Q) In Java applications, we have already synchronized methods to achieve synchronization then what is the requirement to use synchronized block?**

In java applications, if we use synchronized method to achieve synchronization then it will provide synchronization throughout the method irrespective of the actual requirement. If we need synchronization up to a block inside the synchronized method then it will provide unnecessary synchronization for the remaining part of the method, it will increase execution time and it will reduce application performance.

In the above context, to provide synchronization up to the required part then we have to use synchronized block.

- **Synchronized Block:**

It is a set of instructions, it able to allow only one thread at a time to execute instructions, it will not allow more than one thread at a time, it will allow other threads after completion of the present thread execution.

**Syntax:**

```
synchronized(Object o)
{
    ----
    ----
}
```

**EX:**

```
1) class A
2) {
3)     void m1()
4)     {
5)         String thread_Name=Thread.currentThread().getName();
6)         System.out.println("Before Synchronized Block :" +thread_Name);
7)         synchronized(this)
8)         {
9)             for(int i=0;i<10;i++)
10)            {
11)                String thread_Name1=Thread.currentThread().getName();
12)                System.out.println("Inside Synchronized Block :" +thread_Name1);
13)            }
14)        }
15)    }
16) }
17) class MyThread1 extends Thread
18) {
19)     A a;
20)     MyThread1(A a)
```



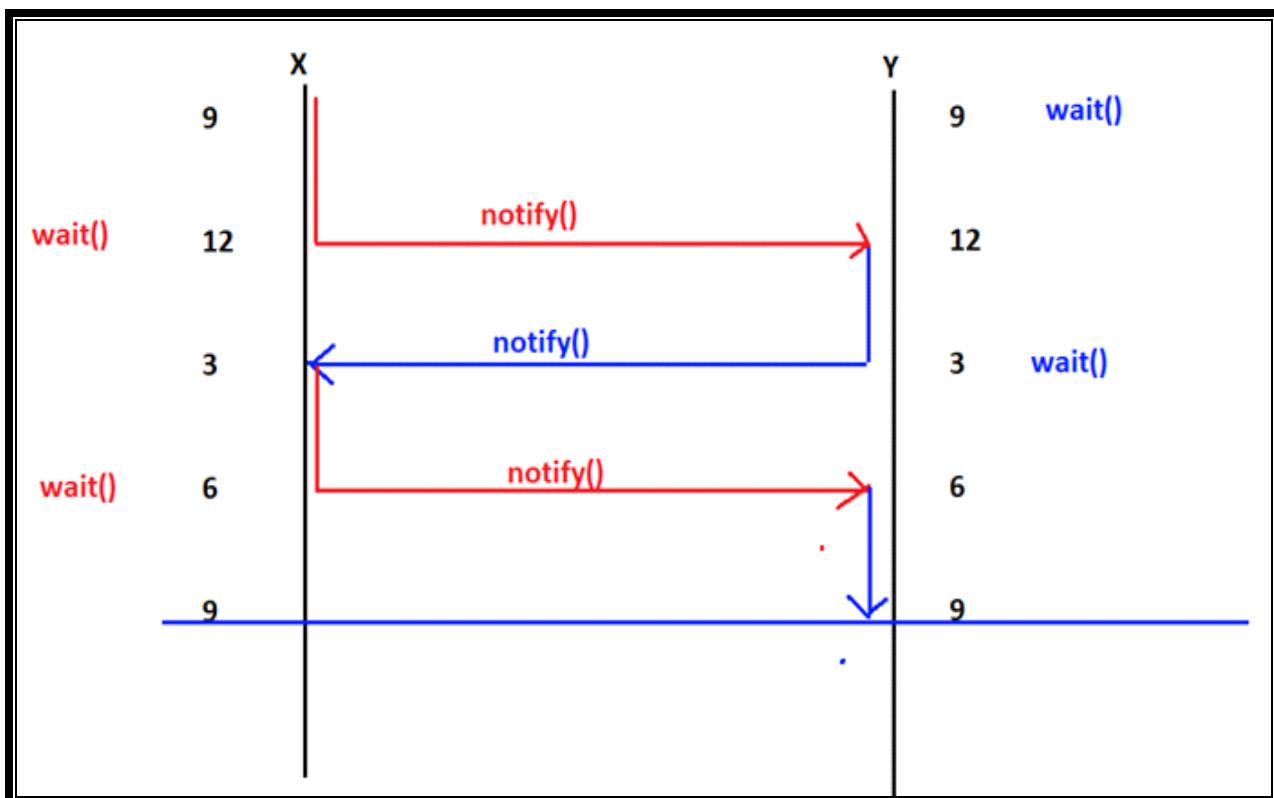
```
21) {
22)     this.a=a;
23) }
24) public void run()
25) {
26)     a.m1();
27) }
28) }
29) class MyThread2 extends Thread
30) {
31)     A a;
32)     MyThread2(A a)
33)     {
34)         this.a=a;
35)     }
36)     public void run()
37)     {
38)         a.m1();
39)     }
40) }
41) class MyThread3 extends Thread
42) {
43)     A a;
44)     MyThread3(A a)
45)     {
46)         this.a=a;
47)     }
48)     public void run()
49)     {
50)         a.m1();
51)     }
52) }
53) class Test
54) {
55)     public static void main(String[] args)
56)     {
57)         A a=new A();
58)         MyThread1 mt1=new MyThread1(a);
59)         MyThread2 mt2=new MyThread2(a);
60)         MyThread3 mt3=new MyThread3(a);
61)
62)         mt1.setName("AAA");
63)         mt2.setName("BBB");
64)         mt3.setName("CCC");
65) }
```



```
66)    mt1.start();
67)    mt2.start();
68)    mt3.start();
69) }
70) }
```

## Inter Thread Communication:

The process of providing communication between more than one thread is called as "Inter Thread Communication".



To perform Inter Thread Communication we have to use the following methods.

- `wait()`
- `notify()`
- `notifyAll()`

Where `wait()` method can be used to keep a thread in waiting state.

Where `notify()` method can be used to give a notification to a thread which is available in waiting state.

Where `notifyAll()` method can be used to give a notification to all the threads which are available in waiting state.



The above methods are provided by JAVA in `java.lang.Object` class.  
If we want to use these methods in java applications then we must provide "Synchronization".

In general, Inter Thread Communication will provide solutions for the problems like "Producer-Consumer" problems.

In Producer-Consumer problem, producer and consumer are two threads, where producer has to produce an item and consumer has to consume that item, the same sequence has to be provided infinite no of times, where Producer must not produce an item without consuming previous item by consumer and consumer must not consume an item without producing that item by producer.

EX:

```
1) class A
2) {
3)     boolean flag=true;
4)     int count=0;
5)     public synchronized void produce()
6)     {
7)         try
8)         {
9)             while(true)
10)            {
11)                if(flag == true)
12)                {
13)                    count=count+1;
14)                    System.out.println("Producer Produced Item"+count);
15)                    flag=false;
16)                    notify();
17)                    wait();
18)                }
19)                else
20)                {
21)                    wait();
22)                }
23)            }
24)        }
25)        catch (Exception e)
26)        {
27)            e.printStackTrace();
28)        }
29)    }
30)    public synchronized void consume()
31)    {
```



```
32) try
33) {
34)     while(true)
35)     {
36)         if(flag == true)
37)         {
38)             wait();
39)         }
40)         else
41)         {
42)             System.out.println("Consumer Consumed Item"+count);
43)             flag=true;
44)             notify();
45)             wait();
46)         }
47)     }
48) }
49) catch (Exception e)
50) {
51)     e.printStackTrace();
52) }
53) }
54)
55) class Producer extends Thread
56) {
57)     A a;
58)     Producer(A a)
59)     {
60)         this.a=a;
61)     }
62)     public void run()
63)     {
64)         a.produce();
65)     }
66)
67) class Consumer extends Thread
68) {
69)     A a;
70)     Consumer(A a)
71)     {
72)         this.a=a;
73)     }
74)     public void run()
75)     {
76)         a.consume();
77)     }
78)
79) }
```



```
77) }
78) }
79) class Test
80) {
81)     public static void main(String[] args)
82)     {
83)         A a=new A();
84)         Producer p=new Producer(a);
85)         Consumer c=new Consumer(a);
86)         p.start();
87)         c.start();
88)     }
89) }
```

## Dead Lock:

Dead Lock is a situation, where more than one thread is depending on each other in circular dependency.

In java applications, once we are getting deadlock then program will stuck in the middle, so that, it will not have any recovery mechanisms, it will have only prevention mechanisms.

EX:

```
1) class Register_Course extends Thread
2) {
3)     Object course_Name;
4)     Object faculty_Name;
5)     Register_Course(Object course_Name, Object faculty_Name)
6)     {
7)         this.course_Name=course_Name;
8)         this.faculty_Name=faculty_Name;
9)     }
10)    public void run()
11)    {
12)        synchronized(course_Name)
13)        {
14)            System.out.println("Register_Course Thread holds course_Name resource and
waiting for faculty_Name resource.....");
15)            synchronized(faculty_Name)
16)            {
17)                System.out.println("Register_Course is success, because, Register_Course thread holds both course_Name and faculty_Name resources");
18)            }
19)        }
```



```
20) }
21) }
22) class Cancel_Course extends Thread
23) {
24)     Object course_Name;
25)     Object faculty_Name;
26)     Cancel_Course(Object course_Name, Object faculty_Name)
27)     {
28)         this.course_Name=course_Name;
29)         this.faculty_Name=faculty_Name;
30)     }
31)     public void run()
32)     {
33)         synchronized(faculty_Name)
34)         {
35)             System.out.println("Cancel_Course Thread holds faculty_Name resource and
waiting for course_Name resource.....");
36)             synchronized(course_Name)
37)             {
38)                 System.out.println("Cancel_Course is success, because, Cancel_Course thread
holds both faculty_Name and course_Name resources");
39)             }
40)         }
41)     }
42) }
43)
44) class Test
45) {
46)     public static void main(String[] args)
47)     {
48)         Object course_Name=new Object();
49)         Object faculty_Name=new Object();
50)         Register_Course rc=new Register_Course(course_Name, faculty_Name);
51)         Cancel_Course cc=new Cancel_Course(course_Name, faculty_Name);
52)         rc.start();
53)         cc.start();
54)     }
55) }
```



# IO Streams



In any programming language, in any application, providing input to the Applications and getting output from the Applications is essential.

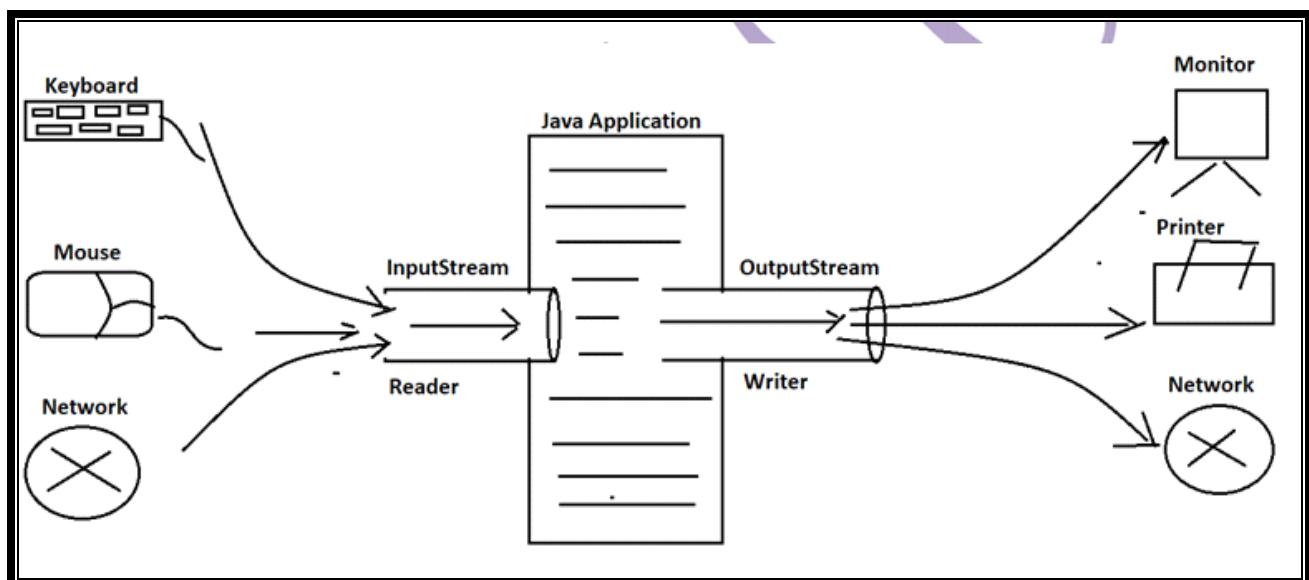
In case of C and C++ applications, we are able to perform input and output operations by using some predefined library in the form of `printf()`, `scanf()`, `cin>>`, `cout<<,.....`

Similarly in Java Applications, to perform input and output operations we have to use streams.

Java has represented all the streams in the form of predefined classes in "java.io" package.

## Stream:

Stream is medium or channel; it will allow the data in continuous flow from input devices to java program and from Java program to output devices.



In Java IOStreams are divided into following ways:

- Byte oriented Streams.
- Character-Oriented Streams

### Byte-Oriented Streams:

These are Streams, which will allow the data in the form of bytes from input devices to Java program and from java program to output devices.

The length of the data in byte-oriented streams is 1 byte.

There are two types of Byte-Oriented Streams

- `InputStream`
- `OutputStream`



- **InputStream:**

It is a byte-oriented Stream, it will allow data in the form of bytes from input devices to Java Applications.

EX:

ByteArrayInputStream  
FilterInputStream  
DataInputStream  
ObjectInputStream  
FileInputStream  
StringBufferInputStream  
BufferedInputStream....

- **OutputStream:**

It is a byte-oriented Stream, it will allow the data in the form of bytes from Java applications to output devices.

EX:

ByteArrayOutputStream  
FilterOutputStream  
DataOutputStream  
FileOutputStream  
PrintStream  
BufferedOutputStream..

**NOTE:** All the ByteOrientedStream classes are terminated with "Stream" word.

**NOTE:** The length of data items in Byte Oriented Streams is 1 byte.

- **Character-Oriented Streams:**

These are the Streams, which will allow the data in the form of characters from input devices to java program and form java program to output devices.

There are two bytes of character-oriented streams

- Reader
- Writer



## **Reader:**

It is a character-oriented stream, it will allow the data in the form of characters from input devices to java program.

**EX:**

CharArrayReader  
FilterReader  
BufferedReader  
FileReader  
InputStreamReader....

## **Writer:**

It is a character-oriented stream, it will allow the data in the form of characters from java program to output devices.

**EX:**

CharArrayWriter  
FilterWriter  
FileWriter  
PrintWriter  
BufferedWriter....

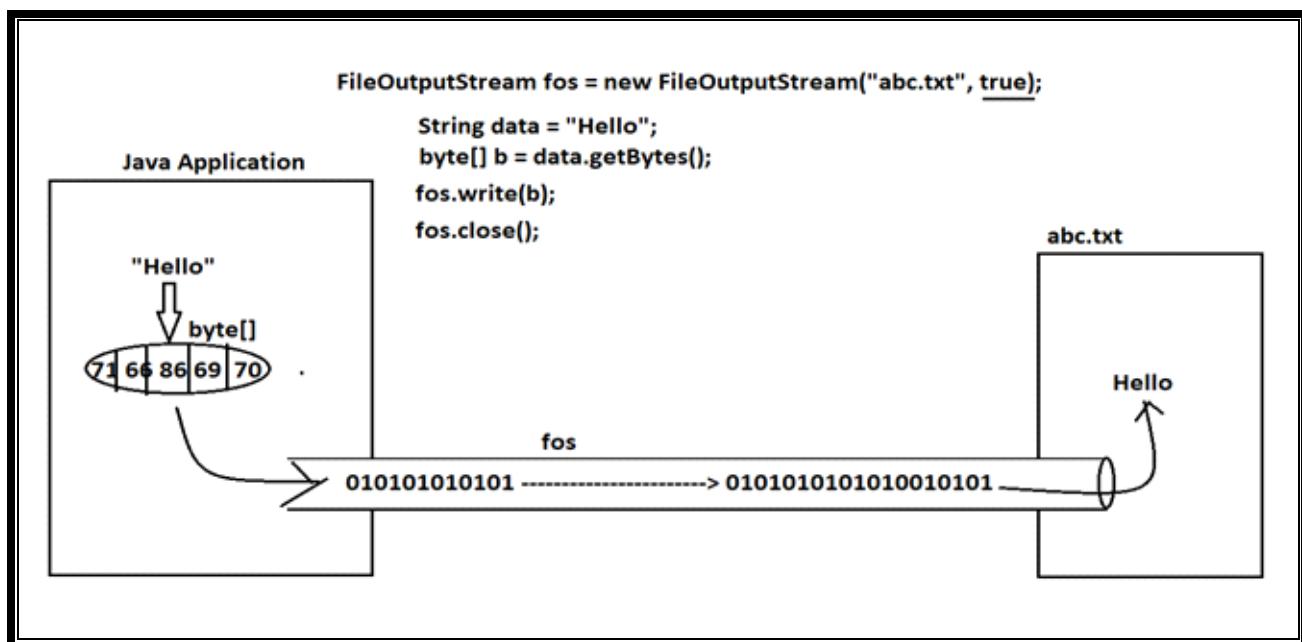
**NOTE:** All the predefined Classes of character-oriented streams are terminated with either Reader or Writer.

**NOTE:** The length of the data in characters-oriented stream is 2 bytes.

## **FileOutPutStream:**

It is byte-oriented Stream, it can be used to transfer the data from Java program to a particular target file.

To transfer the data from Java program to a particular target file by using FileOutPutstream we have to use the following Steps.



- **Create FileOutPutStream between Java program and target file:**

If we want to create `FileOutPutStream` class object then we have to use the following constructors

```
public FileOutPutStream(String target_File)
public FileOutPutStream(String target_File,boolean b)
```

EX:

```
FileOutPutStream fos = new FileOutPutStream("abc.txt");
```

It will override the existed data in the target file at each and every write operation.

```
FileOutPutStream fos=new FileOutPutStream("abc.txt",true);
```

It will not override the existed data in the target file, it will append the specified new data to the existed data in the target file.

When JVM encounter the above instruction, JVM will perform the following tasks.

- JVM will take the specified target file.
- JVM will search for the specified target file at the respective location.
- If the specified target file is available then JVM will establish `FileOutPutStream` from java program to target file.
- If the specified target file is not available then JVM will create a file with the target file name and establish `FileOutPutStream` from Java program to target file.

- **Declare the data and convert into byte[]:**

```
String data = "Hello";
byte[] b = data.getBytes();
```



- **Write Byte Array data into FileOutputStream:**

To write byte[] data into FileOutputStream, we have to use the following method.

public void write(byte[] b) throws IOException

**EX:**

fos.write(b);

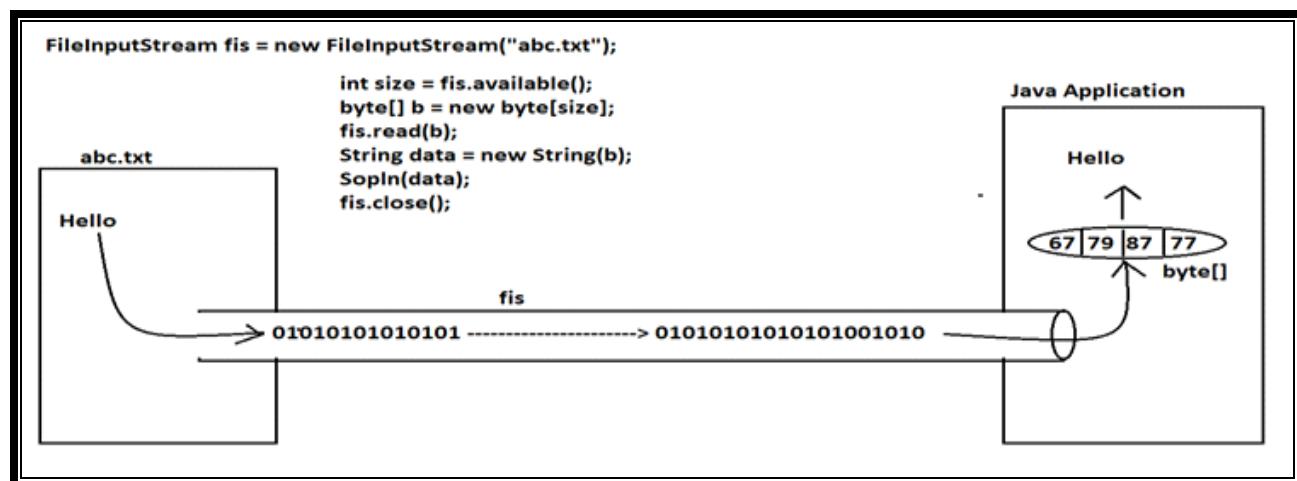
4) Close FileOutputStream:

fos.close();

## 2) FileInputStream:

It is a byte-oriented Stream, it can be used to transfer the data from a particular source file to Java Program.

If we want to transfer the data from source file to java program by using FileInputStream,



We have to use the following Steps:

- **Create FileInputStream class Object:**

To create FileInputStream class object, we have to use the following constructor from java.io.FileInputStream class.

public FileInputStream(String file\_name) throws FileNotFoundException

**EX:** FileInputStream fis = new FileInputStream("abc.txt");

When JVM encounter the above instruction then JVM will perform the following actions.

- JVM will take the specified source file name.
- JVM will search for the specified source file at the respective location.
- If the source file is not available at the respective location then JVM will raise an exception like "java.io.FileNotFoundException".
- If the required source file is available then JVM will establish FileInputStream from source file to JAVA program.



- After creating FileInputStream, JVM will transfer the data from source file to FileInputStream in the form bytes.
- **Get the Size of the Data from FileInputStream and prepare byte[] with the Data Size:**

To get the size of the data from FileInputStream, we have to use the following method  
public int available()

EX: int size = fis.available();  
byte[] b = new byte[size];

- **Read the data from FileInputStream into byte[]:**

To read the data from FileInputStream into byte[], we have to use the following method.

public void read(byte[] b) throws IOException

EX: fis.read(b);

- **Convert data from byte[] to String:**

String data = new String(b);  
System.out.println(data);

## Close FileInputStream:

fis.close();

## Write a Java program to display particular file content on command prompt by taking filename as command line input?

```
1) import java.io.*;  
2) class DisplayEx {  
3)     public static void main(String args[]) throws Exception {  
4)         String file_Name = args[0];  
5)         FileInputStream fis = new FileInputStream(file_Name);  
6)         int size = fis.available();  
7)         byte b[] = new byte[size];  
8)         fis.read();  
9)         String data = new String(b);  
10)        System.out.println(data);  
11)        fis.close();  
12)    }  
13) }
```



## Write a Java program to count no of words available in a particular text file and how many times the word "Durga" is repeated?

```
1) import java.io.*;
2) import java.util.*;
3) class Word_Count_Ex {
4)     public static void main(String args[]) throws Exception {
5)         FileInputStream fis = new FileInputStream("abc.txt");
6)         int size = fis.available();
7)         byte b[] = new byte[size];
8)         fis.read();
9)         String data = new String(b);
10)        StringTokenizer st = new StringTokenizer(data);
11)        int tokens = st.countTokens();
12)        System.out.println("No of words :" +tokens);
13)        int count = 0;
14)        while(st.hasMoreTokens()) {
15)            String token = st.nextToken();
16)            if(token.equals("Durga")){
17)                count = count+1;
18)            }
19)        }
20)        System.out.println("Durga' is repeated :" +count);
21)        fis.close();
22)    }
23} }
```

## Write a Java program to copy an image from a source file to a particular target file?

```
1) import java.io.*;
2) public class Image_Copy_Ex {
3)     public static void main(String args[]) {
4)         FileInputStream fis = new FileInputStream();
5)         int size = fis.available();
6)         byte[] b = new byte[size];
7)         fis.read(b);
8)         FileOutputStream fos = new FileOutputStream("abc.jpg");
9)         fos.write(b);
10)        fis.close();
11)        fos.close();
12)    }
13} }
```



## FileWriter:

This character-oriented Stream can be used to transfer the data from Java Application to a particular target File.

If we want to transfer the data from java applications to a particular target file by using

FileWriter then we have to use the following steps:

- **Create FileWriter Object:**

To create FileWriter class object,we have to use the following constructor.

```
public FileWriter(String target_File)
```

**EX:** `FileWriter fw = new FileWriter("abc.txt");`

It will override the existed content with the new content at each and every write operation.

```
public FileWriter(String target_File,boolean b)
```

**EX:** `FileWriter fw = new FileWriter("abc.txt",true);`

It will append new content to the existed content available in the file at each and every write operation.

When JVM encounter the above instructions, JVM will take the specified file and JVM search for the specified file at the respective location, if the required target file is available then JVM will establish FileWriter from Java application to the target file. If the required target file is not available at the respective location then JVM will create a new file with the same specified file name and establish FileWriter from Java application to the target file.

- **Declare the data which we want to transfer and convert that data into char[]:**

```
String data = "Hello";
char[] ch = data.toCharArray();
```

- **Write char[] data into FileWriter:**

To write char[] data into FileWriter,we have to use the following method.

```
public void write(char[] ch) throws IOException
```

**EX:** `fw.write(ch);`



- **Close FileWriter:**

```
fw.close();
```

**EX:**

```
1) import java.util.*;
2) public class FileWriterEx {
3)     public static void main(String args[]) throws Exception {
4)         FileWriter fw = new FileWriter("abc.txt",true);
5)         String data = "DurgaSoftwareSolutions";
6)         char[] ch = data.toCharArray();
7)         fw.write(ch);
8)         fw.close();
9)     }
10) }
```

## **FileReader:**

This character-oriented stream can be used to transfer the data from a particular source file to Java program.

If we want to transfer the data from a particular source file to Java program by using FileReader then we have to use the following steps:

- **Create FileReader Class Object:**

To create FileReader class object,we have to use the following constructor.

```
public FileReader(String file_Name) throws FileNotFoundException
```

**EX:** FileReader fr = new FileReader("abc.txt");

When JVM encounter the above instruction, JVM will perform the following steps.

- JVM will take source file name from FileReader constructor.
- JVM will check whether the specified file is available or not at the respective location.
- If the specified source file is not available at the respective location then JVM will rise an exception like "java.io.FileNotFoundException".
- If the specified file is existed at the respective location then JVM will establish FileReader from source file to Java program.
- After creating FileReader, JVM will transfer the content of source file to FileReader object in the form of characters.



- **Read Data from FileReader:**

To read data from FileReader, we have to use the following steps.

- Read character by character from FileReader in the form of ASCII values.
- Convert that ASCII values into the respective characters.
- Append the converted characters to a String variable.

Repeat the above steps up to all the characters which are available in the respective source file or up to the end-of-file character i.e "-1".

To read an ASCII value from FileReader, we have to use the following method.

public int read() throws IOException

- **Close FileReader:**

fr.close();

**EX:**

```
1) import java.util.*;
2) public class FReX {
3)     public static void main(String args[])throws Exception {
4)         FileWriter fr = new FileWriter("abc.txt");
5)         String data="";
6)         int val = fr.read();
7)         while(val != -1) {
8)             data = data+(char)val;
9)             val = fr.read();
10)        }
11)        System.out.println(data);
12)        fr.close();
13)    }
14} }
```

## Write A Java Program To Copy A Document From One File To Another File By Using Character Oriented Streams?

```
1) import java.io.*;
2) public class FileCopyEx {
3)     public static void main(String args[])throws Exception {
4)         FileReader fr = new FileReader("hibernatecfg.xml");
5)         String data="";
6)         int val = fr.read();
7)         while(val != -1) {
8)             data = data+(char)val;
9)             val = fr.read();
10)        }
```



```
11)     char[] ch = data.toCharArray();
12)     FileWriter fw = new FileWriter("abc.xml");
13)     fw.write(ch);
14)     fr.close();
15)     fw.close();
16)   }
17} }
```

## Approaches to provide dynamic Input:

There are three approaches to provide dynamic input in java applications.

- BufferedReader
- Scanner
- Console

## BufferedReader:

If we want to take dynamic input by using BufferedReader in java applications then we have to use the following statement.

BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
Where "in" is static variable,it will refer a predefined "InputStream" object which is connected with command prompt.

If we provide data on command prompt then that data will be transferred to InputStream object in the form of binary data.

where "InputStreamReader can be used to convert the data from binary representation to character representation.

where BufferedReader can be used to improve the performance of Java application while performing input operation.

To read the data from BufferedReader, we will use the following method

- readLine()
- read()



## Q) What is the difference between readLine() Method and read() Method?

readLine() method will read a line of text from command prompt [BufferedReader] and it will return that data in the form of String.

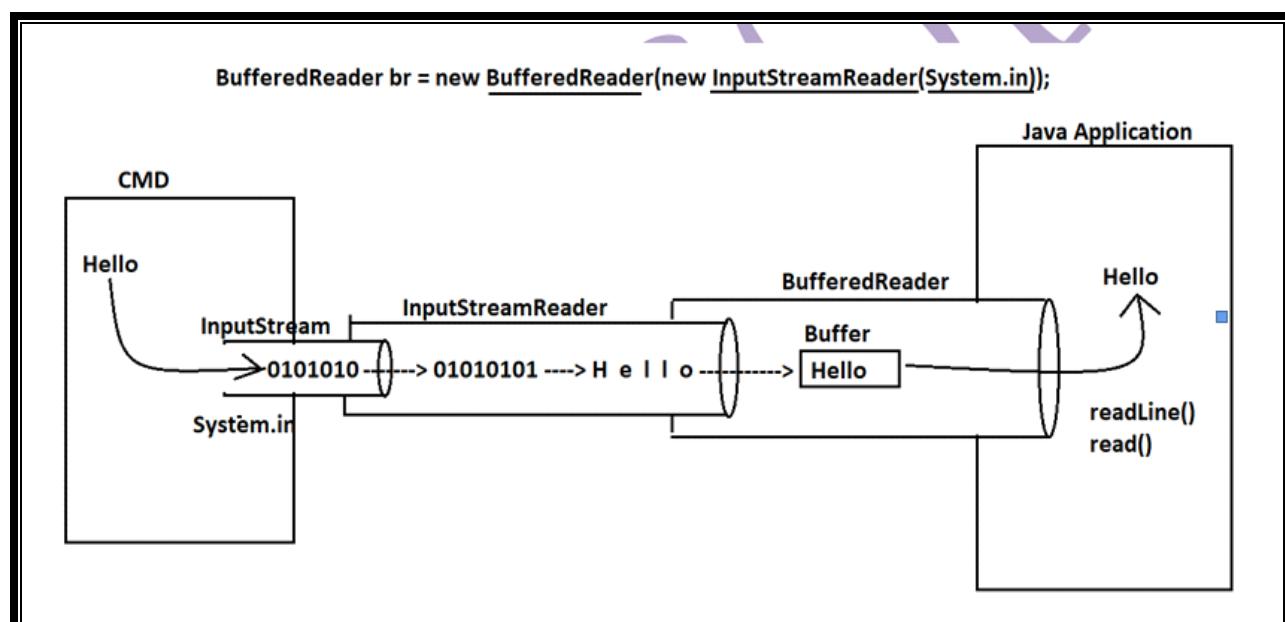
```
public String readLine() throws IOException
```

read() method will read a single character from command prompt [BufferedReader] and it will return that character in the form of its ASCII value.

```
public int read() throws IOException
```

EX:

```
1) import java.io.*;
2) public class BufferedReaderEx {
3)     public static void main(String args[]) throws Exception {
4)         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
5)         System.out.println("Enter Text :");
6)         String data1 = br.readLine();
7)         System.out.println("Enter the same text again :");
8)         int data2 = br.read();
9)         System.out.println("First Entered :" + data1);
10)        System.out.println("Second Entered :" + data2 + "---->" + (char) data2);
11)    }
12) }
```





Consider the following Program:

```
1) import java.io.*;
2) public class BufferedReaderEx {
3)     public static void main(String args[]) throws Exception {
4)         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
5)         System.out.println("First value :");
6)         String val1 = br.readLine();
7)         System.out.println("Second value :");
8)         String val2 = br.readLine();
9)         System.out.println("Addition :" +val1+val2);
10)    }
11) }
```

If we provide 10 and 20 as dynamic input to the above program then the above program will display "1020" value instead of 30 that is the above program has performed String concatenation instead of performing Arithmetic Addition because `br.readLine()` method has return 10 and 20 values in the form String data.

In the above program, if we want to perform Arithmetic operations over dynamic input then we have to convert String data into the respective primitive data, for this we have to use Wrapper Classes.

ThereFor, BufferedReader dynamic input approach is depending on wrapper classes while reading primitive data as dynamic input.

EX:

```
1) import java.io.*;
2) public class BufferedReaderEx {
3)     public static void main(String args[]) throws Exception {
4)         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
5)         System.out.println("First value :");
6)         String val1 = br.readLine();
7)         System.out.println("Second value :");
8)         String val2 = br.readLine();
9)         int f_Val = Integer.parseInt(val1);
10)        int s_Val = Integer.parseInt(val2);
11)        System.out.println("Addition :" +(f_Val+s_Val));
12)    }
13) }
```



## Scanner:

- This class is provided by Java in `java.util` package along with JDK5.0 Version.
- In java applications, if we use `BufferedReader` to dynamic input then we must use wrapper classes while reading primitive data as dynamic Input.
- In java applications, if we use "Scanner" to read dynamic input then it is not required to use wrapper classes while reading primitive data as dynamic input, scanner is able to provide environment to read primitive data directly from command prompt.
- If we want to use scanner in Java applications then we have to use the following steps.

- Create Scanner class Object:

To create Scanner class Object, we have to use the following constructor

`public Scanner(InputStream is)`

EX: `Scanner s=new Scanner(System.in);`

- Read dynamic Input:

To read String data as Dynamic input, we have to use the following method.

`public String next()`

To read primitive data as Dynamic input, we have to use the following method.

`public xxx nextXXX()`

where `xxx` may be `byte,short,int,float.....`

EX:

```
1) import java.util.*;
2) public class ScannerEx
3) {
4)     public static void main(String[] args) throws Exception
5)     {
6)         Scanner s=new Scanner(System.in);
7)         System.out.print("Employee Number :");
8)         int eno=s.nextInt();
9)         System.out.print("Employee Name :");
10)        String ename=s.next();
11)        System.out.print("Employee Salary :");
12)        float esal=s.nextFloat();
13)        System.out.print("Employee Address :");
14)        String eaddr=s.next();
15)
16)        System.out.println("Employee Details");
17)        System.out.println("-----");
18)        System.out.println("Employee Number :" +eno);
```



```
19) System.out.println("Employee Name :"+ename);
20) System.out.println("Employee Salary :" +esal);
21) System.out.println("Employee Address :" +eaddr);
22) }
23) }
```

**EX:**

```
1) import java.util.*;
2) public class ScannerEx1
3) {
4)     public static void main(String[] args) throws Exception
5)     {
6)         Scanner s=new Scanner(System.in);
7)         System.out.print("Enter Text Data :");
8)         String data1=s.nextLine();
9)         System.out.print("Enter The same text data again :");
10)        String data2=s.next();
11)        System.out.println("First Entered :" +data1);
12)        System.out.print("Second Entered :" +data2);
13)    }
14) }
```

## Console:

This class is provided by Java in `java.io` package along with JAVA 6 Version.

In Java applications, to take dynamic input, if we use `BufferedReader` and `Scanner` then we are able to get the following drawbacks:

- We have to consume 2 instructions for each and every dynamic input  
[`s.o.println(..)` and `readLine()` or `nextXXX()` methods]
- These are not providing security for the data like password data, pin numbers.....
- To overcome the above problems, we have to use "Console" dynamic input approach.
- If we want to use Console in Java applications then we have to use the following Steps:

## Create Console Object:

To get Console object, we have to use the following method from "System" class.

`public static Console console()`

**EX:** `Console c = System.console();`



## Read Dynamic Input:

To read String data, we have to use the following method.

```
public String readLine(String msg)
```

To read password data, we have to use the following method.

```
public char[] readPassword(String msg)
```

EX:

```
1) import java.io.*;
2) public class ConsoleEx {
3)     public static void main(String args[]) throws Exception {
4)         Console c = System.console();
5)         String uname = c.readLine("User Name :");
6)         char[] pwd = c.readPassword("PassWord :");
7)         String upwd = new String(pwd);
8)         if(uname.equals("durga") && upwd.equals("durga")) {
9)             System.out.println("Valid User");
10)        }
11)        else {
12)            System.out.println("InValid User");
13)        }
14)    }
15) }
```

## Serialization and Deserialization:

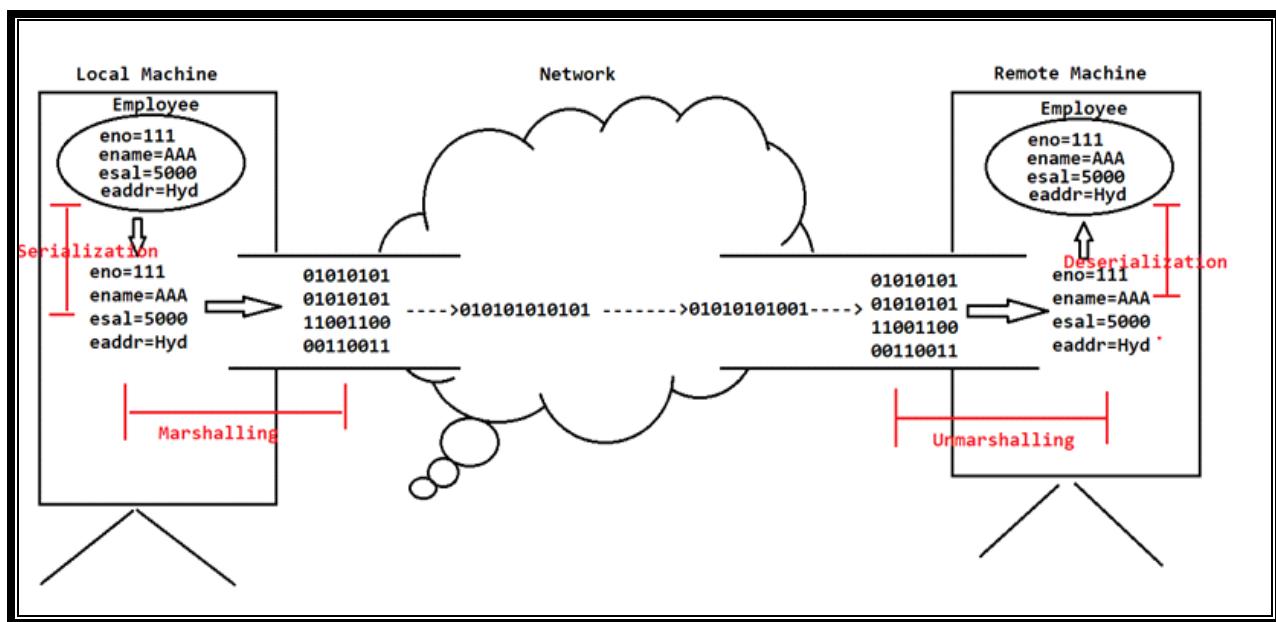
If we design Java applications by distributing application logic over multiple [JVMS] then that Java application is called as Distributed Application.

In general, in Distributed applications, it is frequent requirement to transfer an object [Distributed Object] from one machine to another machine.

In Java, Object is a block of memory, it is not possible to transfer the object through network, where we have to transfer object data from one machine to another machine through network.

To transfer an Object through network from one machine to another machine, first we have to separate the data from an object at local machine and convert the data from system representation to network representation then transfer the data to network.

At remote machine, we have to get the data from network and convert the data from system representation to System representation and reconstruct an object on the basis of data.



The process of converting the data from System representation to network representation is called as "Marshalling".

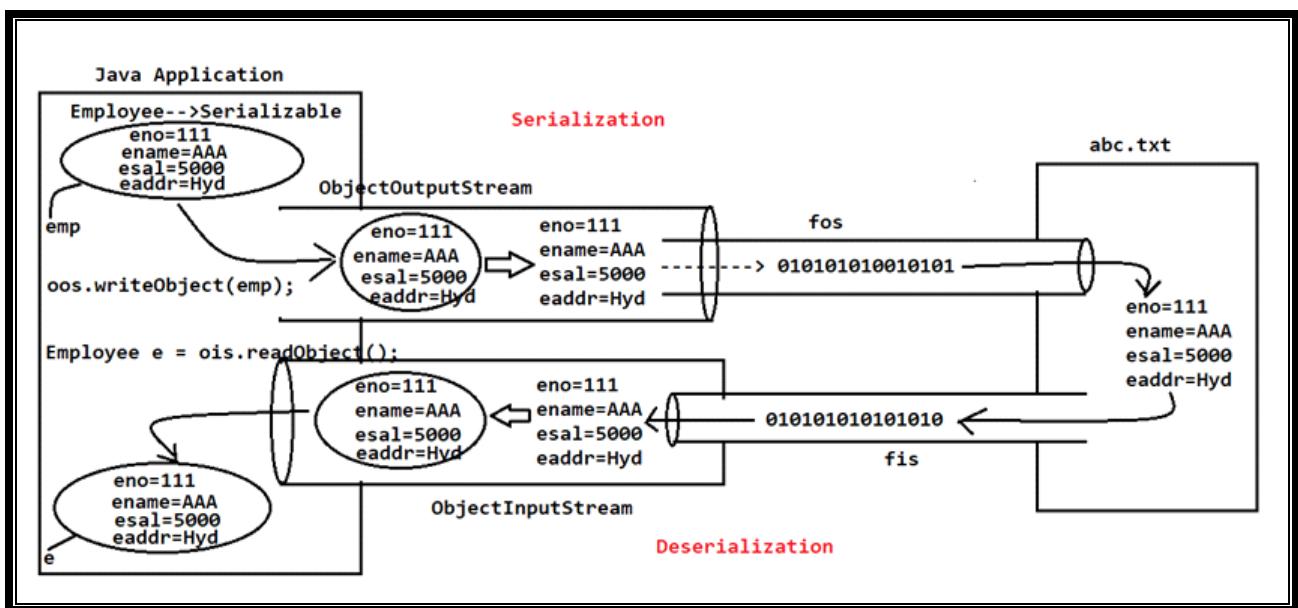
The process of converting the data from Network representation to System representation is called as "UnMarshalling".

The Process of separating the data from an Object is called as "Serialization".

The process of reconstructing an object on the basis of data is called as "Deserialization".

To perform Serialization and Deserialization in Java applications, JAVA has given two predefined byte-oriented streams like `java.io.ObjectOutputStream` for Serialization and `java.io.ObjectInputStream` for Deserialization

In Standalone applications, if we want to perform Serialization and Deserialization over an object then we have to take a file[text file] to store serialized data.



## Steps to Perform Serialization:

- **Create Serializable Object:**
- To create Serializable Object we have to implement `java.io.Serializable` marker interface to the respective class.
- Serializable interface is marker interface, it will make eligible any object for Serialization and Deserialization.

**EX:**

```
class Employee implements Serializable {  
    int eno = 111;  
    String ename = "AAA";  
    float esal = 5000;  
}  
Employee e1 = new Employee();
```

- **Prepare FileOutPutStream with a particular target File:**

```
FileOutPutStream fos=new FileOutPutStream("abc.txt");
```

- **Create ObjectOutPutStream:**

To create `ObjectoutputStream`,we have to use the following constructor.

```
public ObjectOutputStream(FileOutputStream fos)
```

**EX:** `ObjectOutputStream oos=new ObjectOutputStream(fos);`



- **Write Serializable object to ObjectOutputStream:**

To write Serializable object to ObjectOutputStream, we have to use the following method.

public void writeObject(Object obj) throws NotSerializableException

EX: oos.writeObject(e1);

## **Steps To perform DeSerialization:**

- **Create FileInputStream object:**

FileInputStream fis = new FileInputStream("emp.txt");

- **Create ObjectInputStream:**

To create ObjectInputStream class object, we have to use the Following constructor.

public ObjectInputStream(FileInputStream fis)

EX: ObjectInputStream ois=new ObjectInputStream(fis);

- **Read DeSerialized Data from ObjectInputStream:**

To read DeSerialized object from ObjectInputStream, we have to use the following method.

public Object readObject()

EX: Employee e2 = (Employee)ois.readObject();

EX:

```
1) import java.io.*;
2) class Employee implements Serializable
3) {
4)     int eno;
5)     String ename;
6)     float esal;
7)     String eaddr;
8)
9)     Employee(int eno, String ename, float esal, String eaddr)
10)    {
11)        this.eno=eno;
12)        this.ename=ename;
13)        this.esal=esal;
14)        this.eaddr=eaddr;
15)    }
16)
17)    public void getEmpDetails()
18)    {
19)        System.out.println("Employee Details");
```



```
20) System.out.println("-----");
21) System.out.println("Employee Number :" +eno);
22) System.out.println("Employee Name  :" +ename);
23) System.out.println("Employee Salary :" +esal);
24) System.out.println("Employee Address :" +eaddr);
25) }
26) }
27) class SerializationEx
28) {
29)     public static void main(String[] args) throws Exception
30)     {
31)         FileOutputStream fos=new FileOutputStream("emp.txt");
32)         ObjectOutputStream oos=new ObjectOutputStream(fos);
33)         Employee emp1=new Employee(111, "Durga", 50000, "Hyd");
34)         System.out.println("Employee Details before Serialization");
35)         emp1.getEmpDetails();
36)         oos.writeObject(emp1);
37)         System.out.println();
38)         FileInputStream fis=new FileInputStream("emp.txt");
39)         ObjectInputStream ois=new ObjectInputStream(fis);
40)         Employee emp2=(Employee)ois.readObject();
41)         System.out.println("Employee Details After Deserialization");
42)         emp2.getEmpDetails();
43)     }
44) }
```

- ☺ In Object serialization, static members are not allowed.
- ☺ If we serialize any object having static variables then compiler will not rise any error and JVM will not rise any exception but static variables will not be listed in the serialized data in the text file.
- ☺ In object serialization, if we do not want to allow any variable in serialization and deserialization then we have to declare that variable as "transient" variable.
- ☺ transient int eno=111;

EX:

```
1) import java.io.*;
2) class User implements Serializable {
3)     String uname;
4)     transient String upwd;
5)     String uemail;
6)     long umobile;
7)     public static final int MIN_AGE=18;
8)     public static final int MAX_AGE=25;
9)
10)    User(String uname, String upwd, String uemail, long umobile)
```



```
11) {
12)     this.uname=uname;
13)     this.upwd=upwd;
14)     this.uemail=uemail;
15)     this.umobile=umobile;
16) }
17)
18) class Test
19) {
20)     public static void main(String[] args)throws Exception
21)     {
22)         FileOutputStream fos=new FileOutputStream("abc.txt");
23)         ObjectOutputStream oos=new ObjectOutputStream(fos);
24)         User u=new User("abc", "abc123", "abc@duurgasoft.com", 998877);
25)         oos.writeObject(u);
26)     }
27} }
```

In Java applications, if we serialize an object which is not implementing `java.io.Serializable` interface then JVM will rise an exception like "`java.io.NotSerializableException`".

```
1) import java.io.*;
2) class A {
3)     int i = 10;
4)     int j = 20;
5) }
6) class Test {
7)     public static void main(String args[])throws Exception {
8)         FileOutputStream fos = new FileOutputStream("abc.txt");
9)         ObjectOutputStream oos = new ObjectOutputStream(fos);
10)        A a = new A();
11)        oos.writeObject(a);
12)    }
13} }
```

Status:"`java.io.NotSerializableException`".[Exception]

In Java applications, if we implement `Serializable` interface to the super class then automatically all the sub class objects are eligible for Serialization and Deserialization.



EX:

```
1) import java.io.*;
2) class A implements Serializable {
3)     int i = 10;
4)     int j = 20;
5) }
6) class B extends A {
7)     int k = 30;
8)     int l = 40;
9) }
10) class Test {
11)     public static void main(String args[]) throws Exception {
12)         FileOutputStream fos = new FileOutputStream("abc.txt");
13)         ObjectOutputStream oos = new ObjectOutputStream(fos);
14)         B b = new B();
15)         oos.writeObject(b);
16)     }
17} }
```

In Java applications, if we implement Serializable interface in sub class then only sub class properties are allowed in Serialization and deserialization, the respective super class members are not allowed in the Serialization and deserialization.

EX:

```
1) import java.io.*;
2) class A {
3)     int i = 10;
4)     int j = 20;
5) }
6) class B extends A implements Serializable {
7)     int k = 30;
8)     int l = 40;
9) }
10) class Test {
11)     public static void main(String args[]) throws Exception {
12)         FileOutputStream fos = new FileOutputStream("abc.txt");
13)         ObjectOutputStream oos = new ObjectOutputStream(fos);
14)         B b = new B();
15)         oos.writeObject(b);
16)     }
17} }
```

In Java applications, while Serializing an object if any associated object is available then JVM will serialize the respective associated object also but the respective associated



object must implement Serializable interface otherwise JVM will rise an exception like "java.io.NotSerializableException".

EX:

```
1) import java.io.*;
2) class Branch implements Serializable {
3)     String bid;
4)     String bname;
5)     Branch(String bid, String bname) {
6)         this.bid = bid;
7)         this.bname = bname;
8)     }
9) }
10) class Account implements Serializable {
11)     String accNo;
12)     String accName;
13)     Branch branch;
14)     Account(String accNo, String accName, Branch branch) {
15)         this.accNo = accNo;
16)         this.accName = accName;
17)         this.branch = branch;
18)     }
19) }
20) class Employee implements Serializable {
21)     String eid;
22)     String ename;
23)     Account acc;
24)     Employee(String eid, String ename, Account acc) {
25)         this.eid = eid;
26)         this.ename = ename;
27)         this.acc = acc;
28)     }
29) }
30) class Test {
31)     public static void main(String args[]) throws Exception {
32)         FileOutputStream fos = new FileOutputStream("abc.txt");
33)         ObjectOutputStream oos = new ObjectOutputStream(fos);
34)         Branch branch = new Branch("B-111","S R Nagar");
35)         Account acc = new Account("abc123","Durga",branch);
36)         Employee emp = new Employee("E-111","Durga",acc);
37)         oos.writeObject(emp);
38)     }
39) }
```



## Externalization:

As part of object serialization and deserialization we are able to separate the data from Object and stored in a text file and we are able to retrieve that object data from text file to Object in Java application.

In Java applications, to perform serialization and deserialization Java has given "ObjectOutputStream" and "ObjectInputStream" two byte-oriented Streams.

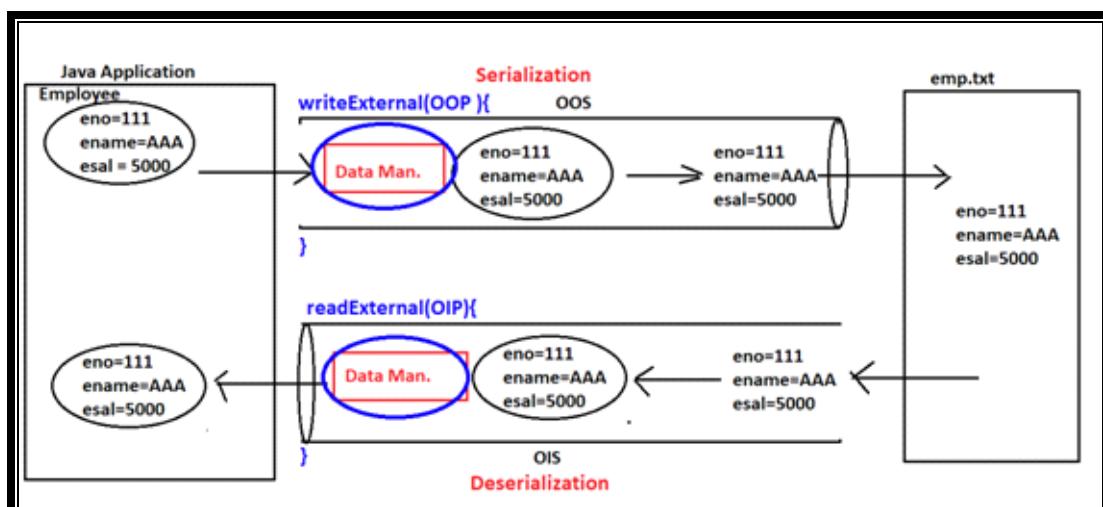
In Java applications, to perform Serialization just we have to send Serializable object to ObjectOutputStream, it will perform serialization internally, where Developers are not having controlling over serialization process.

In Java applications, to perform deserialization just we have to create ObjectInputStream and we have to read deserialized object, where ObjectInputStream will perform deserialization internally, where developers are not having controlling over deserialization process.

In the above context to have controlling over serialization and deserialization processes inorder to provide the services like security, data compression, data decompression, data encoding....over serialized and deserialized data we have to go for "Externalization".

If we want to perform Externalization in java applications, we have to use the following steps.

- 1) Prepare Externalizable object
- 2) Perform Serialization and Deserialization over Externalizable object.





## Prepare Externalizable Object:

In Java applications, if we want to create Serializable object then the respective class must implement `java.io.Serializable` interface.

Similarly, if we want to prepare Externalizable object then the respective class must implement `java.io.Externalizable` interface.

`java.io.Externalizable` is a sub interface to `java.io.Serializable` interface.

`java.io.Serializable` interface is a marker interface, which is not having abstract methods but `java.io.Externalizable` interface is not marker interface, which includes the following methods.

```
public void writeExternal(ObjectOutput oop) throws IOException  
public void readExternal(ObjectInput oip) throws IOException, ClassNotFoundException
```

where `writeExternal()` method will be executed just before performing serialization in `ObjectOutputStream`,

where we have to perform manipulations on the data which we want to serialize.

where `readExternal()` method will be executed immediately after performing Deserialization in `ObjectInputStream`, where we can perform manipulations over the deserialized data.

where `ObjectOutput` is stream, it will carry manipulated data for Serialization.

To put data in `ObjectOutput`, we have to use the following methods.

```
public void writeXXX(xxx data)
```

where `xxx` may be `byte, short, int, UTF[String]`.....

where `ObjectInput` will get serialized data from text file to perform manipulations.

To read data from `ObjectInput` we have to use the following method

```
public void readXXX(xxx data)
```

where `xxx` may be `byte, short, int, UTF[String]`.....

If we want to prepare Externalizable object we have to use the following steps.

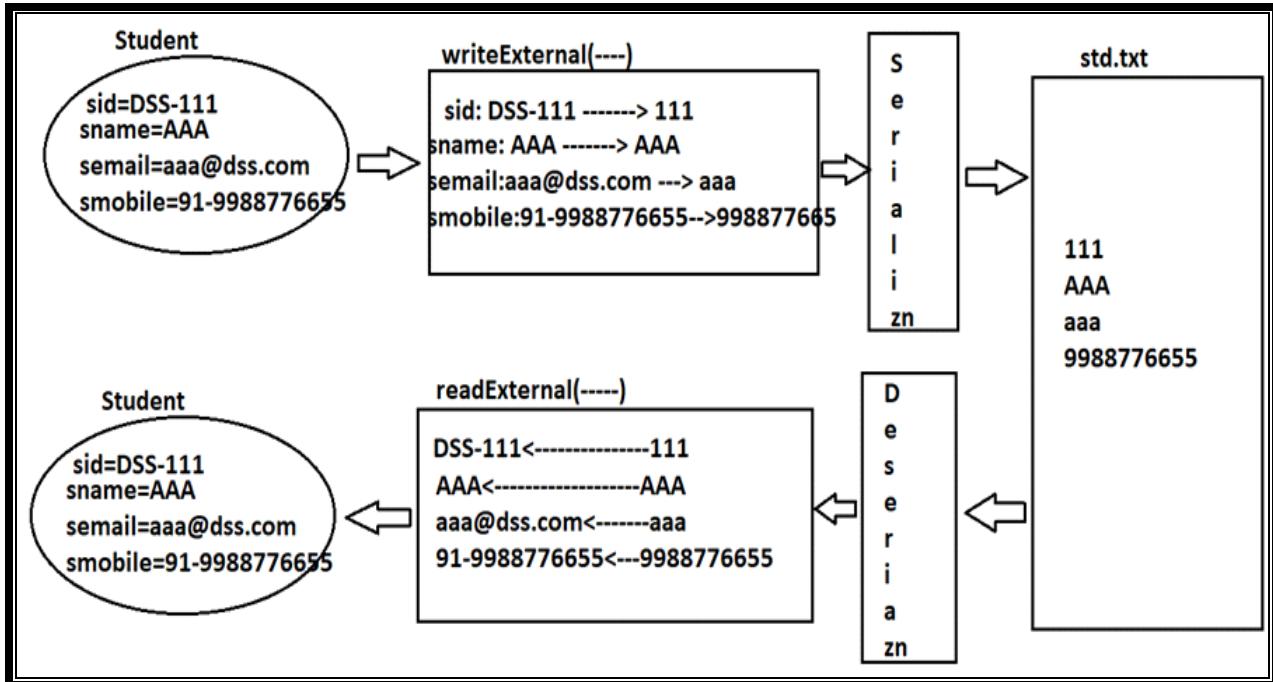
- Declare an user defined class
- Implement `java.io.Externalizable` interface.
- Implement the methods `writeExternal()` and `readExternal()` of Externalizable interface at the user defined class.

```
class Employee implements Externalizable {  
    public void writeExternal(ObjectOutput oop) throws IOException {  
        ---implementation---  
    }  
    public void readExternal(ObjectInput oip) throws IOException,  
        ClassNotFoundException {  
        ---implementation----  
    }
```



```
}
```

```
Employee emp = new Employee();
```



## Perform Serialization and Deserialization over Externalizable object by using ObjectOutputStream and ObjectInputStream:

Same as Serialization and DeSerialization

EX:

```
1) import java.util.*;  
2) import java.io.*;  
3) class Employee implements Externalizable {  
4)     String eid;  
5)     String ename;  
6)     String email;  
7)     String emobile;  
8)     //It will be used to construct object while performing deserialization in  
9)     //Externalization process  
10)    public Employee() {  
11)    }  
12)    Employee(String eid, String ename, String email, String emobile) {  
13)        this.eid = eid;  
14)        this.ename = ename;  
15)        this.email = email;  
16)        this.emobile = emobile;  
17)    }
```



```
18) public void writeExternal(ObjectOutput oop) throws IOException {
19)     try {
20)         StringTokenizer st1=new StringTokenizer(eid,"-");
21)         st1.nextToken();
22)         int no=Integer.parseInt(st1.nextToken());
23)         StringTokenizer st2=new StringTokenizer(email,"@");
24)         String mail=st2.nextToken();
25)         StringTokenizer st3=new StringTokenizer(emobile,"-");
26)         st3.nextToken();
27)         String mobile=st3.nextToken();
28)         oop.writeInt(no);
29)         oop.writeUTF(ename);
30)         oop.writeUTF(mail);
31)         oop.writeUTF(mobile);
32)     }
33)     catch(Exception e) {
34)         e.printStackTrace();
35)     }
36) }
37) public void readExternal(ObjectInput oip) throws IOException,
ClassNotFoundException {
38)     eid = "E-"+oip.readInt();
39)     ename = oip.readUTF();
40)     email = oip.readUTF()+"@durgasoft.com";
41)     emobile = "91-"+oip.readUTF();
42) }
43) public void getEmpDetails() {
44)     System.out.println("Employee Details");
45)     System.out.println("-----");
46)     System.out.println("Employee Id : "+eid);
47)     System.out.println("Employee Name : "+ename);
48)     System.out.println("Employee Mail : "+email);
49)     System.out.println("Employee Mobile: "+emobile);
50) }
51) }
52) class ExternalizableEx {
53)     public static void main(String args[]) throws Exception {
54)         FileOutputStream fos = new FileOutputStream("emp.txt");
55)         ObjectOutputStream oos = new ObjectOutputStream(fos);
56)         Employee emp1 = new Employee("E- 111", "Durga",
"durga@durgasoft.com", "91-9988776655");
57)         System.out.println("Employee Data before Serialization");
58)         emp1.getEmpDetails();
59)         oos.writeObject(emp1);
60)         System.out.println();
```



```
61)     FileInputStream fis = new FileInputStream("emp.txt");
62)     ObjectInputStream ois = new ObjectInputStream(fis);
63)     Employee emp2 = (Employee)ois.readObject();
64)     System.out.println("Employee Data After Deserialization");
65)     emp2.getEmpDetails();
66)
67} }
```

## Files in Java:

File is a storage area to store data.

There are two types of files in Java.

- Sequential Files
- RandomAccessFiles

### **Sequential Files:**

It will allow the user to retrieve data in Sequential manner.

To represent Sequential files, Java has given a predefined class in the form of java.io.File.

To create File class object we have to use the following constructor.

**public File(String file\_Name) throws FileNotFoundException**

**EX: File f = new File("c:/abc/xyz/emp.txt");**

Creating File class object is not sufficient to create a file at directory structure we have to use the following method.

**public File createNewFile()**

To create a Directory, we have to use the following method.

**public File mkdir()**

To get file / directory name we have to use the following method.

**public String getName()**

To get file / directory parent location, we have to use the following method.

**public String getParent()**

To get file / directory absolute path, we have to use the following method.

**public String getAbsolutePath()**



To check whether the created thing File or not, we have to use the following method.  
`public boolean isFile()`

To check whether the created thing is directory or not we have to use the following method.  
`public boolean isDirectory()`

EX:

```
1) import java.io.*;
2) class Test {
3)     public static void main(String args[]) throws Exception {
4)         File f = new File("c:/abc/xyz/emp.txt");
5)         f.createNewFile();
6)         System.out.println(f.isFile());
7)         System.out.println(f.isDirectory());
8)         File f1 = new File("c:/abc/xyz/student");
9)         f1.mkdir();
10)        System.out.println(f.isFile());
11)        System.out.println(f.isDirectory());
12)        System.out.println("File Name :" +f.getName());
13)        System.out.println("Parent Name :" +f.getParent());
14)        System.out.println("Absolute Path :" +f.getAbsolutePath());
15)        int size = fis.available();
16)        byte[] b = new byte[size];
17)        fis.read();
18)        String data = new String(b);
19)        System.out.println(data);
20)    }
21} }
```

## RandomAccessFile:

It is a Storage area, it will allow the user to read data from random positions. To represent this file, java has given a predefined class in the form of "java.io.RandomAccessFile".

To create RandomAccessFile class object, we have to use the following constructor.

`public RandomAccessFile(String file_name, String access_Privileges)`  
where access\_Privileges may be "r" [Read] or "rw" [Read and Write]

To write data into randomAccessFile, we have to use the following method.

`public void writeXXX(xxx value)`  
where xxx may be byte, short, int, UTF[String],.....



To read data from RandomAccessFile, we have to use the following method

public XXX readXXX()

where xxx may be byte, short, int, UTF[String],.....

To move file pointer to a particular position in RandomAccessFile, we have to use the following method.

public void seek(int position)

**EX:**

```
1) import java.io.*;
2) class Test {
3)     public static void main(String args[]) throws Exception {
4)         RandomAccessFile raf = new RandomAccessFile("abc.txt","rw");
5)         raf.writeInt(111);
6)         raf.writeUTF("Durga");
7)         raf.writeFloat(5000.0f);
8)         raf.writeUTF("HYD");
9)         raf.seek(0);
10)        System.out.println("Employee Number :" +raf.readInt());
11)        System.out.println("Employee Name :" +raf.readUTF());
12)        System.out.println("Employee Salary :" +raf.readFloat());
13)        System.out.println("Employee Address :" +raf.readUTF ());
14)    }
15) }
```



# Networking



By using java programming language we are able to prepare the following two types of applications.

- Standalone Application
- Distributed Application

## **Standalone Application:**

If we prepare any java application without using client-server arch then that java application is called as "Standalone Application".

To prepare standalone applications we can use Core Libraries like `java.io`, `java.util`, `java.lang`,....

## **Distributed Application:**

- If we prepare any java application on the basis of Client-Server arch then that java application is called as Distributed application.
- To prepare Distributed applications, JAVA has provided the following set of technologies.
  - ☺ Socket Programming
  - ☺ RMI
  - ☺ CORBA
  - ☺ EJBs
  - ☺ WebServices

## **Socket Programming:**

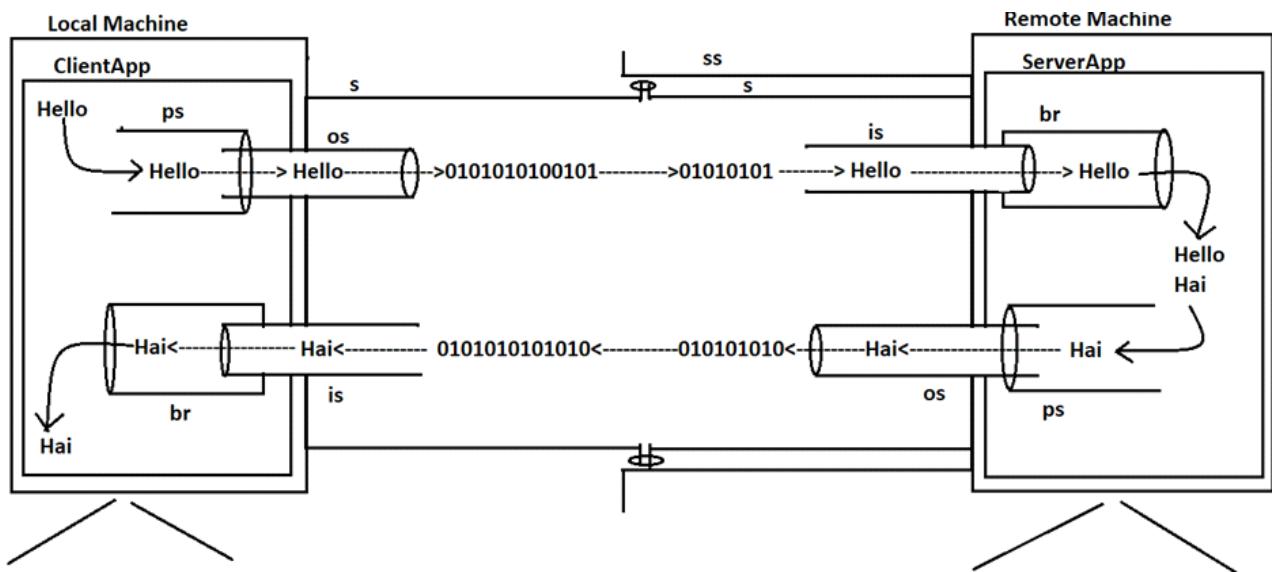
- If we want to prepare distributed applications by using Socket Programming then we have to use Sockets between machines to transfer data from one machine to another machine.
- Socket is a Channel or medium to transfer data from one machine to another machine.
- In Socket programming we have to establish Sockets on the basis of System IP Address and Port Numbers.

### **Q) What is the difference between IPAddress and Port Number?**

- ☺ IP Address is an unique identity to each and every machine over the network and which is provided by network manager at the time of network configuration.
- ☺ Port Number is an unique identity to each and every process being executed with in a single machine and it would be provided by local operating system.
- ☺ To prepare distributed applications by using Socket Programming the required predefined library was provided by JAVA in the form of "`java.net`" package.



## Socket programming Arch:



## Steps to prepare Client Application:

### 1) Create Socket at Client Machine:

- To create Socket class object we have to use the following constructor from java.net.Socket class.-
- `public Socket(String server_IP_Addr, int server_Port_No)`
- EX: `Socket s = new Socket("localhost", 4444);`
- NOTE: If server socket is available at the same machine then we are able to use "localhost" inplace of Server\_IP\_Addr.

### 2) Get OutputStream from Socket:

- To get OutputStream from Socket we have to use the following method from java.net.Socket class.
- `public OutputStream getOutputStream()`
- EX: `OutputStream os = s.getOutputStream();`

### 3) Create PrintStream with OutputStream:

```
PrintStream ps = new PrintStream(os);
```

### 4) Send Data to PrintStream:

```
String data = "Hello";  
ps.println(data);
```



**NOTE:** With the above steps, data will be send to Server, where Server will send response data to client.

## 5) Get InputStream from Socket:

To get InputStream from Socket we have to use the following method.

```
public InputStream getInputStream()
```

```
EX: InputStream is = s.getInputStream();
```

## 6) Create BufferedReader with InputStream:

```
BufferedReader br = new BufferedReader(new InputStreamReader(is));
```

## 7) Read data from BufferedReader:

```
String data = br.readLine();
```

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

## Steps To prepare Server Application:

- Get InputStream from Socket:

```
InputStream is = s.getInputStream();
```

- Create BufferedReader with InputStream:

```
BufferedReader br = new BufferedReader(new InputStreamReader(is));
```

- Read Data from BufferedReader:

```
String data = br.readLine();
```

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

- Get OutputStream from Socket:

```
OutputStream os = s.getOutputStream();
```

- Create PrintStream with OutputStream:

```
PrintStream ps = new PrintStream(os);
```

- Send Data to PrintStream:

```
String data = "Hai";
```

```
ps.println(data);
```

**NOTE:** At Server machine, we have to create ServerSocket and it has to accept the request from client about to assign Socket from server machine in order to establish connection .



---

**EX:** `ServerSocket ss = new ServerSocket(4444);  
Socket s = ss.accept();`

The above application provides one time communication, but, if we want to provide infinite communication then we have to use infinite loops at both client application and Server application.

## **Application-2:**

### **ClientApp.java**

```
1) import java.io.*;  
2) import java.net.*;  
3) public class ClientApp  
4) {  
5)     public static void main(String[] args) throws Exception  
6)     {  
7)         Socket s=new Socket("localhost", 4444);  
8)         OutputStream os=s.getOutputStream();  
9)         PrintStream ps=new PrintStream(os);  
10)        BufferedReader br1=new BufferedReader(new InputStreamReader(System.in));  
11)        InputStream is=s.getInputStream();  
12)        BufferedReader br2=new BufferedReader(new InputStreamReader(is));  
13)        while(true)  
14)        {  
15)            String data1=br1.readLine();  
16)            ps.println(data1);  
17)            String data2=br2.readLine();  
18)            System.out.println(data2);  
19)            if(data1.equals("bye") && data2.equals("bye"))  
20)            {  
21)                System.exit(0);  
22)            }  
23)        }  
24)    }  
25} }
```



## ServerApp.java

```
1) import java.io.*;
2) import java.net.*;
3) public class ServerApp
4) {
5)     public static void main(String[] args) throws Exception
6)     {
7)         ServerSocket ss=new ServerSocket(4444);
8)         Socket s=ss.accept();
9)         InputStream is=s.getInputStream();
10)        BufferedReader br1=new BufferedReader(new InputStreamReader(is));
11)
12)        OutputStream os=s.getOutputStream();
13)        PrintStream ps=new PrintStream(os);
14)        BufferedReader br2=new BufferedReader(new InputStreamReader(System.in));
15)        while(true)
16)        {
17)            String data1=br1.readLine();
18)            System.out.println(data1);
19)            String data2=br2.readLine();
20)            ps.println(data2);
21)
22)            if(data1.equals("bye") && data2.equals("bye"))
23)            {
24)                System.exit(0);
25)            }
26)        }
27)    }
28} }
```



# RMI

[Remote Method Invocation]



To prepare Distributed applications, if we use Socket programming then we have to prepare ServerSocket, Client Side Socket, InputStreams, OutputStreams,..... at local machine and Remote machine explicitly, it will increase burden to the developers.

To overcome the above problems, SUN Microsystems has provided an alternative distributed tech to prepare distributed applications, that is, RMI.

In case of RMI, the complete distributed applications infrastructure like Server socket, socket, input streams and OutputStreams,.... are provided by RMI internally, developers are not required to provide distributed applications infrastructure explicitly.

RMI will use the following two components internally to provide Distributed applications infrastructure.

- Stub
- Skeleton

- **Stub:**

It is a special component in RMI, it will be existed at Local Machine, it will provide Client side socket, Input and Output Streams and it is capable to perform Serialization and Deserialization, Marshalling and Unmarshalling and it able to receive remote method calls from local machine and it will send remote method call to network, it able to receive return values from remote method call from network and it will send that return values to client application.

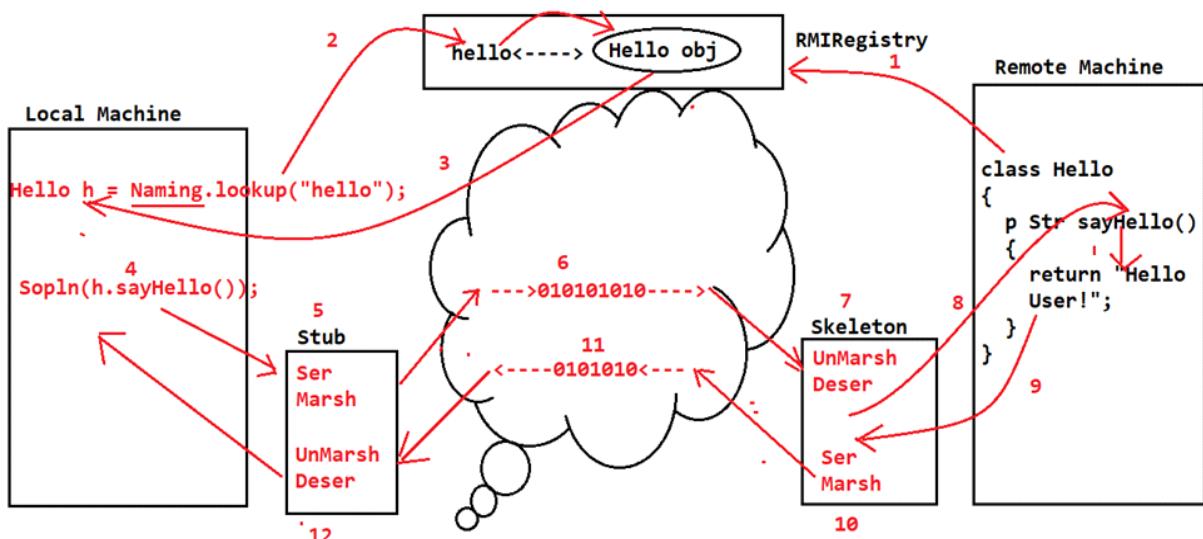
- **Skeleton:**

It is a special component in RMI, it will be existed at Remote machine, it will provide server socket, InputStreams and OutputStream, it is capable to perform Serialization and Deserialization, Marshalling and Unmarshalling and it able to receive remote method call from Network, it able to access remote methods and it able to carry return values of remote methods to network.

**NOTE:** To expose Remote objects in network, RMI will use a separate Registry Software called as "RMIRegistry".



## RMI Arch:



- Create Remote object and keep Remote object in RMIREGISTRY "along with logical name" in order to expose Remote object in network.
- At Local machine, perform lookup operation in RMIREGISTRY on the basis of logical name.
- RMIREGISTRY will return Remote object reference value to local machine or Client Application.
- Access Remote method by using Remote object reference.
- When we access Remote method, Remote method call will be send to Stub.
- Stub will perform the required "Serialization" and "Marshalling" over the remote method call.
- Stub will send remote method call to Network.
- Network will transfer remote method call to Remote machine.
- Skeleton will take remote method call from network.
- Skeleton will perform the required "Unmarshalling" and "Deserialization" over the remote method call.
- Skeleton will access remote method.
- Skeleton will get return value of remote method.
- Skeleton will perform the required "Serialization" and "Marshalling" over the return value.
- Skeleton will send return value to Network.
- Network will carry return value to Local machine.
- Stub will take return value from network.
- Stub will perform the required "Unmarshalling" and "Deserialization" over the return value.
- Stub will send return value to Client application.



## Steps to prepare RMI Application:

- Create Remote interface
- Create Remote interface implementation class
- Create Registry program
- Create Client Application
- Execute RMI Application

### **Create Remote Interface:**

The main intention of Remote interface is to declare all remote services in the form of abstract methods.

#### **STEPS:**

- Declare an user defined interface.
- Extend java.rmi.Remote interface to user defined interface.
- Declare remote methods in the form of abstract methods.
- Throws out java.rmi.RemoteException at each and every remote method.

#### **EX:**

```
1) public interface MyRemote extends java.rmi.Remote  
2) {  
3)     public String myRemoteService1() throws java.rmi.RemoteException;  
4)     public String myRemoteService2() throws java.rmi.RemoteException;  
5)     ----  
6) }
```

### **Create Remote Interface implementation Class:**

The main intention of Implementation class for Remote interface is to provide implementation to service methods.

#### **STEPS:**

- Declare an user defined class.
- Extend java.rmi.server.UnicastRemoteObject abstract class to user defined class in order to make network enable.
- Implement User defined Remote interface in user defined class.
- Declare public and 0-arg constructor in user defined class with throws java.rmi.RemoteException.
- Provide implementation for Service methods.



EX:

```
1) public class MyRemoteImpl extends java.rmi.server.UnicastRemoteObject  
   implements MyRemote  
2) {  
3)     public MyRemoteImpl() throws java.rmi.RemoteException  
4)     {  
5)     }  
6)     public String myRemoteService1() throws java.rmi.RemoteException  
7)     {  
8)         ----  
9)     }  
10)    public String myRemoteService2() throws java.rmi.RemoteException  
11)    {  
12)        ----  
13)    }  
14)    ----  
15) }
```

## 16) Create Registry program:

The main intention of Registry program is to create Remote object and to keep Remote object in RMIRegistry with a particular logical name.

STEPS:

- Declare an user defined class with main() method.
- In main() method, create Objectt for Remote interface implementation class.
- Bind Remote object with a logical name in RMIRegistry by using the following method from java.rmi.Naming class.  
`public static void bind(String logical_Name, Remte r) throws RemoteException`

EX:

```
1) public class MyRegistry  
2) {  
3)     public static void main(String[] args) throws Exception  
4)     {  
5)         MyRemote mr=new MyRemoteImpl();  
6)         java.rmi.Naming.bind("my_rem", mr);  
7)     }  
8) }
```



## **Create Client Application:**

The main intention of Client Application is to get Remote object from RMIRegistry and to access remote methods from Local machine.

### **STEPS:**

- Declare an user defined class with main() method.
- In main() method, get Remote object from RMIRegistry by using the following method from java.rmi.Naming class.  

```
public static Remote lookup(String logical_Name)
```
- Access Remote methods.

### **EX:**

```
1) class ClientApp
2) {
3)     public static void main(String[] args)
4)     {
5)         MyRemote mr = (MyRemote)java.rmi.Naming.lookup("my_rem");
6)         mr.myRemoteService1();
7)         mr.myRemoteService2();
8)         ----
9)         ----
10)    }
11} }
```

## **Execute RMI Application:**

- Compile all resources[.java files]  
D:\java830>javac \*.java
- Start RMIRegistry  
D:\java830>start RMIRegistry
- c)Execute Registry program  
D:\java830>start java MyRegistry
- d)Execute Client Application  
D:\java830>java ClientApp

### **Application1:**

D:\java830\rmi\app1

HelloRemote.java  
HelloRemoteImpl.java  
HelloRegistry.java  
ClientApp.java



## HelloRemote.java

```
1) import java.rmi.*;  
2) public interface HelloRemote extends Remote  
3) {  
4)     public String sayHello(String name)throws RemoteException;  
5) }
```

## HelloRemoteImpl.java

```
1) import java.rmi.*;  
2) import java.rmi.server.*;  
3) public class HelloRemoteImpl extends UnicastRemoteObject implements HelloRemote  
4) {  
5)     public HelloRemoteImpl()throws RemoteException  
6)     {  
7)     }  
8)     public String sayHello(String name)throws RemoteException  
9)     {  
10)         return "Hello..."+name+"!";  
11)     }  
12) }
```

## HelloRegistry.java

```
1) import java.rmi.*;  
2) public class HelloRegistry  
3) {  
4)     public static void main(String[] args)throws Exception  
5)     {  
6)         HelloRemote hr=new HelloRemoteImpl();  
7)         Naming.bind("hello", hr);  
8)         System.out.println("HelloRemote Object is binded with the logical 'hello' in RMIRegistry");  
9)     }  
10) }
```

## ClientApp.java

```
1) import java.rmi.*;  
2) public class ClientApp {  
3)     public static void main(String[] args)throws Exception  
4)     {  
5)         HelloRemote hr=(HelloRemote)Naming.lookup("hello");  
6)         String msg=hr.sayHello("Durga");  
7)         System.out.println(msg);  
8)     }  
9) }
```



## Application2:

D:\java830\rmi\app2

CalculatorRemote.java  
CalculatorRemotelImpl.java  
CalculatorRegistry.java  
ClientApp.java

### CalculatorRemote.java

```
1) import java.rmi.*;  
2) public interface CalculatorRemote extends Remote  
3) {  
4)     public int add(int i, int j) throws RemoteException;  
5)     public int sub(int i, int j) throws RemoteException;  
6)     public int mul(int i, int j) throws RemoteException;  
7) }
```

### CalculatorRemotelImpl.java

```
1) import java.rmi.*;  
2) import java.rmi.server.*;  
3) public class CalculatorRemotelImpl extends UnicastRemoteObject implements  
CalculatorRemote  
4) {  
5)     public CalculatorRemotelImpl() throws RemoteException  
6)     {  
7)     }  
8)     public int add(int i, int j) throws RemoteException  
9)     {  
10)        return i+j;  
11)    }  
12)    public int sub(int i, int j) throws RemoteException  
13)    {  
14)        return i-j;  
15)    }  
16)    public int mul(int i, int j) throws RemoteException  
17)    {  
18)        return i*j;  
19)    }  
20) }
```



## CalculatorRegistry.java

```
1) import java.rmi.*;
2) public class CalculatorRegistry
3) {
4)     public static void main(String[] args) throws Exception
5)     {
6)         CalculatorRemote cr=new CalculatorRemoteImpl();
7)         Naming.bind("cal", cr);
8)         System.out.println("CalculatorRemote object is binded with the logical
9)                             name 'cal' in RMIRegistry");
10} }
```

## ClientApp.java

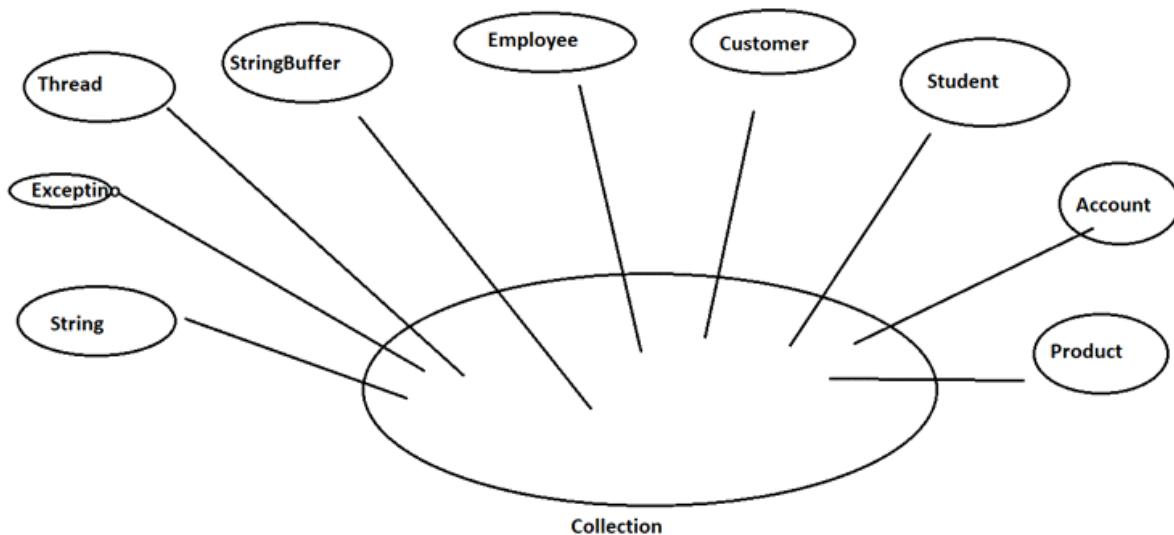
```
1) import java.rmi.*;
2) public class ClientApp
3) {
4)     public static void main(String[] args) throws Exception
5)     {
6)         CalculatorRemote cr=(CalculatorRemote) Naming.lookup("cal");
7)         System.out.println("ADD :" +cr.add(10,5));
8)         System.out.println("SUB :" +cr.sub(10,5));
9)         System.out.println("MUL :" +cr.mul(10,5));
10} }
11} }
```



# Collections



Collection is an object, it able to represent a group of other objects.



**Q) In Java Applications, To Represent A Group Of Other Elements We Have Already Arrays Then What Is The Requirement To Use Collections?**

**OR**

**Q) What Are The Differences Between Array And Collection?**

- Arrays are having fixed size in nature. In case of arrays, we are able to add the elements up to the specified size only, we are unable to add the elements over its size, if we are trying to add elements over its size then JVM will rise an exception like "java.lang.ArrayIndexOutOfBoundsException".

**EX:**

```
Student[] std=new Student[3];
std[0]=new Student();
std[1]=new Student();
std[2]=new Student();
std[3]=new Student();--> ArrayIndexOutOfBoundsException
```

Collections are having dynamically growable nature, even if we add the elements over its size then JVM will not rise any exception.

**EX:**

```
ArrayList al=new ArrayList(3);
al.add(new Student());
al.add(new Student());
al.add(new Student());
al.add(new Student());--> No Exception
```



- In Java, by default, Arrays are able to allow homogeneous elements, if we are trying to add the elements which are not same Array data type then Compiler will rise an error like "Incompatible Types".

**EX:**

```
Student[] std=new Student[3];
std[0]=new Student();
std[1]=new Student();
std[2]=new Customer();--> Incompatible Types Error
```

In Java, by default, Collections are able to allow heterogeneous elements, even we add different types of elements Compiler will not rise any error.

**EX:**

```
ArrayList al=new ArrayList(3);
al.add(new Student());
al.add(new Employee());--> No Error
al.add(new Customer());----> No Error
```

- Arrays are not having predefined methods to perform searching and sorting operations over the elements, in case of arrays to perform searching and sorting operations developers have to provide their own logic.

In case of Collections, predefined methods or predefined Collections are defined to perform Searching and sorting operations over the elements.

**EX:** In Collections, TreeSet was provided to perform sorting order.

```
TreeSet ts=new TreeSet();
ts.add("B");
ts.add("E");
ts.add("A");
ts.add("D");
ts.add("C");
ts.add("F");
System.out.println(ts);
```

**OUTPUT:** [A,B,C,D,E,F]

- Arrays are able to allow only one type of elements, so Arrays are able to improve Typedness in java applications and they are able to perform Type safe operations.

Collections are able to allow different types of elements, so Collections are able to reduce typedness in java applications and they are unable to perform Type safe operations.

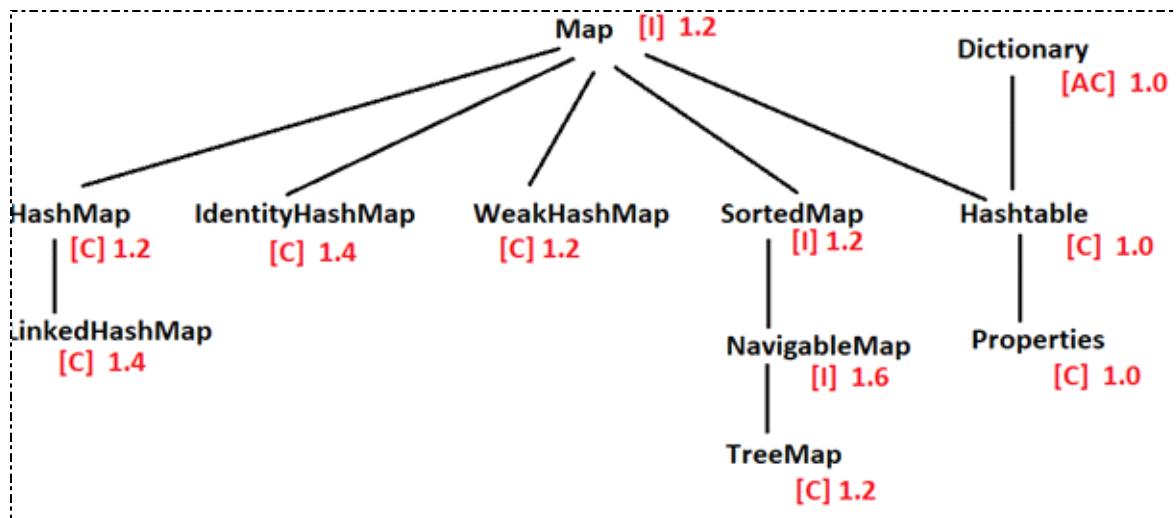
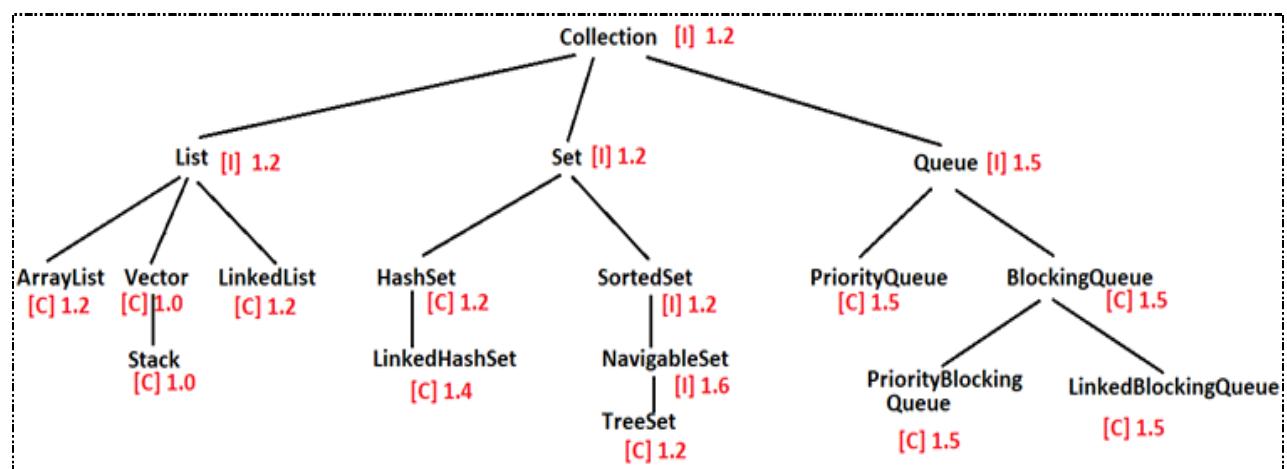


- If we know the no of elements in advance at the time of writing java applications then Arrays are better to use in java applications and they will provide very good performance in java applications, but, But Arrays are not flexible to design applications.

In java applications, Collections are able to provide less performance, but, they will provide flexibility to design applications.

To represent Collection objects in java applications, JAVA has provided predefined classes and interfaces in the form of `java.util` package called as "Collection Framework".

## Q) What are the classes and interfaces are existed in `java.util` package to represent Collections?





## **Q) What are the differences between Collection and Map?**

Collections are able to store all the elements individually, not in the form of Key-value pairs.

**EX:** To store 10 Employee objects we will use Collection.

Maps are able to store all the elements in the form of Key-value pairs.

**EX:** To represent Telephone Directory, where we are representing phone number and Customer Name we have to use Maps.

## **Q) What are the differences between List and Set?**

- List is index based, it able to allow all the elements as per indexing.  
Set is not index based, it able to allow all the elements on the basis of elements hash code values.
- List is able to allow duplicate elements.  
Set is not allowing duplicate elements.
- List is able to allow any number of null values.  
Set is able to allow only one null value.
- List is following insertion order.  
Set is not following insertion order by default.  
**Note:** HashSet is following insertion order.
- List is not following sorting order.  
Sets are not following sorting order by default.  
**Note:** SortedSet, NavigableSet and TreeSet are following Sorting order.
- List is able to allow heterogeneous elements.  
Sets are able to allow heterogeneous elements by default.  
**Note:** SortedSet, NavigableSet and TreeSet are allowing only Homogeneous elements.

## **Collection:**

- It is an interface provided by JAVA along with JDK 1.2 version.
- It able to represent a group of individual elements as single unit.
- It has provided the following methods common to every implementation class.

### **public boolean add(Object obj)**

This method is able to add the specified element to Collection object. If the specified element is added successfully then add(-) method will return "true" value. If the specified element is not added successfully then add() method will return "false" value.



## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         HashSet hs=new HashSet();
7)         System.out.println(hs.add("A"));
8)         hs.add("B");
9)         hs.add("C");
10)        hs.add("D");
11)        System.out.println(hs);
12)        System.out.println(hs.add("A"));
13)        System.out.println(hs);
14)    }
15} }
```

## OUTPUT:

true  
[A,B,C,D]  
false  
[A,B,C,D]

### • public boolean addAll(Collection c)

This method can be used to add all the elements of the specified Collection to the present Collection object. If addition operation is success then addAll() method will return "true" value, if addition operation is failure then addAll() method will return "false" value.

## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         HashSet hs=new HashSet();
7)         hs.add("A");
8)         hs.add("B");
9)         hs.add("C");
10)        hs.add("D");
11)        System.out.println(hs);
12)        HashSet hs1=new HashSet();
13)        System.out.println(hs1.addAll(hs));
14)        System.out.println(hs1);
15)        System.out.println(hs1.addAll(hs));
```



```
16)     System.out.println(hs1);
17) }
18) }
```

**OUTPUT:**

[D,A,B,C]

true

[D,A,B,C]

false

[D,A,B,C]

- **public boolean remove(Object obj)**

This method can be used to remove the specified element from the Collection object. If remove operation is success then remove() method will return true value, if remove operation is failure then remove() method will return false value.

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList al=new ArrayList();
7)         al.add("A");
8)         al.add("B");
9)         al.add("C");
10)        al.add("D");
11)        System.out.println(al);
12)        System.out.println(al.remove("B"));
13)        System.out.println(al);
14)        System.out.println(al.remove("B"));
15)        System.out.println(al);
16)    }
17) }
```

**OUTPUT:**

[A,B,C,D]

true

[A,C,D]

false

[A,C,D]



- **public boolean removeAll(Collection c)**

This method can be used to remove all the elements of the specified Collection from the present Collection object. If remove operation is success then removeAll() method will return true value. If remove operation is not success then removeAll() method will return false value.

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList al=new ArrayList();
7)         al.add("A");
8)         al.add("B");
9)         al.add("C");
10)        al.add("D");
11)        al.add("E");
12)        al.add("F");
13)        System.out.println(al);
14)        ArrayList al1=new ArrayList();
15)        al1.add("B");
16)        al1.add("C");
17)        al1.add("D");
18)        System.out.println(al1);
19)        System.out.println(al.removeAll(al1));
20)        System.out.println(al);
21)        System.out.println(al.removeAll(al1));
22)        System.out.println(al);
23)    }
24) }
```

**OUTPUT:**

[A,B,C,D,E,F]

[B,C,D]

true

[A,E,F]

false

[A,E,F]



- **public boolean contains(Object obj)**

This method will check whether the specified element is existed or not in the Collection object. If the specified element is existed then this method will return "true" value. If the specified element is not existed then this method will return "false" value.

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList al=new ArrayList();
7)         al.add("A");
8)         al.add("B");
9)         al.add("C");
10)        al.add("D");
11)        al.add("E");
12)        al.add("F");
13)        System.out.println(al);
14)        System.out.println(al.contains("B"));
15)        System.out.println(al.contains("X"));
16)    }
17) }
```

**OUTPUT:**

```
[A,B,C,D,E,F]
true
false
```

- **public boolean containsAll(Collection c)**

This method will check whether all the elements of the specified Collection are available or not in the present Collection object. If all the elements are existed then containsAll() method will return true value, if atleast one element is not existed then containsAll() method will return false value.

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList al=new ArrayList();
7)         al.add("A");
```



```
8)     al.add("B");
9)     al.add("C");
10)    al.add("D");
11)    al.add("E");
12)    al.add("F");
13)    System.out.println(al);
14)    ArrayList al1=new ArrayList();
15)    al1.add("B");
16)    al1.add("C");
17)    al1.add("D");
18)    System.out.println(al.containsAll(al1));
19)    al1.add("X");
20)    al1.add("Y");
21)    System.out.println(al.containsAll(al1));
22} }
23}
```

**OUTPUT:**

[A,B,C,D,E,F]

true

false

- **public boolean retainAll(Collection c)**

This method will remove all the elements from the present Collection object except the elements which are existed in the specified Collection object. If at least one element is removed then retainAll() method will return true value. If no elements are removed then retainsAll() method will return false value.

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList al=new ArrayList();
7)         al.add("A");
8)         al.add("B");
9)         al.add("C");
10)        al.add("D");
11)        al.add("E");
12)        al.add("F");
13)        System.out.println(al);
14)        ArrayList al1=new ArrayList();
15)        al1.add("B");
16)        al1.add("C");
```



```
17) al1.add("D");
18) System.out.println(al1);
19) System.out.println(al.retainAll(al1));
20) System.out.println(al);
21) System.out.println(al.retainAll(al1));
22) System.out.println(al);
23) }
24} }
```

**OUTPUT:**

[A,B,C,D,E,F]

[B,C,D]

true

[B,C,D]

false

[B,C,D]

- **public int size()**

This method can be used to return an integer value representing the no of elements which are existed in the Collection object.

- **public void clear()**

This method can be used to remove all elements from Collection object.

- **public boolean isEmpty()**

This method can be used to check whether Collection object is empty or not. If the Collection object is empty then isEmpty() method will return "true" value. If the Collection object is not empty then isEmpty() method will return "false" value.

- **public Object[] toArray()**

This method will return all the elements of the Collection object in the form of Object[].

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList al=new ArrayList();
7)         al.add("A");
8)         al.add("B");
9)         al.add("C");
```



```
10) al.add("D");
11) al.add("E");
12) al.add("F");
13) System.out.println(al);
14) System.out.println(al.size());
15) Object[] obj=al.toArray();
16) for(Object o: obj)
17) {
18)     System.out.print(o+ " ");
19) }
20) System.out.println();
21) System.out.println(al.isEmpty());
22) al.clear();
23) System.out.println(al.isEmpty());
24) System.out.println(al);
25)
26} }
```

#### OUTPUT:

[A,B,C,D,E,F]

6

A B C D E F

false

true

[]

## List:

- List is a direct child interface to Collection interface
- List was provided by JAVA along with its JDK1.2 version
- List is index based, it able to arrange all the elements as per indexing.
- List is able to allow duplicate elements.
- List is following insertion order.
- List is not following Sorting order.
- List is able to allow any number of null values.
- List is able to allow heterogeneous elements.

List interface has provided the following methods common to all of its implementation classes.

- **public void add(int index, Object obj)**

It able to add the specified element at the specified index value.

- **public Object set(int index, Object obj)**

It able to set the specified element at the specified index value.



## Q) What is the difference between *add(–) Method* and *set(–) Method?*

- *add(–) method* is able to perform insert operation. If any element is existed at the specified element then *add()* method will insert the specified new element at the specified index value and *add()* method will adjust the existed element to next index value. If no element is existed at the specified index then *add()* method add the specified element at the specified index.
- *set(–) method* is able to perform replace operation. If any element is existed at the specified index then *set()* method will remove the existed element and *set(–)* method will add the specified element to the specified index and *set()* method will return the removed element. If no element is existed at the specified index value then *set()* method will rise an exception like `java.lang.IndexOutOfBoundsException`.

- **public Object get(int index)**

It will return an element available at the specified index value.

- **public Object remove(int index)**

It will remove and return an element available at the specified index value.

- **public int indexOf(Object obj)**

It will return an index value where the first occurrence of the specified element.

- **public int lastIndexOf(Object obj)**

It will return an index value where the last occurrence of the specified element.

### EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList al=new ArrayList();
7)         al.add("A");
8)         al.add("B");
9)         al.add("C");
10)        al.add("D");
11)        al.add("E");
12)        System.out.println(al);
13)        al.add(1,"X");
14)        System.out.println(al);
15)        al.add(6,"F");
```



```
16) System.out.println(al);
17) al.set(3,"Y");
18) System.out.println(al);
19) //al.set(7,"Z");--->IndexOutOfBoundsException
20) System.out.println(al.get(4));
21) System.out.println(al.remove(6));
22) System.out.println(al);
23) al.add(6,"X");
24) al.add(7,"B");
25) al.add(8,"X");
26) System.out.println(al);
27) System.out.println(al.indexOf("X"));
28) System.out.println(al.lastIndexOf("X"));
29)
30}
```

## ArrayList:

- It was provided by JAVA along with JDK 1.2 version.
- It is a direct implementation class to List interface.
- It is index based.
- It allows duplicate elements.
- It follows insertion order.
- It will not follow sorting order.
- It allows heterogeneous elements.
- It allows any number of null values.
- Its internal data structure is "Resizable Array".
- Its initial capacity is 10 elements.
- Its incremental capacity ration is  
$$\text{new\_Capacity} = (\text{Current\_Capacity} * 3/2) + 1$$
- It is best option for frequent retrieval operations.
- It is not synchronized.
- No method is synchronized method in ArrayList.
- It allows more than one thread to access data.
- It follows parallel execution.
- It will reduce execution time.
- It will improve application performance.
- It will not give guarantee for data consistency.
- It is not thread safe.
- It is not Legacy Collection.



## Constructors:

- **public ArrayList()**

It can be used to create an empty ArrayList object with 10 elements as default capacity value.

EX: `ArrayList al = new ArrayList();`

- **public ArrayList(int capacity)**

It can be used to create an empty ArrayList object with the specified capacity.

EX: `ArrayList al = new ArrayList(20);`

- **public ArrayList(Collection c)**

It can be used to create an ArrayList object with all the elements of the specified Collection object.

EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList al1=new ArrayList();
7)         al1.add("AAA");
8)         al1.add("BBB");
9)         al1.add("CCC");
10)        al1.add("DDD");
11)        System.out.println(al1);
12)        ArrayList al2=new ArrayList(al1);
13)        System.out.println(al2);
14)    }
15) }
```

OUTPUT:

[AAA,BBB,CCC,DDD]  
[AAA,BBB,CCC,DDD]



## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList al=new ArrayList();
7)         al.add("A");
8)         al.add("B");
9)         al.add("C");
10)        al.add("D");
11)        al.add("E");
12)        System.out.println(al);
13)        al.add("B");
14)        System.out.println(al);
15)        al.add(new Integer(10));
16)        System.out.println(al);
17)        al.add(null);
18)        al.add(null);
19)        System.out.println(al);
20)    }
21) }
```

## Vector:

- It was introduced in JDK1.0 version.
- It is Legacy Collection.
- It is a direct implementation class to List interface.
- It is index based.
- It allows duplicate elements.
- It follows insertion order.
- It will not follow sorting order.
- It allows heterogeneous elements.
- It allows any number of null values.
- Its internal data structure is "Resizable Array".
- Its initial capacity is 10 elements.
- It is best choice for frequent retrieval operations.
- It is not good for frequent insertions and deletion operations.
- Its incremental capacity is double the current capacity.  
$$\text{New\_capacity} = 2 * \text{Current\_Capacity}$$
- It is synchronized element.
- All the methods of vector class are synchronized.
- It allows only one thread at a time.
- It follows sequential execution.
- It will increase execution time.
- It will reduce application performance.



- It is giving guarantee for data consistency.
- It is threadsafe.

## Constructors:

- **public Vector()**

It can be used to create an empty Vector object with the initial capacity 10 elements.

**EX:** `Vector v = new Vector();`  
`System.out.println(v.capacity());`  
**OUTPUT:** 10

- **public Vector(int capacity)**

It can be used to create an empty vector object with the specified capacity value.

**EX:** `Vector v = new Vector(20);`  
`System.out.println(v.capacity());`  
**OUTPUT:** 20

- **public Vector(int capacity, int incremental\_Ratio)**

This constructor can be used to create an empty Vector object with the specified initial capacity and with the specified incremental ratio.

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         Vector v=new Vector(5,5);
7)         System.out.println(v.capacity());
8)         for(int i=1;i<=6;i++)
9)         {
10)             v.add(i);
11)         }
12)         System.out.println(v.capacity());
13)         for(int i=7;i<=11;i++)
14)         {
15)             v.add(i);
16)         }
17)         System.out.println(v.capacity());
18)     }
19) }
```



## OUTPUT:

5  
10  
15

- **public Vector(Collection c)**

This constructor can be used to create Vector object with all the elements of the specified Collection object.

## EX:

```
1) import java.util.*;  
2) class Test  
3) {  
4)     public static void main(String[] args)  
5)     {  
6)         Vector v=new Vector();  
7)         v.add("A");  
8)         v.add("B");  
9)         v.add("C");  
10)        v.add("D");  
11)        System.out.println(v);  
12)        Vector v1=new Vector(v);  
13)        System.out.println(v1);  
14)    }  
15) }
```

## OUTPUT:

[A,B,C,D]  
[A,B,C,D]

## Methods:

- **public void addElement(Object obj)**

It will add the specified element to Vector.

- **public Object firstElement()**

It will return first element of the Vector.

- **public Object lastElement()**

It will return last element of the Vector.



- **public Object elementAt(int index)**  
It will return an element available at the specified index.
- **public void removeElement(Object obj)**  
It will remove the specified element from Vector.
- **public void removeElementAt(int index)**  
It will remove an element existed at the specified index value.
- **public void removeAllElements()**  
It will remove all elements from Vector.

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         Vector v=new Vector();
7)         v.addElement("A");
8)         v.addElement("B");
9)         v.addElement("C");
10)        v.addElement("D");
11)        v.addElement("E");
12)        System.out.println(v);
13)        System.out.println(v.firstElement());
14)        System.out.println(v.lastElement());
15)        System.out.println(v.elementAt(3));
16)        v.removeElement("D");
17)        System.out.println(v);
18)        v.removeElementAt(2);
19)        System.out.println(v);
20)        v.removeAllElements();
21)        System.out.println(v);
22)    }
23} }
```



### **Q) What are the differences between ArrayList and Vector?**

- **ArrayList Class was introduced in JDK 1.2 Version**  
Vector class was introduced in JDK1.0 version.
- **ArrayList is not Legacy Collection**  
Vector is Legacy Collection.
- **ArrayList is not synchronized**  
Vector is synchronized.
- **No method is synchronized Method in ArrayList**  
Almost all the methods are synchronized methods in vector.
- **ArrayList allows more than one Thread at a Time to access Data**  
Vector allows only one thread at a time to access data.
- **ArrayList follows parallel Execution**  
Vector follows sequential execution.
- **ArrayList is able to reduce Application Execution Time**  
Vector is able to increase application execution time.
- **ArrayList is able to improve Application Performance**  
Vector is able to reduce application performance.
- **ArrayList is not giving guarantee for data consistency.**  
Vector is giving guarantee for data consistency.
- **ArrayList is not threadsafe.**  
Vector is threadsafe.
- **ArrayList Incremental Capacity is (Current Capacity\*3/2)+1**  
Vector incremental capacity is  $2 * \text{Current\_Capacity}$
- **We are unable to get Capacity Value of ArrayList, because, no capacity() Method in ArrayList Class.**  
We can get capacity value of Vector, because, capacity() method is existed in vector class.



## Stack:

It was introduced in JDK 1.0 version, it is a Legacy Collection and it is a child class to Vector class. It able to arrange all the elements as per "Last In First Out" [LIFO] alg.

## Constructor:

`public Stack()`

It will create an empty Stack object.

EX: `Stack s = new Stack();`

## Methods:

- **public void push(Object obj)**

It will add the specified element to Stack.

- **public Object pop()**

It will remove and return top of the stack.

- **public Object peek()**

It will return top of the stack.

- **public int search(Object obj)**

It will check whether the specified element is existed or not in the Stack, if the specified element is not existed then it will return '-1' value, if the specified element is existed then it will return its position.

## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         Stack s=new Stack();
7)         s.push("A");
8)         s.push("B");
9)         s.push("C");
10)        s.push("D");
11)        s.push("E");
12)        System.out.println(s);
13)        System.out.println(s.pop());
14)        System.out.println(s);
15)        System.out.println(s.peek());
```



```
16) System.out.println(s);
17) System.out.println(s.search("B"));
18) System.out.println(s.search("X"));
19)
20}
```

## LinkedList:

- It was introduced in JDK1.2 version.
- It is not Legacy Collection.
- It is a direct implementation class to List interface.
- It is index based.
- It allows duplicate elements.
- It follows insertion order.
- It is not following sorting order.
- It allows heterogeneous elements.
- It allows null values in any number.
- Its internal data structure is "Double Linked List".;
- It is best choice for frequent insertions and deletions.
- It is not synchronized Collection.
- No method is synchronized in LinkedList.
- It allows more than one thread to access data.
- It will follow parallel execution.
- It will decrease execution time.
- It will improve application performance.
- It is not giving guarantee for data consistency.
- It is not threadsafe.

## Constructors:

- **public LinkedList()**

It will create an empty LinkedList object.

EX: `LinkedList l1 = new LinkedList();`

- **public LinkedList(Collection c)**

It will create LinkedList object with all the elements of the specified Collection object.

EX:

`LinkedList l1 = new LinkedList();`

`l1.add("A");`

`l1.add("B");`

`l1.add("C");`

`l1.add("D");`



```
System.out.println(l1);
LinkedList l2=new LinkedList(l1);
System.out.println(l2);
```

**OUTPUT:** [A, B, C, D]  
[A, B, C, D]

## Methods:

- **public void addFirst(Object obj)**

It will add the specified element as first element to LinkedList.

- **public void addLast(Object obj)**

It will add the specified element as last element to LinkedList.

- **public Object getFirst()**

It will return first element from LinkedList.

- **public Object getLast()**

It will return last element from LinkedList.

- **public void removeFirst()**

It will remove first element from LinkedList.

- **public void removeLast()**

It will remove last element from LinkedList.

## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         LinkedList l1=new LinkedList();
7)         l1.add("A");
8)         l1.add("B");
9)         l1.add("C");
10)        l1.add("D");
11)        l1.add("E");
12)        System.out.println(l1);
13)        l1.addFirst("X");
14)        l1.addLast("Y");
```



```
15) System.out.println(lI);
16) lI.removeFirst();
17) lI.removeLast();
18) System.out.println(lI);
19) System.out.println(lI.getFirst());
20) System.out.println(lI.getLast());
21)
22}
```

## Cursors / Iterators in Collections:

In java applications, when we pass Collection object reference variable as parameter to `System.out.println()` method, then, JVM will execute `toString()` method internally. Initially `toString()` method was implemented in `java.lang.Object` class, it was implemented in such a way that to return a String contains "Class\_Name@Ref\_val". In java applications, Collection classes are not depending on `Object` class `toString()` method, they are having their own `toString()` method , which are implemented in such a way to return a String contains all the elements of the Collection object by enclosed with [].

### EX:

```
ArrayList al = new ArrayList();
al.add("A");
al.add("B");
al.add("C");
al.add("D");
System.out.println(al);
```

OUTPUT: [A, B, C, D]

As per the requirement, we don't want to display all the Elements at a time on command prompt, we want to retrieve elements one by one individually from Collection objects and we want to display all the elements one by one on Command prompt.

To achieve the above requirement, Collection Framework has provided the following three Cursors or Iterators.

- Enumeration
- Iterator
- ListIterator



## • Enumeration:

- It is a Legacy Cursor, it is applicable for only Legacy Collections to retrieve elements in one by one fashion.
- To retrieve elements from Collections by using Enumeration we have to use the following steps.

## • Create Enumeration Object:

To create Enumeration object we have to use the following method from Legacy Collections.

```
public Enumeration elements()
```

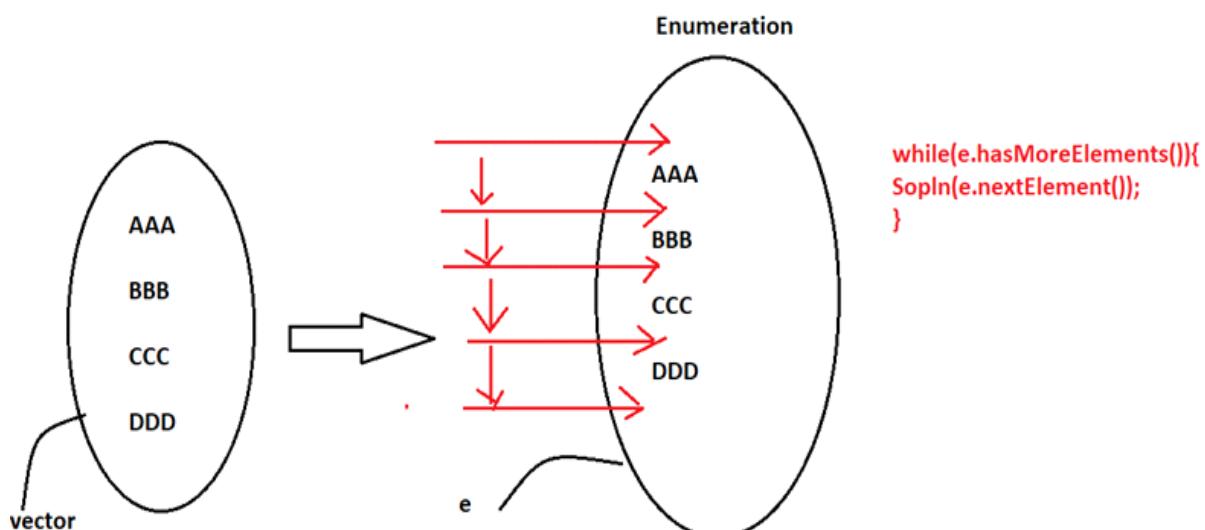
## • Retrieve Elements from Enumeration:

- Check whether more elements are available or not from Current cursor position by using the following method.  

```
public boolean hasMoreElements()
```

  - It will return true value if at least next element is existed.
  - It will return false value if no element is existed from current cursor position.
- If at least next element is existed then read next element and move cursor to next position by using the following method.  

```
public Object nextElement()
```





## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         Vector v=new Vector();
7)         v.add("A");
8)         v.add("B");
9)         v.add("C");
10)        v.add("D");
11)        v.add("E");
12)        System.out.println(v);
13)        Enumeration e=v.elements();
14)        while(e.hasMoreElements())
15)        {
16)            System.out.println(e.nextElement());
17)        }
18)    }
19) }
```

## Drawbacks:

- Enumeration is applicable for only Legacy Collections.
- Enumeration is able to allow only read operation while iterating elements.

## • Iterator:

- Iterator is an interface provided JAVA along with its JDK1.2 version.
- Iterator can be used to retrieve all the elements from Collection objects in one by one fashion.
- Iterator is applicable for all the Collection interface implementation classes to retrieve elements.
- Iterator is able to allow both read and remove operations while iterating elements.
- If we want to use Iterator in java applications then we have to use the following steps.

## • Create Iterator Object:

- To create Iterator object we have to use the following method from all Collection implementation classes.
- `public Iterator iterator()`
- EX: `Iterator it = al.iterator();`



## **• Retrieve Elements from Iterator:**

- To retrieve elements from Iterator we have to use the following steps.
- Check whether next element is existed or not from the current cursor position by using the following method.

### **public boolean hasNext()**

- This method will return true if next element is existed.
- This method will return false if no element is existed from current cursor position.
- If next element is existed then read next element and move cursor to next position by using the following method.
- **public Object next()**

**NOTE:** To remove an element available at current cursor position then we have to use the following method

**public void remove()**

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList al=new ArrayList();
7)         al.add("A");
8)         al.add("B");
9)         al.add("C");
10)        al.add("D");
11)        al.add("E");
12)        System.out.println(al);
13)        Iterator it=al.iterator();
14)        while(it.hasNext())
15)        {
16)            String element=(String)it.next();
17)            System.out.println(element);
18)            if(element.equals("C"))
19)            {
20)                it.remove();
21)            }
22)        }
23)        System.out.println(al);
24)    }
25} }
```



## Q) What are the differences between Enumeration and Iterator?

- Enumeration is Legacy Cursor, it was introduced in JDK1.0 version.  
Iterator is not Legacy Cursor, it was introduced in JDK1.2 version.
- Enumeration is not Universal Cursor, it is applicable for only Legacy Collections.  
Iterator is an universal Cursor, it is applicable for all Collection implementations.
- Enumeration is able to allow only read operation while iterating elements.  
Iterator is able to allow both read operation and remove operation while iterating elements.

### • ListIterator:

- It is an interface provided by JAVA along with JDK1.2 version.
- It able to allow to read elements in both forward direction and backward direction.
- It able to allow the operations like read, insert, replace and remove while iterating elements.
- If we want to use ListIterator in java applications then we have to use the following steps.

### • Create ListIterator Object:

- To create ListIterator object we have to use the following method.
- `public ListIterator listIterator()`
- EX: `ListIterator lit = li.listIterator();`

### • Retrieve Elements from ListIterator

To retrieve elements from ListIterator in Forward direction then we have to use the following methods.

#### public boolean hasNext()

It will check whether next element is existed or not from the current cursor position.

#### public Object next()

It will return next element and it will move cursor to the next position in forward direction.

#### public int nextIndex()

It will return next index value from the current cursor position.



To retrieve elements in Backward direction we have to use the following methods.

## public boolean hasPrevious()

It will check whether previous element is existed or not from the current cursor position, If previous element is existed then it will return "true" value, if previous element is not existed then it will return false value.

## public Object previous()

It will return previous element and it will move cursor to the next previous position.

## public int previousIndex()

It will return previous index value from the current cursor position.

To perform the operations like remove, insert and replace over the elements we have to use the following methods.

- public void remove()
- public void add(Object obj)
- public void set(Object obj)

EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         LinkedList ll=new LinkedList();
7)         ll.add("A");
8)         ll.add("B");
9)         ll.add("C");
10)        ll.add("D");
11)        ll.add("E");
12)        ll.add("F");
13)        System.out.println(ll);
14)        ListIterator lit=ll.listIterator();
15)        System.out.println("Elements in Forward Direction");
16)        while(lit.hasNext())
17)        {
18)            System.out.println(lit.nextInt()+"--->"+lit.next());
19)        }
20)        System.out.println();
21)        System.out.println("Elements in Backward Direction");
22)        while(lit.hasPrevious())
23)        {
24)            System.out.println(lit.previousIndex()+"--->"+lit.previous());
```



```
25)    }
26) }
27) }
```

**EX:**

```
1) import java.util.*;
2) class Test1
3) {
4)     public static void main(String[] args)
5)     {
6)         LinkedList ll=new LinkedList();
7)         ll.add("A");
8)         ll.add("B");
9)         ll.add("C");
10)        ll.add("D");
11)        ll.add("E");
12)        ll.add("F");
13)        System.out.println(ll);
14)        ListIterator lit=ll.listIterator();
15)        while(lit.hasNext())
16)        {
17)            String element=(String)lit.next();
18)            if(element.equals("B"))
19)            {
20)                lit.add("X");
21)            }
22)            if(element.equals("D"))
23)            {
24)                lit.set("Y");
25)            }
26)            if(element.equals("E"))
27)            {
28)                lit.remove();
29)            }
30)        }
31)        System.out.println(ll);
32)    }
33} }
```



## Q) What are the differences between *Enumeration*, *Iterator* and *ListIterator*?

- Enumeration is applicable for only Legacy Collections.  
Iterator is applicable for all Collection implementations.  
ListIterator is applicable for only List implementations.
- Enumeration and Iterator are allowed to iterate elements in only Forward direction.  
ListIterator is able to allow to iterate elements in both Forward direction and backward direction.
- Enumeration is able to allow only read operation while iterating elements.  
Iterator is able to allow both read and remove operations while iterating elements.  
ListIterator is able to allow the operations like insert, replace, remove and read operations while iterating elements.

## Set:

- It was introduced in JDK 1.2 version.
- It is a direct child interface to Collection interface.
- It is not index based, it able to arrange all the elements on the basis of elements hashcode values.
- It will not allow duplicate elements.
- It will not follow insertion order.  
**Note:** LinkedHashSet will follow insertion order.
- It will not follow Sorting order.  
**Note:** SortedSet, NavigableSet and TreeSet are following Sorting order.
- It able to allow only one null value.  
**Note:** SortedSet, NavigableSet and TreeSet are not allowing even single null value.

## HashSet:

- HashSet is a direct implementation class to Set interface.
- It was introduced in JDK 1.2 version.
- It is not index based, it able to arrange all the elements on the basis of elements hashcode values.
- It will not allow duplicate elements.
- It will not follow insertion order.
- It will not follow Sorting order.
- It able to allow only one null value.
- Its interal data strucuter is "Hashtable".
- Its initial capacity is "16" elements and its initial fill\_Ratio is 75%.
- It is not synchronized.
- Almost all the methods are not synchronized in HashSet



- It allows more than one thread at a time.
- It follows parallel execution.
- It will reduce execution time.
- It improves performance of the applications.
- It is not giving guarantee for data consistency.
- It is not threadsafe.

## Constructors:

- **public HashSet()**

This constructor can be used to create an empty HashSet object with 16 elements as initial capacity and 75% fill ratio.

EX: `HashSet hs = new HashSet();`

- **public HashSet(int capacity)**

This constructor can be used to create an empty HashSet object with the specified capacity as initial capacity and with the default fill ratio 75%.

EX: `HashSet hs = new HashSet(20);`

- **public HashSet(int capacity, float fill Ratio)**

This constructor can be used to create an empty HashSet object with the specified capacity and with the specified fill ratio.

EX: `HashSet hs = new HashSet(20, 0.85f);`

- **public HashSet(Collection c)**

This constructor can be used to create HashSet object with all the elements of the specified Collection.

EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         HashSet hs1=new HashSet();
7)         hs1.add("A");
8)         hs1.add("B");
9)         hs1.add("C");
10)        hs1.add("D");
11)        hs1.add("E");
12)        System.out.println(hs1);
13)        HashSet hs2=new HashSet(hs1);
```



```
14)     System.out.println(hs2);
15) }
16) }
```

## OUTPUT:

[D,E,A,B,C]  
[D,E,A,B,C]

## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         HashSet hs=new HashSet();
7)         hs.add("A");
8)         hs.add("B");
9)         hs.add("C");
10)        hs.add("D");
11)        hs.add("E");
12)        System.out.println(hs);
13)        hs.add("B");
14)        System.out.println(hs);
15)        hs.add(null);
16)        hs.add(null);
17)        System.out.println(hs);
18)        hs.add(new Integer(10));
19)        System.out.println(hs);
20)    }
21) }
```

## LinkedHashSet:

### Q) What are the differences between *HashSet* and *LinkedHashSet*?

- HashSet was introduced in JDK 1.2 version.  
LinkedhashSet was introduced in JDK 1.4 version.
- HashSet is not following insertion order.  
LinkedHashSet is following insertion order.
- The internal data strucer of HashSet is "Hashtable".  
The internal data strucer of LinkedHashSet is "Hashtable" and "LinkedList".



## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         HashSet hs=new HashSet();
7)         hs.add("A");
8)         hs.add("B");
9)         hs.add("C");
10)        hs.add("D");
11)        hs.add("E");
12)        System.out.println(hs);
13)        LinkedHashMap lhs=new LinkedHashMap();
14)        lhs.add("A");
15)        lhs.add("B");
16)        lhs.add("C");
17)        lhs.add("D");
18)        lhs.add("E");
19)        System.out.println(lhs);
20)    }
21) }
```

## OUTPUT:

[D,E,A,B,C]  
[A,B,C,D,E]

## SortedSet:

- It was introduced in JDK1.2 version.
- It is a child interface to Set interface.
- It is not index based.
- It is not allowing duplicate elements.
- It is not following insertion order.
- It follows Sorting order.
- It allows only homogeneous elements.
- It will not allow heterogeneous elements, if we are trying to add heterogeneous elements then JVM will rise an exception like `java.lang.ClassCastException`.
- It will not allow null values, if we are trying to add any null value then JVM will rise an exception like `java.lang.NullPointerException`.
- It able to allow only Comparable objects by default, if we are trying to add non comparable objects then JVM will rise an exception like `java.lang.ClassCastException`.

**Note:** If we are trying to add non comparable objects then we have to use Comparator.



## Methods:

- **public Object first()**

It will return first element from SortedSet.

- **public Object last()**

It will return last element from SortedSet.

- **public SortedSet headSet(Object obj)**

It will return SortedSet object with the elements which are less than the specified element.

- **public SortedSet tailSet(Object obj)**

It will return SortedSet object with the elements which are greater than or equals to the specified element.

- **public SortedSet subSet(Object obj1, Object obj2)**

It will return SortedSet object with all elements which are greater than or equals to the specified first element and which are less than the specified second element.

### EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         TreeSet ts=new TreeSet();
7)         ts.add("D");
8)         ts.add("F");
9)         ts.add("B");
10)        ts.add("E");
11)        ts.add("C");
12)        ts.add("A");
13)        System.out.println(ts);
14)        System.out.println(ts.first());
15)        System.out.println(ts.last());
16)        System.out.println(ts.headSet("D"));
17)        System.out.println(ts.tailSet("D"));
18)        System.out.println(ts.subSet("B","E"));
19)    }
20) }
```



## NavigableSet

It was introduced in JAVA6 version, it is a child interface to SortedSet interface, it is following all the properties of SortedSet and it has define methods to provide navigations over the elements.

### Methods:

- **public Object ceiling(Object obj)**

It will return lowest element among all the elements which are greater than or equals to the specified element.

- **public Object higher(Object obj)**

It will return lowest element among all the elements which are greater than the specified element.

- **public Object floor(Object obj)**

It will return highest element among all the elements which are less than or equals to the specified element.

- **Trpublic Object lower(Object obj)**

It will return highest element among all the elements which are less than the specified element.

- **public Object pollFirst()**

It will remove and return first element from NavigableSet.

- **public Object pollLast()**

It will remove and return last element from NavigableSet.

- **public NavigableSet descendingSet()**

It will return all elements in the form of NavigableSet in descending order.

### EX:

```
1) import java.util.*;  
2) class Test  
3) {  
4)     public static void main(String[] args)  
5)     {  
6)         TreeSet ts=new TreeSet();  
7)         ts.add("D");
```



```
8)    ts.add("F");
9)    ts.add("B");
10)   ts.add("E");
11)   ts.add("C");
12)   ts.add("A");
13)   System.out.println(ts);
14)   System.out.println(ts.ceiling("D"));
15)   System.out.println(ts.higher("D"));
16)   System.out.println(ts.floor("D"));
17)   System.out.println(ts.lower("D"));
18)   System.out.println(ts.descendingSet());
19)   ts.pollFirst();
20)   ts.pollLast();
21)   System.out.println(ts);
22) }
23) }
```

## TreeSet:

- It was introduced in JDK 1.2 version.
- It is not Legacy Collection.
- It has provided implementation for Collection, Set, SortedSet and NavigableSet interfaces.
- It is not index based.
- It is not allowing duplicate elements.
- It is not following insertion order.
- It follows Sorting order.
- It allows only homogeneous elements.
- It will not allow heterogeneous elements, if we are trying to add heterogeneous elements then JVM will rise an exception like java.lang.ClassCastException.
- It will not allow null values, if we are trying to add any null value then JVM will rise an exception like java.lang.NullPointerException.
- It able to allow only Comparable objects by default, if we are trying to add non comparable objects then JVM will rise an exception like java.lang.ClassCastException.

**NOTE:** If we are trying to add non comparable objects then we have to use java.util.Comparator.

- Its internal data structure is "Balanced Tree".
- It is mainly for frequent search operations.



## Constructors:

- **public TreeSet()**

It can be used to create an Empty TreeSet object.

EX: `TreeSet ts = new TreeSet();`

- **public TreeSet(Comparator c)**

It will create an empty TreeSet object with the explicit Sorting mechanism in the form of Comparator

EX: `TreeSet ts = new TreeSet(new MyComparator());`

- **public TreeSet(SortedSet ts)**

It will create TreeSet object with all elements of the specified SortedSet.

EX:

```
1) import java.util.*;  
2) class Test  
3) {  
4)     public static void main(String[] args)  
5)     {  
6)         TreeSet ts1=new TreeSet();  
7)         ts1.add("B");  
8)         ts1.add("C");  
9)         ts1.add("F");  
10)        ts1.add("A");  
11)        ts1.add("E");  
12)        ts1.add("D");  
13)        System.out.println(ts1);  
14)        TreeSet ts2=new TreeSet(ts1);  
15)        System.out.println(ts2);  
16)    }  
17} }
```



## 4) public TreeSet(Collection c)

It able to create TreeSet object with all the elements of the specified Collection.

EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList al=new ArrayList();
7)         al.add("B");
8)         al.add("C");
9)         al.add("F");
10)        al.add("A");
11)        al.add("E");
12)        al.add("D");
13)        System.out.println(al);
14)        TreeSet ts=new TreeSet(al);
15)        System.out.println(ts);
16)    }
17) }
```

EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         TreeSet ts=new TreeSet();
7)         ts.add("B");
8)         ts.add("C");
9)         ts.add("F");
10)        ts.add("A");
11)        ts.add("E");
12)        ts.add("D");
13)        System.out.println(ts);
14)        ts.add("B");
15)        System.out.println(ts);
16)        //ts.add(null);--> NullPointerException
17)        //ts.add(new Integer(10));-->ClassCastException
18)        //ts.add(new StringBuffer("BBB"));-->ClassCastException
19)    }
20) }
```



When we add elements to the TreeSet object, TreeSet object will arrange all the elements in a particular sorting order with the following algorithm.

- TreeSet will construct a Tree [Balanced Tree] on the basis of the elements.

To construct Balanced Tree we have to use the following steps.

- If the element is first element to the TreeSet object then make that element as "Root Node".
- If the element is not first element then access compareTo(-) method over the present element by passing previous elements one by one of the balanced Tree right from root node until the present element is located in Tree.
  - If compareTo(-) method returns -ve value then go to left child of the present node and access again compareTo(-) method by passing left child. If no left child is existed then make the present element as left child
  - If compareTo(-) method returns +ve value then go to right child and access again compareTo(-) by passing right as parameter. If no right child is existed then make the present element as right child.
  - If compareTo(-) method return 0 value then discard the present element and declare that the present element is a duplicate element of the existed element.
- TreeSet will retrieve all the elements from balanced Tree by following in order traversal.

In String class, compareTo() method was implemented like below

```
str1.compareTo(str2);
```

- If str1 come first when compared with str2 as per dictionary order then compareTo() method will return -ve value.
- If str2 come first when compared with str1 in dictionary order then compareTo() method will return +ve value.
- If str1 and str2 are same or available at same location in dictionary order then compareTo(-) method will return 0 value.

If we want to add user defined elements like Employee, Student, Customer to TreeSet then we have to use the following steps.

- Declare an user defined class.
- Implement java.lang.Comparable interface in User defined class.
- Provide implementation for compareTo() method in user defined class.
- In main class, in main() method, create objects for user defined class and add objects to TreeSet object.



```
1) package com.durgasoftware.app05.test;
2)
3) import java.util.TreeSet;
4)
5) class Employee implements Comparable{
6)     int eno;
7)     String ename;
8)     float esal;
9)     String eaddr;
10)
11)    public Employee(int eno, String ename, float esal, String eaddr){
12)        this.eno = eno;
13)        this.ename = ename;
14)        this.esal = esal;
15)        this.eaddr = eaddr;
16)    }
17)
18)    @Override
19)    public int compareTo(Object obj) {
20)        Employee emp = (Employee)obj;
21)        int val = 0;
22)        val = this.ename.compareTo(emp.ename);
23)        return -val;
24)    }
25)
26)    @Override
27)    public String toString() {
28)        return "\nEmployee{" +
29)            "eno=" + eno +
30)            ", ename=\"" + ename + '\"' +
31)            ", esal=" + esal +
32)            ", eaddr=\"" + eaddr + '\"' +
33)            '}';
34)    }
35) }
36) public class Test {
37)     public static void main(String[] args) {
38)         Employee emp1 = new Employee(111, "Durga", 5000, "Hyd");
39)         Employee emp2 = new Employee(222, "Neelesh", 6000, "Hyd");
40)         Employee emp3 = new Employee(333, "Aishwary", 7000, "Hyd");
41)         Employee emp4 = new Employee(444, "Gopi Krishna", 8000, "Hyd");
42)         Employee emp5 = new Employee(555, "Tahauddin", 9000, "Hyd");
43)
44)         TreeSet ts = new TreeSet();
45)         ts.add(emp1);
```



```
46)    ts.add(emp2);
47)    ts.add(emp3);
48)    ts.add(emp4);
49)    ts.add(emp5);
50)
51)    System.out.println(ts);
52) }
53) }
```

If we add non-comparable objects to TreeSet object then JVM will rise an exception like `java.lang.ClassCastException`, because, Non-Comparable objects are not providing `compareTo()` method internally, but, it is required to the TreeSet inorder to provide sorting order over elements.

If we want to add non-Comparable objects to TreeSet object then we must provide sorting logic to TreeSet object explicitly, for this, we have to use `java.util.Comparator` interface.

If we want to use Comparator interface in java applications then we have to use the following steps.

- Declare an User defined class.
- Implement `java.util.Comparator` interface in user defined class.
- Provide implementation for Comparator interface methods in user defined class.

```
public boolean equals(Object obj)
public int compare(Object obj1, Object obj2)
```

**Note:** In User defined class it is not required to implement `equals()` method, because, `equals()` method will come from default super class `Object`.

- Provide User defined Comparator object to TreeSet object

**EX:** `MyComparator mc = new MyComparator();`  
`TreeSet ts = new TreeSet(mc);`

**EX:**

```
1) import java.util.*;
2) class MyComparator implements Comparator
3) {
4)     public int compare(Object obj1, Object obj2)
5)     {
6)         StringBuffer s1=(StringBuffer)obj1;
7)         StringBuffer s2=(StringBuffer)obj2;
8)
9)         int length1=s1.length();
10)        int length2=s2.length();
11)        int val=0;
```



```
12) if(length1<length2)
13) {
14)     val=-100;
15) }
16) else if(length1>length2)
17) {
18)     val=100;
19) }
20) else
21) {
22)     val=0;
23) }
24) return -val;
25) }
26}
27) class Test
28{
29)     public static void main(String[] args)
30)     {
31)         StringBuffer sb1=new StringBuffer("AAA");
32)         StringBuffer sb2=new StringBuffer("BB");
33)         StringBuffer sb3=new StringBuffer("CCCC");
34)         StringBuffer sb4=new StringBuffer("D");
35)         StringBuffer sb5=new StringBuffer("EEEE");
36)         MyComparator mc=new MyComparator();
37)         TreeSet ts=new TreeSet(mc);
38)         ts.add(sb1);
39)         ts.add(sb2);
40)         ts.add(sb3);
41)         ts.add(sb4);
42)         ts.add(sb5);
43)         System.out.println(ts);
44)     }
45} }
```

EX:

```
1) import java.util.*;
2) class MyComparator implements Comparator
3) {
4)     public int compare(Object obj1, Object obj2)
5)     {
6)         Student s1=(Student)obj1;
7)         Student s2=(Student)obj2;
8)
9)         int val=s1.saddr.compareTo(s2.saddr);
```



```
10)    return -val;
11) }
12) }
13) class Student
14) {
15)     String sid;
16)     String sname;
17)     String saddr;
18)
19)     Student(String sid, String sname, String saddr)
20)     {
21)         this.sid=sid;
22)         this.sname=sname;
23)         this.saddr=saddr;
24)     }
25)     public String toString()
26)     {
27)         return "["+sid+","+sname+","+saddr+"]";
28)     }
29) }
30) class Test
31) {
32)     public static void main(String[] args)
33)     {
34)         Student std1=new Student("S-111", "Durga", "Hyd");
35)         Student std2=new Student("S-222", "Anil", "Chennai");
36)         Student std3=new Student("S-333", "Rahul", "Banglore");
37)         Student std4=new Student("S-444", "Rameshh", "Pune");
38)         MyComparator mc=new MyComparator();
39)         TreeSet ts=new TreeSet(mc);
40)         ts.add(std1);
41)         ts.add(std2);
42)         ts.add(std3);
43)         ts.add(std4);
44)         System.out.println(ts);
45)     }
46} }
```

Q) In Java applications, if we provide both implicit Sorting through Comparable and explicit sorting through Comparator to the TreeSet object at a time then which Sorting logic would be preferred by TreeSet inorder to Sort elements?

If we provide both implicit Sorting through Comparable and Explicit Sorting through Comparator to the TreeSet object at a time then TreeSet will take explicit Sorting through Comparator to sort all the elements.



EX:

```
1) import java.util.*;
2) class MyComparator implements Comparator
3) {
4)     public int compare(Object obj1, Object obj2)
5)     {
6)         Customer cust1=(Customer)obj1;
7)         Customer cust2=(Customer)obj2;
8)
9)         int val=cust1.caddr.compareTo(cust2.caddr);
10)        return -val;
11)    }
12) }
13) class Customer implements Comparable
14) {
15)     String cid;
16)     String cname;
17)     String caddr;
18)
19)     Customer(String cid, String cname, String caddr)
20)     {
21)         this.cid=cid;
22)         this.cname=cname;
23)         this.caddr=caddr;
24)     }
25)     public String toString()
26)     {
27)         return "["+cid+","+cname+","+caddr+"]";
28)     }
29)     public int compareTo(Object obj)
30)     {
31)         Customer cust=(Customer)obj;
32)         int val=this.caddr.compareTo(cust.caddr);
33)         return val;
34)     }
35) }
36) class Test
37) {
38)     public static void main(String[] args)
39)     {
40)         Customer c1=new Customer("C-111", "Durga", "Hyd");
41)         Customer c2=new Customer("C-222", "Anil", "Chennai");
42)         Customer c3=new Customer("C-333", "Rahul", "Banglore");
43)         Customer c4=new Customer("C-444", "Ramesh", "Pune");
44)         MyComparator mc=new MyComparator();
```



```
45) TreeSet ts=new TreeSet(mc);
46) ts.add(c1);
47) ts.add(c2);
48) ts.add(c3);
49) ts.add(c4);
50) System.out.println(ts);
51)
52} }
```

## Queue:

- It was introduced in JDK 5.0 version.
- It is a direct child interface to Collection Interface.
- It able to arrange all the elements as per FIFO [First In First Out], but, it is possible to change this algorithm as per our requirement.
- It able to allow duplicate elements.
- It is not following Insertion order.
- It is following Sorting order.
- It will not allow null values, if we add null value then JVM will rise an Exception like `java.lang.NullPointerException`.
- It will not allow heterogeneous elements, if we add heterogeneous elements then JVM will rise an exception like `java.lang.ClassCastException`.
- It able to allow only homogeneous elements.
- It able to allow comparable objects by default, if we add non comparable objects then JVM will rise an exception like `java.lang.ClassCastException`.
- If we want to add non comparable objects then we have to use Comparator.
- It able to manage all elements prior to process.

## Methods:

- **public void offer(Object obj)**

It can be used to insert the specified element to Queue.

- **public Object peek()**

It can be used to return head element of the Queue.

- **public Object element()**

It can be used to return head element of the Queue

**Note:** If we access `peek()` method on an empty Queue then `peek()` will return "null" value. If we access `element()` method on an empty Queue then `element()` method will rise an exception like `java.util.NoSuchElementException`



- **public Object poll()**

It can be used to return and remove head element from Queue.

- **public Object remove()**

It can be used to return and remove head element from Queue.

**Note:** If we access poll() method on an empty Queue then poll() method will return "null" value. If we access remove() method on an empty Queue then remove() method will rise an exception like "java.util.NoSuchElementException".

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         PriorityQueue q=new PriorityQueue();
7)         q.offer("A");
8)         q.offer("B");
9)         q.offer("C");
10)        q.offer("D");
11)        q.offer("E");
12)        q.offer("F");
13)        System.out.println(q);
14)        System.out.println(q.peek());
15)        System.out.println(q);
16)        System.out.println(q.element());
17)        System.out.println(q);
18)        /*
19)        PriorityQueue q1=new PriorityQueue();
20)        System.out.println(q1.peek());--> Null
21)        System.out.println(q1.element());--> Exception
22)        */
23)        System.out.println(q.poll());
24)        System.out.println(q);
25)        System.out.println(q.remove());
26)        System.out.println(q);
27)        /*
28)        PriorityQueue q1=new PriorityQueue();
29)        System.out.println(q1.poll());--> Null
30)        System.out.println(q1.remove());-->Exception
31)        */
32)    }
33) }
```



## PriorityQueue:

- It was introduced in JDK 5.0 version.
- It is not Legacy Collection.
- It is a direct implementation class to Queue interface.
- It able to arrange all the elements prior to processing on the basis of the priorities.
- It able to allow duplicate elements.
- It is not following Insertion order.
- It is following Sorting order.
- It will not allow null values, if we add null value then JVM will rise an Exception like `java.lang.NullPointerException`.
- It will not allow heterogeneous elements, if we add heterogeneous elements then JVM will rise an exception like `java.lang.ClassCastException`.
- It able to allow only homogeneous elements.
- It able to allow comparable objects by default, if we add non comparable objects then JVM will rise an exception like `java.lang.ClassCastException`.
- If we want to add non comparable objects then we have to use Comparator.
- Its initial capacity 11 elements.
- It is not synchronized .
- No method is synchronized in PriorityQueue.
- It allows more than one thread at a time to access data.
- It follows parallel execution.
- It able to reduce application execution time.
- It able to improve application performance.
- It is not giving guarantee for Data consistency.
- It is not threadsafe.

## Constructors:

### • public PriorityQueue()

It able to create an empty PriorityQueue object

EX: `PriorityQueue p = new PriorityQueue();`

### • public PriorityQueue(int capacity)

It can be used to create an empty Queue with the specified capacity.

EX: `PriorityQueue p = new PriorityQueue(20);`

### • public PriorityQueue(int capacity, Comparator c)

It able to create an empty PriorityQueue with explicit sorting logic through Comparator and the specified capacity.

EX: `MyComparator mc = new MyComparator();  
PriorityQueue p = new PriorityQueue(20,mc);`



- **public PriorityQueue(SortedSet ss)**

It able to create PriorityQueue object with all the elements of the specified SortedSet.

EX:

```
TreeSet ts=new TreeSet();
ts.add("B");
ts.add("E");
ts.add("C");
ts.add("A");
ts.add("D");
System.out.println(ts);
PriorityQueue p = new PriorityQueue(ts);
System.out.println(p);
```

- **public PriorityQueue(Collection c)**

It able to create PriorityQueue object with all the elements of the specified Collection.

EX:

```
ArrayList al = new ArrayList();
al.add("A");
al.add("B");
al.add("C");
al.add("D");
System.out.println(al);
PriorityQueue p = new PriorityQueue(al);
System.out.println(p);
```

EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         PriorityQueue p=new PriorityQueue();
7)         p.add("A");
8)         p.add("D");
9)         p.add("B");
10)        p.add("C");
11)        p.add("F");
12)        p.add("E");
13)        System.out.println(p);
14)        p.add("B");
15)        System.out.println(p);
16)        //p.add(null);-->NullPointerException
17)        //p.add(new Integer(10));->ClassCastException
```



```
18)    //p.add(new StringBuffer("BBB"));->ClassCastException  
19) }  
20) }
```

## Map:

- It was introduced in JDK 1.2 version.
- It is not child interface to Collection Interface.
- It able to arrange all the elements in the form of Key-value pairs.
- In Map, both keys and values are objects.
- Duplicates are not allowed at keys, but values may be duplicated.
- Only one null value is allowed at keys side, but, any number of null values are allowed at values side.
- Both keys and Values are able to allow heterogeneous elements.
- Insertion order is not followed.
- Sorting order is not followed.

## Methods:

- **public void put(Object key, Object value)**

It will add the specified key-value pair to Map.

- **public void putAll(Map m)**

It will add all key-value pairs of the specified map to the present Map object.

- **public Object get(Object key)**

It will return value of the specified key.

- **public Object remove(Object key)**

It will remove a key-value pair from Map on the basis of the specified key.

- **public int size()**

It will return number of key-value pairs of a Map

- **public boolean containsKey(Object key)**

It will check whether the specified key is existed or not at keys side.

- **public boolean containsValue(Object key)**

It will check whether the specified value is available or not at values side.



- **public Set keySet()**

It will return all keys in the form of a Set.

- **public Collection values()**

It will return all values in the form of a Collection object.

- **public boolean isEmpty()**

It will check whether the Map object is empty or not, if the present map object is empty then it will return true value otherwise it will return false value.

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         HashMap hm=new HashMap();
7)         hm.put("A","AAA");
8)         hm.put("B","BBB");
9)         hm.put("C","CCC");
10)        hm.put("D","DDD");
11)        hm.put("E","EEE");
12)        System.out.println(hm);
13)        HashMap hm1=new HashMap();
14)        hm1.put("X","XXX");
15)        hm1.put("Y","YYY");
16)        System.out.println(hm1);
17)        hm.putAll(hm1);
18)        System.out.println(hm);
19)        System.out.println(hm.get("B"));
20)        System.out.println(hm.remove("E"));
21)        System.out.println(hm.size());
22)        System.out.println(hm.isEmpty());
23)        System.out.println(hm.containsKey("D"));
24)        System.out.println(hm.containsValue("DDD"));
25)        System.out.println(hm.keySet());
26)        System.out.println(hm.values());
27)    }
28} }
```



## HashMap:

- It was introduced in JDK1.2 version.
- It is not Legacy
- It is an implementation class to Map interface.
- It able to arrange all the elements in the form of Key-value pairs.
- In HashMap, both keys and values are objects.
- Duplicates are not allowed at keys, but values may be duplicated.
- Only one null value is allowed at keys side, but, any number of null values are allowed at values side.
- Both keys and Values are able to allow heterogeneous elements.
- Insertion order is not followed.
- Sorting order is not followed.
- Its internal data structure is "Hashtable".
- Its initial capacity is 16 elements.
- It is not synchronized
- No method is synchronized in HashMap
- It allows more than one thread to access data.
- It follows parallel execution.
- It will reduce application execution time.
- It will improve application performance.
- It is not giving guarantee for data consistency.
- It is not threadsafe.

## Constructors:

- public HashMap()
- public HashMap(int capacity)
- public HashMap(int capacity, float fill\_Ratio)
- public HashMap(Map m)

## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         HashMap hm=new HashMap();
7)         hm.put("A","AAA");
8)         hm.put("B", "BBB");
9)         hm.put("C", "CCC");
10)        hm.put("D", "DDD");
11)        hm.put("E", "EEE");
12)        System.out.println(hm);
13)        hm.put("B", "FFF");
14)        System.out.println(hm);
```



```
15) hm.put("F","CCC");
16) System.out.println(hm);
17) hm.put(null,"GGG");
18) hm.put(null,"HHH");
19) hm.put("G",null);
20) hm.put("H",null);
21) System.out.println(hm);
22) hm.put(new Integer(10), new Integer(20));
23) System.out.println(hm);
24) }
25) }
```

## LinkedHashMap:

### Q) What are the differences between *HashMap* and *LinkedHashMap*?

- *HashMap* was introduced in JDK 1.2 version.  
*LinkedHashMap* was introduced in JDK 1.4 version.
- *HashMap* is not following insertion order.  
*LinkedHashMap* is following insertion order.
- *HashMap* internal data structure is *Hashtable*.  
*LinkedHashMap* internal data structure is *Hashtable+LinkedList*

### EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         HashMap hm=new HashMap();
7)         hm.put("A","AAA");
8)         hm.put("B","BBB");
9)         hm.put("C","CCC");
10)        hm.put("D","DDD");
11)        hm.put("E","EEE");
12)        System.out.println(hm);
13)
14)        LinkedHashMap lhm=new LinkedHashMap();
15)        lhm.put("A","AAA");
16)        lhm.put("B","BBB");
17)        lhm.put("C","CCC");
```



```
18)    ihm.put("D","DDD");
19)    ihm.put("E","EEE");
20)    System.out.println(ihm);
21) }
22} }
```

## IdentityHashMap:

### Q) What are the differences between *HashMap* and *IdentityHashMap*?

- *HashMap* was introduced in JDK 1.2 version.  
*IdentityHashMap* was introduced in JDK 1.4 version.
- *HashMap* and *IdentityHashMap* are not allowing duplicate keys, to check duplicate keys *HashMap* will use *equals()* method, but, *IdentityHashMap* will use '*==*' operator.

**Note:** '*==*' operator will perform references comparison always, but, *equals()* method was defined in *Object* class initially, later on it was overridden in *String* class and in all wrapper classes in order to perform contents comparison.

### EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         Integer in1=new Integer(10);
7)         Integer in2=new Integer(10);
8)         HashMap hm=new HashMap();
9)         hm.put(in1,"AAA");
10)        hm.put(in2,"BBB");// in2.equals(in1)-->true
11)        System.out.println(hm);// {10=BBB}
12)
13)        IdentityHashMap ihm=new IdentityHashMap();
14)        ihm.put(in1, "AAA");
15)        ihm.put(in2, "BBB");// in2 == in1 --> false
16)        System.out.println(ihm);// {10=AAA, 10=BBB}
17)    }
18} }
```



## WeakHashMap:

### Q) What is the difference between *HashMap* and *WeakHashMap*?

- Once if we add an element to *HashMap* then *HashMap* is not allowing Garbage Collector to destroy its objects.
- Even if we add an element to *WeakHashMap* then *WeakHashMap* is able to allow Garbage Collector to destroy elements.

EX:

```
1) import java.util.*;
2) class A
3) {
4)     public String toString()
5)     {
6)         return "A";
7)     }
8) }
9) class Test
10) {
11)     public static void main(String[] args)
12)     {
13)         A a1=new A();
14)         HashMap hm=new HashMap();
15)         hm.put(a1, "AAA");
16)         System.out.println("HM Before GC :" +hm); // {A=AAA}
17)         a1=null;
18)         System.gc();
19)         System.out.println("HM After GC :" +hm); // {A=AA}
20)
21)         A a2=new A();
22)         WeakHashMap whm=new WeakHashMap();
23)         whm.put(a2, "AAA");
24)         System.out.println("WHM Before GC :" +whm); // {A=AAA}
25)         a2=null;
26)         System.gc();
27)         System.out.println("WHM After GC :" +whm); // {}
```



**NOTE:** In Java applications, Garbage Collector will destroy objects internally. In java applications, it is possible to destroy objects explicitly by activating GarbageCollector, for this, we have to use the following two steps.

- Nullify the respective object reference.
- Access System.gc() method, it will access finalize() method internally just before destroying object.

**EX:**

```
1) class A {  
2)     A0 {  
3)         System.out.println("Object Creating");  
4)     }  
5)     public void finalize() {  
6)         System.out.println("Object Destroying");  
7)     }  
8) }  
9) class Test {  
10)    public static void main(String[] args) {  
11)        A a = new A();  
12)        a = null;  
13)        System.gc();  
14)    }  
15) }
```

## SortedMap:

- It was introduced in JDK1.2 version.
- It is a direct child interface to Map interface
- It able to allow elements in the form of Key-Value pairs, where both keys and values are objects.
- It will not allow duplicate elements at keys side, but, it able to allow duplicate elements at values side.
- It will not follow insertion order.
- It will follow sorting order.
- It will not allow null values at keys side. If we are trying to add null values at keys side then JVM will rise an exception like java.lang.NullPointerException.
- It will not allow heterogeneous elements at keys side, if we are trying add heterogeneous elements then JVM will rise an exception like java.lang.ClassCastException.
- It able to allow only comparable objects at keys side by default, if we are trying to add non comparable objects then JVM will rise an exception like java.lang.ClassCastException.
- If we want to add non comparable objects then we must use Comparator.



## Methods:

```
public Object firstKey()  
public Object lastKey()  
public SortedMap headMap(Object key)  
public SortedMap tailMap(Object key)  
public SortedMap subMap(Object obj1, Object obj2)
```

### EX:

```
1) import java.util.*;  
2) class Test  
3) {  
4)     public static void main(String[] args)  
5)     {  
6)         TreeMap tm=new TreeMap();  
7)         tm.put("B", "BBB");  
8)         tm.put("E", "EEE");  
9)         tm.put("D", "DDD");  
10)        tm.put("A", "AAA");  
11)        tm.put("F", "FFF");  
12)        tm.put("C", "CCC");  
13)        System.out.println(tm);  
14)        System.out.println(tm.firstKey());  
15)        System.out.println(tm.lastKey());  
16)        System.out.println(tm.headMap("D"));  
17)        System.out.println(tm.tailMap("D"));  
18)        System.out.println(tm.subMap("B", "E"));  
19)    }  
20) }
```

## NavigableMap:

It was introduced in JAVA6 version, it is a child interface to SortedMap and it has defined methods to provide navigations over the elements.

### Methods:

- public Object CeilingKey(Object obj)
- public Object higherKey(Object obj)
- public Object floorKey(Object obj)
- public Object lowerKey(Object obj)
- public NavigableMap descendingMap()
- public Map.Entry pollFirstEntry()
- public Map.Entry pollLastEntry()



## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         TreeMap tm=new TreeMap();
7)         tm.put("A", "AAA");
8)         tm.put("B", "BBB");
9)         tm.put("C", "CCC");
10)        tm.put("D", "DDD");
11)        tm.put("E", "EEE");
12)        tm.put("F", "FFF");
13)        System.out.println(tm);
14)        System.out.println(tm.descendingMap());
15)        System.out.println(tm.ceilingKey("D"));
16)        System.out.println(tm.higherKey("D"));
17)        System.out.println(tm.floorKey("D"));
18)        System.out.println(tm.lowerKey("D"));
19)        System.out.println(tm.pollFirstEntry());
20)        System.out.println(tm.pollLastEntry());
21)        System.out.println(tm);
22)    }
23)}
```

## TreeMap:

- It was introduced in JDK 1.2 version.
- It is not Legacy.
- It is an implementation class to Map, SortedMap and NavigableMap interfaces.
- It able to allow elements in the form of Key-Value pairs, where both keys and values are objects.
- It will not allow duplicate elements at keys side, But, it able to allow duplicate elements at values side.
- It will not follow insertion order.
- It will follow sorting order.
- It will not allow null values at keys side. If we are trying to add null values at keys side then JVM will rise an exception like java.lang.NullPointerException.
- It will not allow heterogeneous elements at keys side, if we are trying add heterogeneous elements then JVM will rise an exception like java.lang.ClassCastException.
- It able to allow only comparable objects at keys side by default, if we are trying to add non comparable objects then JVM will rise an exception like java.lang.ClassCastException.
- If we want to add non comparable objects then we must use Comparator.



- Its internal data Struter is "Red-Black Tree".
- It is not synchronized.
- No methods are synchronized in TreeMap.
- It allows more than one thread to access data.
- It will follow parallel execution.
- It will reduce execution time.
- It will improve application performance.
- It is not giving guarantee for Data Consistency.
- It is not threadsafe.

## Constructors:

- `public TreeMap()`
- `public TreeMap(Comparator c)`
- `public TreeMap(SortedMap sm)`
- `public TreeMap(Map m)`

## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         TreeMap tm=new TreeMap();
7)         tm.put("A", "AAA");
8)         tm.put("B", "BBB");
9)         tm.put("C", "CCC");
10)        tm.put("D", "DDD");
11)        System.out.println(tm);
12)        tm.put("B", "EEE");
13)        System.out.println(tm);
14)        tm.put("E", "CCC");
15)        System.out.println(tm);
16)        //tm.put(null, "EEE");-->NullPointerException
17)        tm.put("F",null);
18)        System.out.println(tm);
19)        //tm.put(new Integer(10), new Integer(20));-->CCE
20)        System.out.println(tm);
21)        tm.put("G", new Integer(20));
22)        System.out.println(tm);
23)        //tm.put(new StringBuffer("BBB"), "GGG");-->CCE
24)        tm.put("H", new StringBuffer("HHH"));
25)        System.out.println(tm);
26)    }
27} }
```



## Hashtable:

### **Q) What are the differences between *HashMap* and *Hashtable*?**

- **HashMap was introduced in JDK 1.2 version.**  
**Hashtable was introduced in JDK 1.0 version.**
- **HashMap is not Legacy.**  
**Hashtable is Legacy.**
- **In HashMap, one null value is allowed at keys side and any number of null values are allowed at values side.**  
**In case of Hashtable, null values are not allowed at both keys and values side.**
- **HashMap is not synchronized.**  
**Hashtable is synchronized.**
- **No method is synchronized in HashMap.**  
**Almost all the methods are synchronized in Hashtable**
- **HashMap allows more than one thread to access data.**  
**Hashtable allows only one thread at a time to access data.**
- **HashMap follows parallel execution.**  
**Hashtable follows sequential execution.**
- **HashMap will reduce execution time.**  
**Hashtable will increase execution time.**
- **HashMap will improve application performance.**  
**Hashtable will reduce application performance.**
- **HashMap will not give guarantee for data consistency.**  
**Hashtable will give guarantee for data consistency**
- **HashMap is not threadsafe.**  
**Hashtable is threadsafe.**



## EX:

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         HashMap hm=new HashMap();
7)         hm.put("A", "AAA");
8)         hm.put("B", "BBB");
9)         hm.put("C", "CCC");
10)        hm.put("D", "DDD");
11)        System.out.println(hm);
12)        hm.put(null, "EEE");
13)        hm.put(null, "FFF");
14)        hm.put("E",null);
15)        hm.put("F", null);
16)        System.out.println(hm);
17)        System.out.println();
18)        Hashtable ht=new Hashtable();
19)        ht.put("A", "AAA");
20)        ht.put("B", "BBB");
21)        ht.put("C", "CCC");
22)        ht.put("D", "DDD");
23)        System.out.println(ht);
24)        //ht.put(null, "EEE");-->NullPointerException
25)        //ht.put("E", null);-->NullPointerException
26)    }
27} }
```

## Properties:

In Java applications, if we have any data which we want to change frequently then we have to manage that type of data in a properties file, otherwise we have to perform recompilation on every modification.

The main purpose of properties files in java applications is,

- To manage labels of the GUI components in GUI appl.
- To manage locale respective messages in I18N Appl.
- To manage exception messages in Exception handling.
- To manage validation messages in Data validations.



**EX:**

### user.properties

uname=User Name

upwd=User password

uname.required=User Name is Required

upwd.required=User Password is required

exception.insufficientfunds=Funds are not sufficient in your Account.

In Java applications, to represent data of a particular properties file we have to use java.util.Properties class.

To get data from a particular properties file to Properties object we have to use the following steps.

- Create Properties file with the data in the form of Key-value pairs.
- Create Properties class object.
- Create FileInputStream to get data from properties file.
- Load data from FileInputStream to Properties object by using the following method.  
`public void load(FileInputStream fis)`
- Get data from Properties object by using the following method.  
`public String getProperty(String key)`

To send data from Properties object to properties file we have to use the following steps.

- Create Properties class object.

- Set data to Properties object by using the following method.

`public void setProperty(String key, String val)`

- Create FileOutputStream with the target file.

- Store Properties object data to FileOutputStream by using the following method.

`public void store(FileOutputStream fos, String des)`

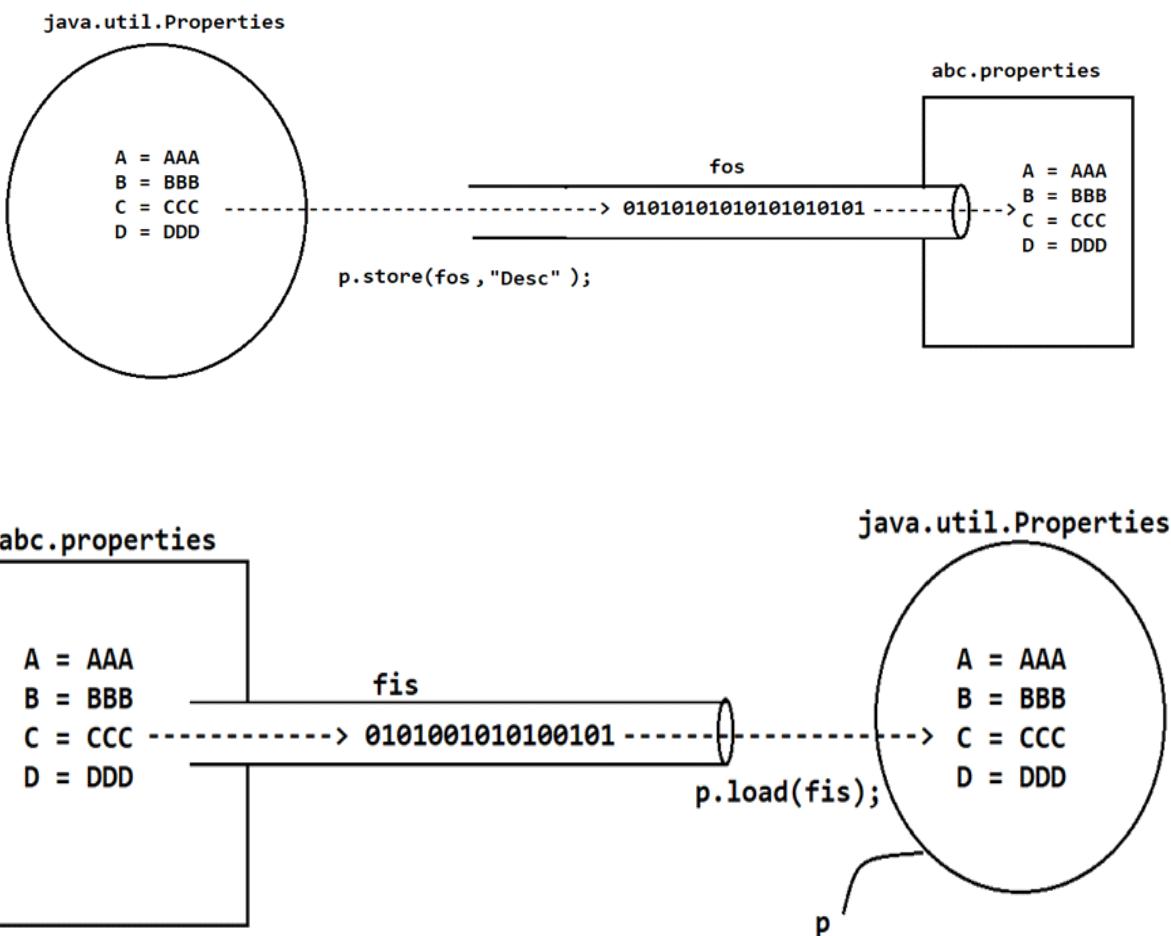
### db.properties

driver\_Class=sun.jdbc.odbc.JdbcOdbcDriver

driver\_URL=jdbc:odbc:dsn\_name

db\_User=system

db\_Password=durga



## Test.java

```
1) import java.util.*;
2) import java.io.*;
3) class Test
4) {
5)     public static void main(String[] args)throws Exception
6)     {
7)         Properties p=new Properties();
8)         FileInputStream fis=new FileInputStream("db.properties");
9)         p.load(fis);
10)        System.out.println("JDBC Parameters");
11)        System.out.println("-----");
12)        System.out.println("Driver_Class :" +p.getProperty("driver_Class"));
13)        System.out.println("Driver_URL   :" +p.getProperty("driver_URL"));
14)        System.out.println("DB_User      :" +p.getProperty("db_User"));
15)        System.out.println("DB_PAssword  :" +p.getProperty("db_Password"));
16)    }
17) }
```



## Test1.java

```
1) import java.util.*;
2) import java.io.*;
3) class Test1
4) {
5)     public static void main(String[] args) throws Exception
6)     {
7)         Properties p=new Properties();
8)         p.setProperty("uname", "Durga");
9)         p.setProperty("upwd", "durga123");
10)        p.setProperty("uqual", "M Tech");
11)        p.setProperty("uemail", "durga@durgasoftware.com");
12)        p.setProperty("umobile", "91-9988776655");
13)
14)        FileOutputStream fos=new FileOutputStream("user.properties");
15)        p.store(fos, "User Details");
16)    }
17} }
```



# Generics



## What is the Requirement to use Generics in Java Applications?

- To represent one thing if we allow only one data type then it is type safe operation.

**EX:** Arrays are providing type safe operation.

```
Student[] std = new Student[3];
std[0] = new Student();----> Valid
std[1] = new Student();----> Valid
std[2] = new Customer();----> Invalid
```

To represent one thing if we allow more than one type then it is Type unsafe operation.

**EX:** Collection objects are providing type Unsafe operation, because, Collection objects are able to allow heterogeneous elements.

```
ArrayList al = new ArrayList();
al.add("AAA");
al.add(new Employee());
al.add(new Integer(10));
al.add(new StringBuffer("CCC"));
```

In Collections to improve typedness or to provide type safe operations then we have to use "Generics".

- If we add elements to the Collection object and if we want to retrieve elements from Collection objects we must perform type casting, because, get(--) methods of Collections are able to retrieve elements in the form of java.lang.Object type.

**EX:**

```
ArrayList al = new ArrayList();
al.add("A");
al.add("B");
al.add("C");
al.add("D");
System.out.println(al);
String str = al.get(2);---> Compilation Error, Incompatible types.
String str = (String)al.get(2);--> Valid
```

If we use Generics along with Collections then it is not required to type cast while iterating elements.

The main intention of Generics is,

- To provide typesafe operations in Collections.
- To avoid Type casting while retrieving elements from Collections.



Generics is a Type parameter specified along with Collections in order to fix a particular type of elements to add.

### Syntax:

Collection\_Name<Generic\_Type> ref = new Collection\_Name<GenericType>([ Params]);

**EX:** `ArrayList<String> al = new ArrayList<String>();`

The above `ArrayList` object is able to add only `String` type elements, if we are trying to add any other elements then Compiler will rise an error.

**EX:**

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         ArrayList<String> al=new ArrayList<String>();
7)         al.add("A");
8)         al.add("B");
9)         al.add("C");
10)        al.add("D");
11)        al.add("E");
12)        System.out.println(al);
13)        al.add(new Integer(10));----> Error
14)        System.out.println(al);
15)    }
16} 
```

Before JDK 5.0 version, `ArrayList` class is

```
1) public class ArrayList
2) {
3)     public void add(Object obj)
4)     {
5)     }
6)     public Object get(int index)
7)     {
8)     }
9)     ----
10)    ----
11} 
```



---

From JDK 5.0 version ArrayList class is

```
1) public class ArrayList<T>
2) {
3)     public void add(T t)
4)     {
5)     }
6)     public T get(int index)
7)     {
8)     }
9) }
```

In the above ArrayList we can provide any user defined data type to T.

**EX 1:** `ArrayList<String> al = new ArrayList<String>();`

Here ArrayList class is converted internally as

```
1) public class ArrayList<String>
2) {
3)     public void add(String t)
4)     {
5)     }
6)     public String get(int index)
7)     {
8)     }
9) }
```

Here add() method is able to take only String elements to add to the ArrayList and get() method will return only String elements.

```
al.add("A");---> Valid  
al.add("B");---> Valid  
al.add("C");---> Valid  
al.add(new Integer(10));---> Invalid.  
String str = al.get(2);
```

**EX 2:** `ArrayList<Integer> al = new ArrayList<Integer>();`

Here ArrayList class is converted internally like below.

```
1) public class ArrayList<Integer>
2) {
3)     public void add(Integer t)
4)     {
5)     }
6)     public Integer get(int index)
```



7) {  
8) }  
9) }

Here add() method is able to take only Integer elements to add to the ArrayList and get() method will return only Integer elements.

```
al.add(new Integer(10));--> Valid  
al.add(new Integer(20));--> Valid  
al.add(30);--> Valid, Autoboxing  
al.add("AAA");--> Invalid.  
Integer in=al.get(1);--> valid  
int i=al.get(2);--> Valid, Auto-Unboxing
```

If any java class is having Generic type parameter at its declaration then that class is called as "Generic Class".

**NOTE:** "Generics" feature is provided by JAVA along with its JDK5.0 version.

**EX:**

```
class Account<T>  
{  
}
```

```
Account<Savings> acc=new Account<Savings>();  
Account<Current> acc=new Account<Current>();
```

In the above Generic class declaration, we can use any valid java identifier as Type parameter.

**EX:**

```
class Account<X>  
{  
}
```

Status: Valid

**EX:**

```
class Account<Durga>  
{  
}
```

Status: Valid



---

In the above Generic class declaration, we can use any no of Type parameters by provide ',' separator.

EX:

```
class HashMap<K,V>
{
}
```

Where 'K' is key.

Where 'V' is value.

EX:

```
1) class Account<T>
2) {
3)     T obj;
4)     public void set(T obj)
5)     {
6)         this.obj=obj;
7)     }
8)     public T get()
9)     {
10)        return obj;
11)    }
12)    public void display_Type()
13)    {
14)        System.out.println(obj.getClass().getName());
15)    }
16) }
17) class Test
18) {
19)     public static void main(String[] args)
20)     {
21)         Account<String> acc=new Account<String>();
22)         acc.set("Savings_Account");
23)         System.out.println(acc.get());
24)         acc.display_Type();
25)     }
26) }
```



## Bounded Types:

In Generic Classes, we can bound the type parameter for a particular range by using "extends" keyword.

### EX 1:

```
class Test<T>
{
}
```

It is unbounded type, we can pass any type of parameter.

```
Test<String> t=new Test<String>();
Test<Integer> t=new Test<Integer>();
```

### EX 2:

```
class Test<T extends X>
{
}
```

If X is a class type then we can pass either X type elements or sub class elements of X type as parameter types.

### EX:

```
class Payment
{
}
class CashPayment extends Payment
{
}
class CardPayment extends Payment
{
}
class Bill<T extends Payment>
{
}
```

```
Bill<Payment> bill = new Bill<Payment>();--> Valid
```

```
Bill<CashPayment> bill = new Bill<CashPayment>();--> Valid
```

```
Bill<CardPayment> bill = new Bill<CardPayment>();--> Valid
```

### EX:

```
class Test<T extends Number>
{
}
Test<Number> t=new Test<Number>();--> Valid
```



---

```
Test<Integer> t=new Test<Integer>();----> Valid  
Test<Float> t=new Test<Float>();----> Valid  
Test<String> t=new Test<String>();---> Invalid
```

If 'X' is an interface then we are ABle to pass either X type elements of its implementation class types as parameter.

**EX:**

```
interface Java  
{  
}  
class CoreJava implements Java  
{  
}  
class AdvJava implements Java  
{  
}  
class Course<T extends Java>  
{  
}
```

```
Course<Java> crs1 = new Course<Java>(); --->Valid  
Course<CoreJava> crs2 = new Course<CoreJava>();--> Valid  
Course<AdvJava> crs3 = new Course<AdvJava>();---> Valid
```

**NOTE:** Bouded parameters are not allowing "implements" keyword and "super" keyword.

```
class Test<T implements Runnable>  
{  
}
```

Status: Invalid

**EX:**

```
class Test<T super String>  
{  
}
```

Status: Invalid

In Generic classes, we can use more than one type as bounded parameter by using '&' symbol.



**EX:**

```
class Test<T extends Number & Serializable>
{
}
```

Here Test class is able to allow the elements which must be either same as Number or sub classes to the number and which must be the implementations of Serializable interface.

```
Test<Integer> t=new Test<Integer>();--> Valid
Test<Float> t=new Test<Float>();--> Valid
Test<String> t=new Test<String>();--> Invalid.
```

**EX:**

```
class Test<T extends Number & Thread>
{
}
```

Status: Invalid

Reason: extends keyword will not allow two class types at a time.

**EX:**

```
class Test<T extends Runnable & Serializable>
{
}
```

Status: valid.

Reason: extends keyword is able to allow more than one interface type.

**EX:**

```
class Test<T extends Number & Runnable>
{
}
```

Status: Valid.

Reason: Extends is able to allow both class type and interface type , but, first we have to specify class type then we ahve to specify Interface type.

**EX:**

```
class Test<T extends Runnable & Number>
{
}
```

Status: Invalid.

Reason: extends keyword allows first class\_Name then interface\_Name



EX:

```
1) import java.util.*;
2) class Account
3) {
4)     String accNo;
5)     String accName;
6)     String accType;
7)
8)     Account(String accNo, String accName, String accType)
9)     {
10)         this.accNo=accNo;
11)         this.accName=accName;
12)         this.accType=accType;
13)     }
14)
15) }
16) class Bank
17) {
18)     public ArrayList<Account> getAccountsList(ArrayList<Account> al)
19)     {
20)         al.add(new Account("a111", "AAA", "Savings"));
21)         al.add(new Account("b111", "BBB", "Savings"));
22)         al.add(new Account("c111", "CCC", "Savings"));
23)         al.add(new Account("d111", "DDD", "Savings"));
24)         return al;
25)     }
26) }
27) class Test {
28)     public static void main(String[] args)
29)     {
30)         ArrayList<Account> al=new ArrayList<Account>();
31)         Bank bank=new Bank();
32)         ArrayList<Account> acc_List=bank.getAccountsList(al);
33)         System.out.println("ACCNO\tACCNAME\tACCTYPE");
34)         System.out.println("-----");
35)         for(Account acc: acc_List)
36)         {
37)             System.out.println(acc.accNo+"\t"+acc.accName+"\t"+acc.accType);
38)         }
39)     }
40} }
```



# GUI

## (Graphical User Interface)

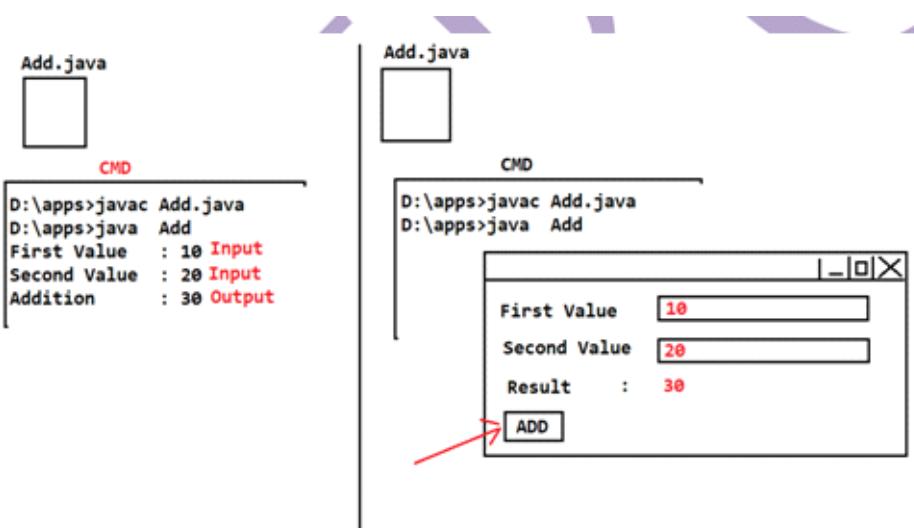


By using Java programming language, we are able to prepare the following two types of applications.

- CUI Applications
- GUI Applications

## Q) What is the difference between CUI Application and GUI Application?

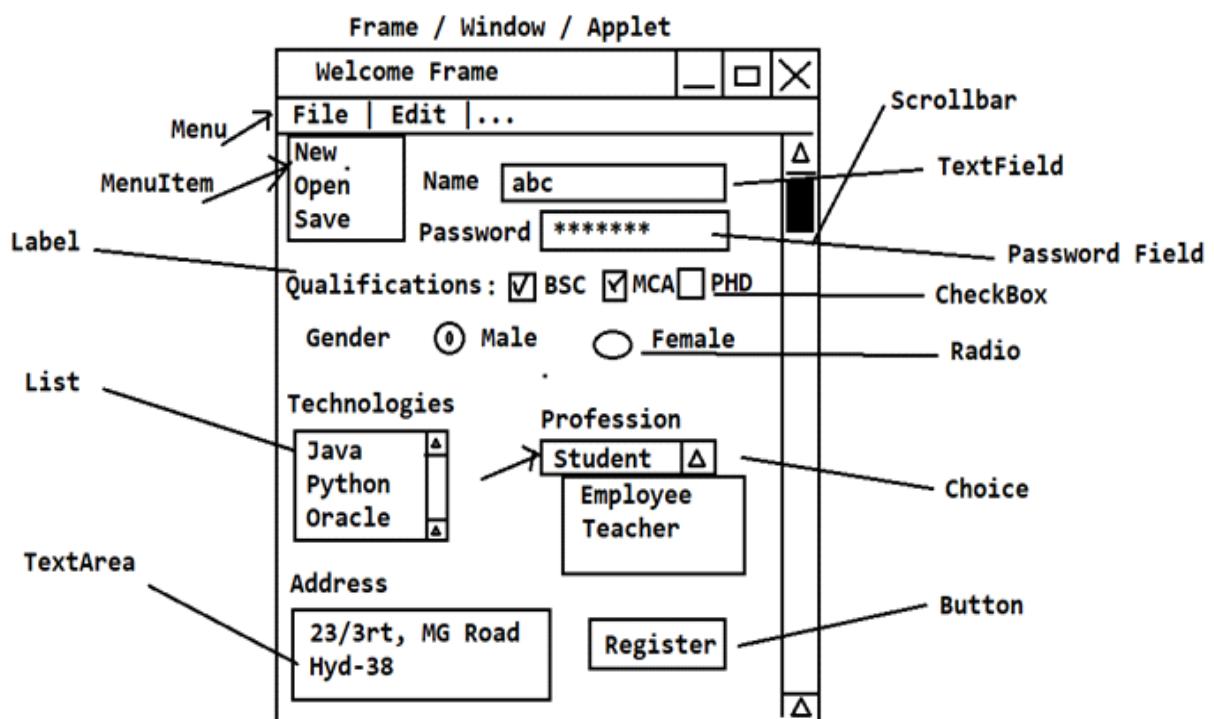
- CUI Application is a java application, it would be designed in such a way that it take input from command prompt and to provide output on the same command prompt, where command prompt is acting as an user interface and it is supporting characters data.
- GUI Application is a java application, it would be designed in such a way that to take input from the collection of Graphics components and to provide output on the same collection of Graphics components, where the collection of graphics components is acting an user interface and it is supporting graphics data.
- To prepare GUI Applications, JAVA has provided the complete predefined library in the form of the following three packages.
  - `java.awt`
  - `java.applet`
  - `javax.swing`



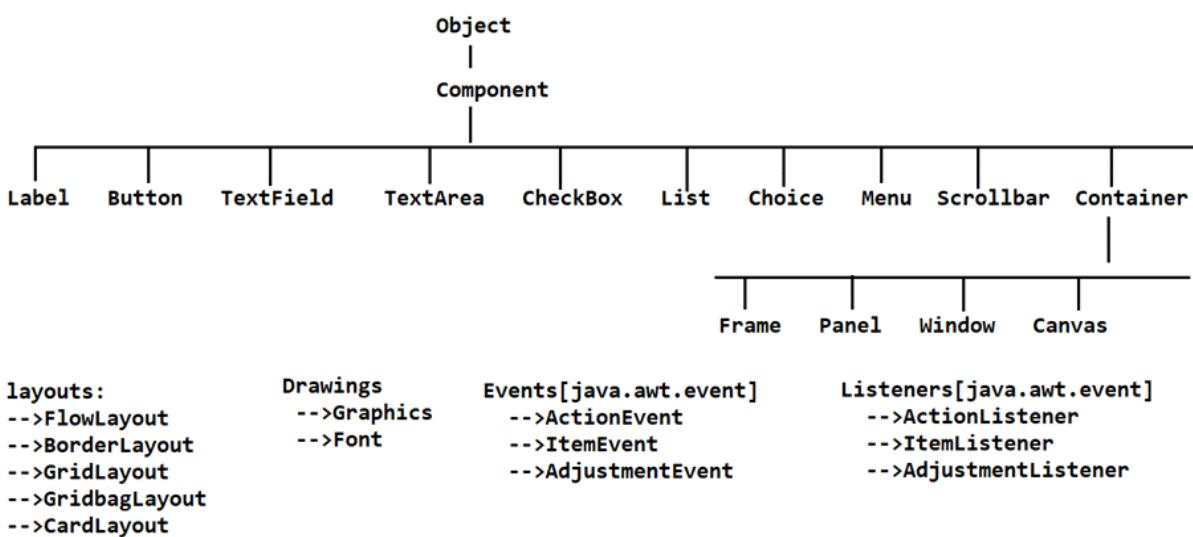


## AWT [Abstract Windowing Toolkit]

- AWT is a toolkit or collection of predefined classes and interfaces to design GUI Applications.
- AWT is a framework, it can be used to prepare Desktop applications.
- AWT is an API, it can be used to prepare Window based applications.
- To prepare GUI applications, AWT is able to provide support for the following GUI components.



To represent the above GUI components, `java.awt` package has provided the following predefined classes.



## Frame:

- Frame is a container, it able to manage some other GUI components.
- To represent Frame in GUI applications, JAVA has provided a separate predefined class in the form of `java.awt.Frame`.
- To create Frame class object we have to use the following constructor from Frame class.

### public Frame()

It will create a Frame without title.

### public Frame(String title)

It will create a Frame with the specified title.

In GUI applications, when we create Frame then Frame will be created with invisible mode, where to give visibility to the Frame we will use the following methods.

### public void show()

It able to show the frame, it unable to hide frame and this method is deprecated method.

### public void setVisible(boolean b)

If b is true then it will provide visibility to the Frame, if b is false then it will hide Frame.

In GUI applications, when we create Frame then Frame will be created with 0 width and 0 height, where to provide size to the Frame explicitly we have to use the following method.  
`public void setSize(int width, int height)`



To set a particular title to the Frame explicitly we have to use the following method.  
`public void setTitle(String title)`

To set a particular background color to the frame we have to use the following method.  
`public void setBackground(Color clr)`

EX:

```
1) import java.awt.*;
2) public class FrameEx
3) {
4)     public static void main(String[] args)
5)     {
6)         Frame f=new Frame();
7)         //f.show();
8)         f.setVisible(true);
9)         f.setSize(500,500);
10)        f.setTitle("Frame Demo");
11)        f.setBackground(Color.green);
12)    }
13) }
```

In GUI applications, above approach is not suggestible to create Frames, because, we are unable to perform customizations over the Frame. To implement customizations over the Frame we have to create user defined Frames.

To create User defined frames we have to use the following steps.

- Declare an user defined class.
- Extend User defined class from `java.awt.Frame` class.
- Declare a 0-arg constructor in user defined class.
- Provide all Frame properties in User defined class constructor.
- In main class, in `main()` method, create object for user defined Frame class.

EX:

```
1) import java.awt.*;
2) class MyFrame extends Frame
3) {
4)     MyFrame()
5)     {
6)         this.setVisible(true);
7)         this.setSize(500,500);
8)         this.setTitle("Custom Frame Example");
9)         this.setBackground(Color.red);
10)    }
11) }
12) class CustomFrameEx
```



```
13) {
14)     public static void main(String[] args)
15)     {
16)         MyFrame mf=new MyFrame();
17)     }
18) }
```

If we want to display textual data on frame then we have to override paint(-) method provided by Frame class.

```
public void paint(Graphics g)
```

To define Font properties , java has provided a predefined class in the form of `java.awt.Font`, to craete object for Font class we have to use the following constructor. `public Font(String font_Type, int font_Style, int font_Size)`.

To set foreground color to the Frame we have to use the following method from `java.awt.Frame` class.

```
public void setForeground(Color clr)
```

To set Font properties to the Graphics object we have to use the following method.

```
public void setFont(Font f)
```

**NOTE:** When we create user defined Frame class object, JVM has to access super class 0-arg constructor, that is, Frame class 0-arg constructor, where in Frame class 0-arg constructor JVM will access repaint(-) method, where repaint() method will access paint() method.

**EX:**

```
1) import java.awt.*;
2) class LogoFrame extends Frame
3) {
4)     LogoFrame()
5)     {
6)         this.setVisible(true);
7)         this.setSize(700,400);
8)         this.setTitle("Paint Example");
9)         this.setBackground(Color.green);
10)    }
11)    public void paint(Graphics g)
12)    {
13)        Font f=new Font("arial", Font.BOLD+Font.ITALIC, 35);
14)        g.setFont(f);
15)        this.setForeground(Color.red);
16)        g.drawString("DURGA SOFTWARE SOLUTIONS", 100,100);
17)    }
```



```
18)
19) class PaintEx
20) {
21)     public static void main(String[] args)
22)     {
23)         LogoFrame lf=new LogoFrame();
24)     }
25) }
```

## Event Handling / Event Delegation Model:

In GUI applications, we may use the GUI components like Buttons, Check boxes, radio Buttons,.... When we click on button or select item in check box,.. the respective GUI components are able to rise the respective events, here GUI components are not capable to handle the generated events. In this context, to handle the generated events we have to use another component internally that is Listener.

In GUI applications, Listener will take events from GUI components, handle them, perform the required actions by executing listener methods and return the result back to GUI application. Here, the process delegating events from GUI components to Listeners in order to handle is called as "Event Delegation Model" or Event Handling.

To implement Event handling in GUI applications, JAVA has provided the following predefined library as part of  
"java.awt.event" package.

**Procedure to implement Event Handling in GUI Applications:**

- Declare container class as sub class to java.awt.Frame class.
- Declare a 0-arg constructor in container class.
- Create the required GUI component in container class constructor.
- Select Listener on the basis of GUI Component.
- Declare an implementation class for Listener interface.
- Provide implementation for all Listener interface methods in Listener implementation class as per the requirement.
- Add Listener to the respective GUI Component by using the following method.  
`public void addXXXListener(XXXListener l)`

Where XXXListener may be ActionListener, ItemListener,...

**NOTE:** In GUI Applications, in general, we will take container class as implementation class for Listener interface, in the same container class Listener interface methods must be implemented.



EX:

```
1) class MyFrame extends Frame// Container class
2) {
3)     MyFrame()// Constructor
4)     {
5)         Button b=new Button("Submit");// GUI Comp.
6)         b.addActionListener(new MyActionListenerImpl());
7)     } // Adding Listener to GUI Component
8)
9)
10) class MyActionListenerImpl implements ActionListener
11) { // Listener Implementation class and Listener methods impl
12)     public void actionPerformed(ActionEvent ae)
13)     {
14)         ----implementation-----
15)     }
16) }
```

EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) class WindowListenerImpl implements WindowListener
4) {
5)     public void windowOpened(WindowEvent we)
6)     {
7)         System.out.println("WindowOpened");
8)     }
9)     public void windowClosed(WindowEvent we)
10)    {
11)        System.out.println("WindowClosed");
12)    }
13)     public void windowClosing(WindowEvent we)
14)    {
15)        System.out.println("WindowClosing");
16)        System.exit(0);
17)    }
18)     public void windowIconified(WindowEvent we)
19)    {
20)        System.out.println("WindowIconified");
21)    }
22)     public void windowDeiconified(WindowEvent we)
23)    {
24)        System.out.println("windowDeiconified");
25)    }
26)     public void windowActivated(WindowEvent we)
```



```
27) {
28)     System.out.println("WindowActivated");
29) }
30) public void windowDeactivated(WindowEvent we)
31) {
32)     System.out.println("WindowDeactivated");
33) }
34) }
35) class MyFrame extends Frame
36) {
37)     MyFrame()
38) {
39)         this.setVisible(true);
40)         this.setSize(500,500);
41)         this.setTitle("Window Events Example");
42)         this.setBackground(Color.green);
43)         this.addWindowListener(new WindowListenerImpl());
44)     }
45) }
46) class WindowEventsEx
47) {
48)     public static void main(String[] args)
49)     {
50)         MyFrame mf=new MyFrame();
51)     }
52) }
```

In Window Events, if we want to provide only window closing option to the frame then we have to use only `windowClosing()` method in the above implementation. To provide only `windowClosing()` method if we implement `WindowListener` interface then we must provide implementation for all the methods of `WindowListener` interface, this approach will increase unnecessary methods implementations in GUI Applications.

To overcome the above problem, to avoid unnecessary methods implementations AWT has provided an adapter class in the form of "`java.awt.event.WindowAdapter` abstract class.

`java.awt.event.WindowAdapter` abstract class is a direct implementation abstract class to `WindowListener` interface and it has provided empty implementation [ {} ] for all the methods of `WindowListener` interface.

If we want to use `WindowAdapter` abstract class in GUI Applications then we have to declare an user defined class and it must extend `java.awt.event.WindowAdapter` abstract class, where we have to override the required method.



EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) class WindowListenerImpl extends WindowAdapter
4) {
5)     public void windowClosing(WindowEvent we)
6)     {
7)         System.exit(0);
8)     }
9) }
10) class MyFrame extends Frame
11) {
12)     MyFrame()
13)     {
14)         this.setVisible(true);
15)         this.setSize(500,500);
16)         this.setTitle("Window Events Example");
17)         this.setBackground(Color.green);
18)         this.addWindowListener(new WindowListenerImpl());
19)     }
20) }
21) class WindowEventsEx
22) {
23)     public static void main(String[] args)
24)     {
25)         MyFrame mf=new MyFrame();
26)     }
27) }
```

To provide window closing option to the frame if we use the above approach then we have to provide a separate class for implementing windowClosing(-) method, it will increase no of classes in GUI applications, here to remove this type of unnecessary classes we have to use anonymous inner class of WindowAdapter abstract class as parameter to addWindowListener(-) method directly.

EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) class MyFrame extends Frame
4) {
5)     MyFrame()
6)     {
7)         this.setVisible(true);
8)         this.setSize(500,500);
9)         this.setTitle("Window Events Example");
```



```
10)     this.setBackground(Color.green);
11)     this.addWindowListener(new WindowAdapter()
12)     {
13)         public void windowClosing(WindowEvent we)
14)         {
15)             System.exit(0);
16)         }
17)     });
18) }
19)
20) class WindowEventsEx
21) {
22)     public static void main(String[] args)
23)     {
24)         MyFrame mf=new MyFrame();
25)     }
26} }
```

EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) class MouseListenerImpl implements MouseListener
4) {
5)     public void mousePressed(MouseEvent me)
6)     {
7)         System.out.println("Mouse Pressed["+me.getX()+","+me.getY()+"]");
8)     }
9)     public void mouseReleased(MouseEvent me)
10)    {
11)        System.out.println("Mouse Released["+me.getX()+","+me.getY()+"]");
12)    }
13)     public void mouseClicked(MouseEvent me)
14)    {
15)        System.out.println("Mouse Clicked["+me.getX()+","+me.getY()+"]");
16)    }
17)     public void mouseEntered(MouseEvent me)
18)    {
19)        System.out.println("Mouse Entered["+me.getX()+","+me.getY()+"]");
20)    }
21)     public void mouseExited(MouseEvent me)
22)    {
23)        System.out.println("Mouse Exited["+me.getX()+","+me.getY()+"]");
24)    }
25) }
26) class MyFrame extends Frame
```



```
27)
28) MyFrame()
29) {
30)     this.setVisible(true);
31)     this.setSize(500,500);
32)     this.setTitle("Mouse Events Example");
33)     this.setBackground(Color.yellow);
34)     this.addWindowListener(new WindowAdapter()
35)     {
36)         public void windowClosing(WindowEvent we)
37)         {
38)             System.exit(0);
39)         }
40)     });
41)
42)     this.addMouseListener(new MouseListenerImpl());
43) }
44) }
45) class MouseEventsEx
46) {
47)     public static void main(String[] args)
48)     {
49)         MyFrame mf=new MyFrame();
50)     }
51) }
```

EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) class KeyListenerImpl implements KeyListener
4) {
5)     public void keyPressed(KeyEvent ke)
6)     {
7)         System.out.println("KeyPressed["+ke.getKeyChar()+"]");
8)     }
9)     public void keyTyped(KeyEvent ke)
10)    {
11)        System.out.println("KeyTyped["+ke.getKeyChar()+"]");
12)    }
13)     public void keyReleased(KeyEvent ke)
14)    {
15)        System.out.println("KeyReleased["+ke.getKeyChar()+"]");
16)    }
17) }
18) class MyFrame extends Frame
```



```
19)
20) MyFrame()
21) {
22)     this.setVisible(true);
23)     this.setSize(500,500);
24)     this.setTitle("Key Events Example");
25)     this.setBackground(Color.green);
26)     this.addWindowListener(new WindowAdapter()
27)     {
28)         public void windowClosing(WindowEvent we)
29)         {
30)             System.exit(0);
31)         }
32)     });
33)     this.addKeyListener(new KeyListenerImpl());
34) }
35)
36) class KeyEventsEx
37) {
38)     public static void main(String[] args)
39)     {
40)         MyFrame mf=new MyFrame();
41)     }
42} }
```

## Layout Mechanisms:

The main purpose of Layout mechanism is to arrange all the GUI components in container in a particular order.

There are five layout mechanisms in AWT.

- FlowLayout
- BorderLayout
- GridLayout
- GridbagLayout
- CardLayout

To set a particular Layout mechanism to the Container we have to use the following method.

public void setLayout(XXXLayout l)

Where XXXLayout may be FlowLayout, BorderLayout,.....



- **FlowLayout:**

- It able to arrange all the GUI components in rows manner.
- To represent this Layout mechanism AWT has provided a predefined class in the form of `java.awt.FlowLayout`.
- To set flow layout to the Frame we have to use the following instruction.  
`f.setLayout(new FlowLayout());`

- **BorderLayout:**

- It able to arrange GUI components along with borders of the containers.
- To represent this Layout mechanism AWT has provided a predefined class in the form of `java.awt.BorderLayout`.
- To set this layout mechanism, we have to use the following instruction.  
`f.setLayout(new BorderLayout());`
- To represents the locations in container, `BorderLayout` class has provided the constants like EAST, NORTH, SOUTH, WEST, CENTER.
- In case of Border layout, if we want to add the elements to the container we have to use the following method.  
`public void add(int location, Component c)`  
EX: `f.add(BorderLayout.CENTER, new WelcomePanel());`

- **GridLayout:**

- It able to arrange all the GUI components in rows and columns wise in the form of grids.
- To represent this layout mechanism, AWT has provided a predefined class in the form of `java.awt.GridLayout`.
- To set Grid layout mechanism we have to use the following instruction.  
`f.setLayout(new GridLayout(4,4));`  
Where 4, 4 is no of rows and no columns.

If we want to use `GridLayout` in GUI applications then we must follow the following rules.

- All the GUI components must have equals size.
- We must not leave any empty grid in between GUI components.

- **GridbagLayout:**

- This layout mechanism is almost all same as `GridLayout`, but, it able to allow empty grids between the components and it able to allow the GUI components having variable sizes.
- To represent this layout mechanism, AWT has provided a predefined class in the form of `java.awt.GridbagLayout`.
- To use this layout mechanism we have to use the following instruction.  
`f.setLayout(new GridbagLayout(3,3));`



## **Card Layout:**

- It able to arrange all the elements in the form of cards which are available in overlapped manner.
- To represent this layout , AWT has provided a predefined class in the form of `java.awt.CardLayout`.
- To use this layout mechanism we have to use the following code.  
`f.setLayout(new CardLayout(4));`  
Where 4 is no of cards.

## **Label:**

- It is an output GUI component, it able to display a line of data on Frame.
- To represent Label GUI component AWT has provided a predefined class in the form of `java.awt.Label`.
- To create Object for Label class we have to use the following constructors.
  - `public Label()`
  - `public Label(String label)`
  - `public Label(String label, int alignment)`Where alignment may be the constants like LEFT, RIGHT, CENTER from Label class.

## **Button:**

- It is an output GUI component, it able to generate ActionEvent in order to perform a particular Action.
- To represent this GUI component AWT has provided a predefined class in the form of "`java.awt.Button`".
- To create Button class objects we have to use the following constructors.
  - `public Button()`
  - `public Button(String label)`
- To set label to the button and to get label from Button we have to use the following methods.
  - `public void setLabel(String label)`
  - `public String getLabel()`
- To get clicked button label among the multiple buttons we have to use the following method.
  - `public String getActionCommand()`



EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) class ColorsFrame extends Frame implements ActionListener
4) {
5)     Button b1, b2, b3;
6)     ColorsFrame()
7)     {
8)         this.setVisible(true);
9)         this.setSize(500,500);
10)        this.setTitle("Colors Frame");
11)        this.setLayout(new FlowLayout());
12)        this.addWindowListener(new WindowAdapter()
13)        {
14)            public void windowClosing(WindowEvent we)
15)            {
16)                System.exit(0);
17)            }
18)        });
19)
20)        b1=new Button("Red");
21)        b2=new Button("Green");
22)        b3=new Button("Blue");
23)
24)        b1.addActionListener(this);
25)        b2.addActionListener(this);
26)        b3.addActionListener(this);
27)
28)        b1.setBackground(Color.red);
29)        b2.setBackground(Color.green);
30)        b3.setBackground(Color.blue);
31)
32)        this.add(b1);
33)        this.add(b2);
34)        this.add(b3);
35)
36)    }
37)    public void actionPerformed(ActionEvent ae)
38)    {
39)        String label=ae.getActionCommand();
40)        if(label.equals("Red"))
41)        {
42)            this.setBackground(Color.red);
43)        }
44)        if(label.equals("Green"))
```



```
45)  {
46)      this.setBackground(Color.green);
47)  }
48)  if(label.equals("Blue"))
49)  {
50)      this.setBackground(Color.blue);
51)  }
52) }
53)
54) class ButtonEx
55) {
56)     public static void main(String[] args)
57)     {
58)         ColorsFrame cf=new ColorsFrame();
59)     }
60} }
```

## TextField:

- It is an input GUI component, it able to take a single line of data from User.
- To represent this GUI component, AWT has provided a predefined class in the form of "java.awt.TextField".
- To create TextField class object we have to use the following constructors.
  - `public TextField()`
  - `public TextField(int size)`
  - `public TextField(String def_msg, int size)`
- To set data to the TextField and to get data from Text field we have to use the following methods.
  - `public void setText(String text)`
  - `public String getText()`

## TextArea:

- It is an input GUI component, it able to take multiple lines of data from user.
- To represent this GUI component, AWT has provided a predefined class in the form of "java.awt.TextArea".
- To create TextArea class object AWT has provided the following constructors.
  - `public TextArea()`
  - `public TextArea(int rows, int columns)`
  - `public TextArea(String def_Message, int rows, int cols)`



- To set data to Text Area and to get data from Text Area we have to use the following methods.
  - `public void setText(String text)`
  - `public String getText()`

**NOTE:** If we want to prepare Password field in GUI applications then we have to take a text filed, where we have to hide data with a particular character called as Echo Character, where to set a particular Echo Character we have to use the following method.

```
public void setEchoChar(char c)
```

**EX:**

```
1) import java.awt.*;
2) import java.awt.event.*;
3) class UserFrame extends Frame implements ActionListener
4) {
5)     Label l1, l2, l3, l4, l5;
6)     TextField tf1, tf2, tf3, tf4;
7)     TextArea ta;
8)     Button b;
9)
10)    String uname="";
11)    String upwd="";
12)    String uemail="";
13)    String umobile="";
14)    String uaddr="";
15)
16)    UserFrame()
17)    {
18)        this.setVisible(true);
19)        this.setSize(500,500);
20)        this.setTitle("User Frame");
21)        this.setBackground(Color.green);
22)        this.setLayout(new FlowLayout());
23)        this.addWindowListener(new WindowAdapter()
24)        {
25)            public void windowClosing(WindowEvent we)
26)            {
27)                System.exit(0);
28)            }
29)        });
30)
31)    l1=new Label("User Name");
32)    l2=new Label("Password");
33)    l3=new Label("User Email");
```



```
34)    l4=new Label("User Mobile");
35)    l5=new Label("User Address");
36)
37)    tf1=new TextField(20);
38)    tf2=new TextField(20);
39)    tf2.setEchoChar('*');
40)    tf3=new TextField(20);
41)    tf4=new TextField(20);
42)
43)    ta=new TextArea(3,15);
44)    b=new Button("Registration");
45)    b.addActionListener(this);
46)
47)    Font f=new Font("arial", Font.BOLD, 15);
48)    l1.setFont(f);
49)    l2.setFont(f);
50)    l3.setFont(f);
51)    l4.setFont(f);
52)    l5.setFont(f);
53)    tf1.setFont(f);
54)    tf2.setFont(f);
55)    tf3.setFont(f);
56)    tf4.setFont(f);
57)    ta.setFont(f);
58)    b.setFont(f);
59)
60)    this.add(l1); this.add(tf1);
61)    this.add(l2); this.add(tf2);
62)    this.add(l3); this.add(tf3);
63)    this.add(l4); this.add(tf4);
64)    this.add(l5); this.add(ta);
65)    this.add(b);
66)
67} }
68) public void actionPerformed(ActionEvent ae)
69) {
70)    uname=tf1.getText();
71)    upwd=tf2.getText();
72)    uemail=tf3.getText();
73)    umobile=tf4.getText();
74)    uaddr=ta.getText();
75)    repaint();
76} }
77) public void paint(Graphics g)
78) {
```



```
79)     Font f=new Font("arial", Font.BOLD, 20);
80)     g.setFont(f);
81)     g.drawString("User Name :" +uname,50,250);
82)     g.drawString("Password :" +upwd,50,300);
83)     g.drawString("User Email :" +uemail,50,350);
84)     g.drawString("User Mobile :" +umobile,50,400);
85)     g.drawString("User Address :" +uaddr,50,450);
86) }
87} }
88) class TextFieldEx
89) {
90)     public static void main(String[] args)
91)     {
92)         UserFrame uf=new UserFrame();
93)     }
94) }
```

## Checkbox:

- It is an input GUI component; it can be used to select an item represented by the present Checkbox.
- To represent this GUI component AWT has provided a predefined class in the form of "java.awt.Checkbox".
- To create Checkbox class objects we have to use the following constructors
  - `public Checkbox()`
  - `public Checkbox(String label)`
  - `public Checkbox(String label, boolean state)`
  - `public Checkbox(String label, CheckboxGroup cg, boolean state)`
- To set label to the Checkbox and to get label from Check box we have to use the following methods.
  - `public void setLabel(String label)`
  - `public String getLabel()`
- To get current state of the check box and to set a particular state to check box we have to use the following methods.
  - `public void setState(boolean b)`
  - `public boolean getState()`



## Radio:

- It is an input GUI component, it able to allow to select an item.
- In general, Checkboxes are able to allow multiple selections, but, Radio button is able to allow single selection.
- To create radio buttons we have to use Checkbox class object with CheckboxGroup object reference.

EX:

```
CheckboxGroup cg = new CheckboxGroup();
Checkbox cb1 == new Checkbox("Male", cg, false);
Checkbox cb2 == new Checkbox("Female", cg, false);
```

To set label to checkbox and to get label from Checkbox we have to use the following methods.

- public void setLabel(String label)
- public String getLabel()

To set a particular state to the checkbox and to get current state from checkbox we have to use the following methods.

- public void setState(boolean b)
- public boolean getState()

EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) class UserFrame extends Frame implements ItemListener
4) {
5)     Label l1, l2;
6)     Checkbox cb1, cb2, cb3, cb4, cb5;
7)     String uqual="";
8)     String ugender="";
9)
10)    UserFrame()
11)    {
12)        this.setVisible(true);
13)        this.setSize(500,500);
14)        this.setTitle("Check Box Example");
15)        this.setBackground(Color.green);
16)        this.setLayout(new FlowLayout());
17)        this.addWindowListener(new WindowAdapter()
18)        {
19)            public void windowClosing(WindowEvent we)
20)            {
21)        }
```



```
22)     System.exit(0);
23)   }
24) };
25)
26) l1=new Label("User Qualifications");
27) l2=new Label("User Gender");
28) cb1=new Checkbox("BSC",null, false);
29) cb2=new Checkbox("MCA",null, false);
30) cb3=new Checkbox("PHD",null, false);
31) CheckboxGroup cg=new CheckboxGroup();
32) cb4=new Checkbox("Male",cg, false);
33) cb5=new Checkbox("Female",cg, false);
34)
35) cb1.addItemListener(this);
36) cb2.addItemListener(this);
37) cb3.addItemListener(this);
38) cb4.addItemListener(this);
39) cb5.addItemListener(this);
40)
41) Font f=new Font("arial", Font.BOLD, 15);
42) l1.setFont(f);
43) l2.setFont(f);
44) cb1.setFont(f);
45) cb2.setFont(f);
46) cb3.setFont(f);
47) cb4.setFont(f);
48) cb5.setFont(f);
49)
50) this.add(l1);
51) this.add(cb1);
52) this.add(cb2);
53) this.add(cb3);
54) this.add(l2);
55) this.add(cb4);
56) this.add(cb5);
57) }
58) public void itemStateChanged(ItemEvent ie)
59) {
60)   if(cb1.getState() == true)
61)   {
62)     uqual=uqual+cb1.getLabel()+" ";
63)   }
64)   if(cb2.getState() == true)
65)   {
66)     uqual=uqual+cb2.getLabel()+" ";
```



```
67)    }
68)    if(cb3.getState() == true)
69)    {
70)        uqual=uqual+cb3.getLabel()+" ";
71)    }
72)    if(cb4.getState() == true)
73)    {
74)        ugender=cb4.getLabel();
75)    }
76)    if(cb5.getState() == true)
77)    {
78)        ugender=cb5.getLabel();
79)    }
80)    repaint();
81) }
82) public void paint(Graphics g)
83) {
84)     Font f=new Font("arial", Font.BOLD, 30);
85)     g.setFont(f);
86)     g.drawString("Qualifications :"+uqual, 50, 300);
87)     g.drawString("Gender :" +ugender, 50, 350);
88)     uqual="";
89) }
90} }
91) class CheckBoxEx
92) {
93)     public static void main(String[] args)
94)     {
95)         UserFrame uf=new UserFrame();
96)     }
97) }
```

## List:

- It is an input GUI component, It able to represent more than one element to select.
- To represent this GUI component AWT has provided a predefined class `java.awt.List`

- To create List class object we have to use the following constructors.

```
public List()
public List(int size)
public List(int size, boolean b)
```

- To add an item to List we have to use the following method.

```
public void add(String item)
```



- To get the selected item from List we have to use the following method.  
`public String getSelectedItem()`

- To get the selected items from List we have to use the following method.  
`public String[] getSelectedItems()`

## Choice:

- List is able to allow more than one item selection but Choice is able to allow only one item selection.
- To represent this GUI component AWT has provided a predefined class in the form of `java.awt.Choice`.
- To create Object for Choice class we have to use the following constructor.
- `public Choice()`
- To add an item to Choice object we have to use the following method.
- `public void add(String item)`
- To get the selected item from Choice we have to use the following method.
- `public String getSelectedItem()`

## EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) class UserFrame extends Frame implements ItemListener
4) {
5)     Label l1, l2;
6)     List l;
7)     Choice ch;
8)     String utech="";
9)     String uprof="";
10)
11)    UserFrame()
12)    {
13)        this.setVisible(true);
14)        this.setSize(500,500);
15)        this.setTitle("List Example");
16)        this.setBackground(Color.green);
17)        this.setLayout(new FlowLayout());
18)        this.addWindowListener(new WindowAdapter()
19)        {
20)            public void windowClosing(WindowEvent we)
21)            {
22)                System.exit(0);
23)            }
24)        });
25)
```



```
26)    l1=new Label("User Technologies");
27)    l2=new Label("User Profession");
28)    l=new List(4,true);
29)    l.add("C");
30)    l.add("C++");
31)    l.add("JAVA");
32)    l.add(".NET");
33)    l.add("Oracle");
34)    l.addItemListener(this);
35)
36)    ch=new Choice();
37)    ch.add("Student");
38)    ch.add("Employee");
39)    ch.add("Teacher");
40)    ch.addItemListener(this);
41)
42)    this.add(l1); this.add(l);
43)    this.add(l2); this.add(ch);
44) }
45) public void itemStateChanged(ItemEvent ie)
46) {
47)    String[] items=l.getSelectedItems();
48)    for(int i=0;i<items.length;i++)
49)    {
50)       utech=utech+items[i]+" ";
51)    }
52)    uprof=ch.getSelectedItem();
53)    repaint();
54) }
55) public void paint(Graphics g)
56) {
57)    Font f=new Font("arial", Font.BOLD, 30);
58)    g.setFont(f);
59)    g.drawString("Technologies :" +utech, 50, 300);
60)    g.drawString("Profession :" +uprof, 50, 350);
61)    utech="";
62) }
63) }
64) class ListEx
65) {
66)    public static void main(String[] args)
67)    {
68)       UserFrame uf=new UserFrame();
69)    }
70) }
```



## Menu:

- This GUI component is able to provide a list of items to select.
- To represent Menu in GUI applications, AWT has provided a set of predefined classes like `java.awt.MenuBar`, `java.awt.Menu` and `java.awt.MenuItem`.
- To use menu in GUI applications we have to use the following steps.
  - Create `MenuBar` and Add `MenuBar` to Frame.
  - Create `Menu` and add `Menu` to `MenuBar`.
  - Create `MenuItem`s and add `MenuItem`s to `Menu`.

### EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) class MyFrame extends Frame implements ActionListener
4) {
5)     MenuBar mb;
6)     Menu m;
7)     MenuItem mi1, mi2, mi3;
8)     String item="";
9)
10)    MyFrame()
11)    {
12)        this.setVisible(true);
13)        this.setSize(500,500);
14)        this.setTitle("Menu Example");
15)        this.setBackground(Color.green);
16)        this.addWindowListener(new WindowAdapter()
17)        {
18)            public void windowClosing(WindowEvent we)
19)            {
20)                System.exit(0);
21)            }
22)        });
23)
24)        mb=new MenuBar();
25)        this.setMenuBar(mb);
26)
27)        m=new Menu("File");
28)        mb.add(m);
29)
30)        mi1=new MenuItem("New");
31)        mi2=new MenuItem("Open");
32)        mi3=new MenuItem("Save");
33)        m.add(mi1);
34)        m.add(mi2);
```



```
35)    m.add(mi3);
36)
37)    mi1.addActionListener(this);
38)    mi2.addActionListener(this);
39)    mi3.addActionListener(this);
40) }
41) public void actionPerformed(ActionEvent ae)
42) {
43)    item=ae.getActionCommand();
44)    repaint();
45) }
46) public void paint(Graphics g)
47) {
48)    Font f=new Font("arial", Font.BOLD, 30);
49)    g.setFont(f);
50)    g.drawString("Selected Item :" +item, 50, 300);
51) }
52) }
53) class MenuEx
54) {
55)    public static void main(String[] args)
56)    {
57)       MyFrame mf=new MyFrame();
58)    }
59) }
```

## Scrollbar:

- It able to move frame top to bottom or left to right in order to visible the components.
- To represent Scroll bar AWT has provided a predefined class in the form of `java.awt.Scrollbar`.
- To prepare Scrollbar class object we have to use the following constructor.  
`public Scrollbar(int type)`  
Where type may be `VERTICAL` or `HORIZONTAL` constants from Scrollbar class.

**NOTE:** Scrollbar is able to rise `AdjustmentEvent` and which is handled by `AdjustmentListener` by executing `adjustmentValueChanged(-)` method.



EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) class MyFrame extends Frame implements AdjustmentListener
4) {
5)     Scrollbar sb;
6)     int position;
7)
8)     MyFrame()
9)     {
10)         this.setVisible(true);
11)         this.setSize(500,500);
12)         this.setTitle("Scrollbar Example");
13)         this.setLayout(new BorderLayout());
14)         this.setBackground(Color.green);
15)         this.addWindowListener(new WindowAdapter()
16)         {
17)             public void windowClosing(WindowEvent we)
18)             {
19)                 System.exit(0);
20)             }
21)         });
22)
23)         sb=new Scrollbar(Scrollbar.VERTICAL);
24)         sb.addAdjustmentListener(this);
25)
26)         this.add(BorderLayout.EAST, sb);
27)     }
28)     public void adjustmentValueChanged(AdjustmentEvent ae)
29)     {
30)         position=sb.getValue();
31)         repaint();
32)     }
33)     public void paint(Graphics g)
34)     {
35)         Font f=new Font("arial", Font.BOLD, 30);
36)         g.setFont(f);
37)         g.drawString("Position : "+position, 50, 300);
38)     }
39)
40) class ScrollBarEx
41) {
42)     public static void main(String[] args)
43)     {
44)         MyFrame mf=new MyFrame();
```



```
45) }  
46) }
```

## Applets:

- Applet is a container, it able to manage some other GUI components.
- To represent Applet, JAVA has provided a predefined class in the form of `java.applet.Applet`.
- If we want to use Applets in GUI applications then we have to use the following steps.
  - Create Applet class
  - Create Applet Configuration File
  - Execute Applet

### • Create Applet Class

- Declare an user defined class
- Extend `java.applet.Applet` class to user defined class.
- In user defined Applet class, override Applet lifecycle methods like

```
public void init()  
public void start()  
public void stop()  
public void destroy()
```

### • Create Applet Configuration File

- The main intention of Applet configuration file is to declare Applet class and its location and to declare size of the Applet,.....
- Applet configuration file is an html file, which includes `<applet>` tag.

**Syntax:** `<applet code="---" width="--" height="--">  
 </applet>`

Where "code" attribute will take the name and location of the Applet class.

Where "width" and "height" attributes are used to take applet size.

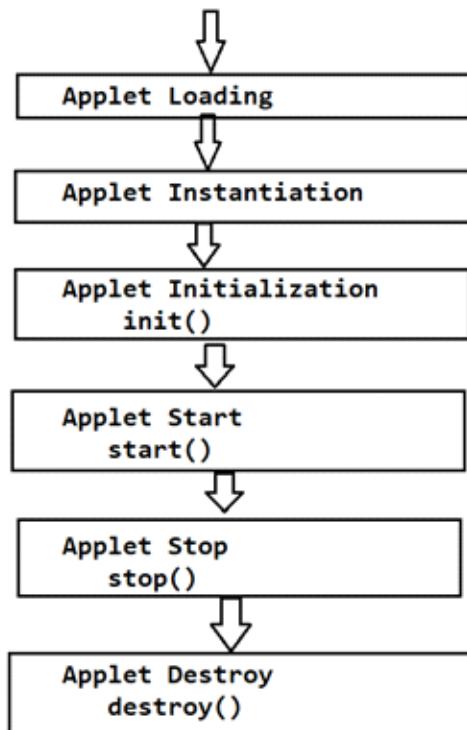
### • Execute Applet:

There are two approaches to execute Applets.

- Use "appletviewer" command on command prompt.  
**EX:** `D:\java830>appletviewer MyApplet.html`
- Use browser to execute Applet  
In this case, open Applet configuration file in browser.



**NOTE:** In general, Applets are executed by following Applet lifecycle only, Applets are not having main() method to execute.



## LogoApplet.java

```
1) import java.awt.*;
2) import java.applet.*;
3) public class LogoApplet extends Applet
4) {
5)     public void paint(Graphics g)
6)     {
7)         Font f=new Font("arial", Font.BOLD, 30);
8)         g.setFont(f);
9)         g.drawString("DURGA SOFTWARE SOLUTIONS",100,100);
10)    }
11) }
```

## LogoApplet.html

```
<applet code="LogoApplet" width="700" height="500">
</applet>
```

On Command Prompt

```
D:\java830>javac LogoApplet.java
D:\java830>appletviewer LogoApplet.html
```



## SWING:

Swing is a package in Java, it able to provide all GUI components to prepare GUI applications.

### Q) What are the differences between AWT and SWING?

- AWT GUI components are platform dependent GUI components.  
SWING GUI Components are platform Independent GUI Components.
- AWT GUI Components are able to follow the local operating system provided Look And Feel, it is fixed up to the present operating system.  
SWING GUI components are having Pluggable Look And Feel nature irrespective of the operating system which we used.
- AWT GUI components are Heavy Weight GUI components, they will more execution time and much memory.  
Swing GUI components are light weight GUI components; they will take less memory and less execution time.
- AWT GUI components are able to provide less performance in GUI applications.  
SWING GUI components are able to provide very good performance in GUI applications.
- AWT GUI components are PEER GUI components, for every GUI component a separate duplicate component is created at Operating System.  
SWING GUI components are not PEER GUI components, no separate duplicate component will be created at Operating System.
- AWT is not following MVC [Model-View-Controller] design pattern internally to design GUI applications.  
SWING is following MVC design pattern to design GUI applications.
- AWT is not providing tool tip text support.  
SWING is providing Tool Tip Text Support
- In AWT, we are able to add all the GUI components to Frame directly.  
In SWING, we are able to add all the GUI components to the Panes.

NOTE: There are four types of Panes in SWING

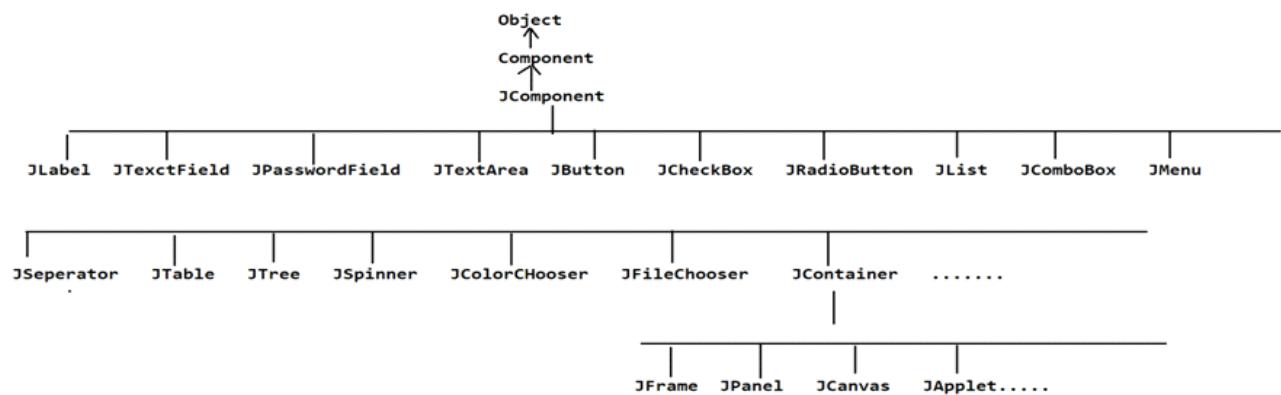
- RootPane
- LayeredPane
- ContentPane
- GlassPane.



In SWING we will use mainly ContentPane to manage GUI components, where to get ContentPane we have to use the following method.

```
public Container getContentPane()
```

To represent all GUI components in SWING, javax.swing package has provided the following predefined classes.



### NOTE:

To close Frames in AWT we have to use **WindowListener** implementation class or **WindowAdapter** abstract class or Anonymous inner class of **WindowAdapter** abstract class. In case of SWING to provide window closing option to the Frame we have to use the following instruction.

```
f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

### EX:

```
1) import javax.swing.*;
2) import javax.swing.event.*;
3) import java.awt.*;
4) import java.awt.event.*;
5) class RegistrationFrame extends JFrame implements ActionListener
6) {
7)     JLabel l1,l2,l3,l4,l5,l6,l7;
8)     JTextField tf;
9)     JPasswordField pf;
10)    JCheckBox cb1,cb2,cb3;
11)    JRadioButton rb1,rb2;
12)    JList l;
13)    JComboBox cb;
14)    JTextArea ta;
15)    JButton b;
16)
17)    Container c;
```



```
18) String uname="";upwd="";uqual="";ugen="";utech="";uprof="";uaddr="";
19) RegistrationFrame()
20) {
21)     this.setVisible(true);
22)     this.setSize(500,600);
23)     this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
24)     c=ContentPane();
25)     c.setBackground(Color.cyan);
26)     //this.setForeground(Color.red);
27)     c.setLayout(null);
28)     l1=new JLabel("User Name");
29)     l1.setBounds(50,100,100,10);
30)     l2=new JLabel("Password");
31)     l2.setBounds(50,150,100,10);
32)     l3=new JLabel("Qualification");
33)     l3.setBounds(50,200,100,10);
34)     l4=new JLabel("Gender");
35)     l4.setBounds(50,250,100,10);
36)     l5=new JLabel("Technologies");
37)     l5.setBounds(50,300,100,10);
38)     l6=new JLabel("Proffession");
39)     l6.setBounds(50,350,100,10);
40)     l7=new JLabel("Address");
41)     l7.setBounds(50,400,100,10);
42)
43)     tf=new JTextField(20);
44)     tf.setBounds(150,90,100,30);
45)     tf.setToolTipText("This Is Text Field");
46)
47)     pf=new JPasswordField(20);
48)     pf.setBounds(150,140,100,30);
49)     pf.setToolTipText("This Is Password Field");
50)
51)     cb1=new JCheckBox("BSC");
52)     cb1.setBounds(150,190,60,30);
53)     cb2=new JCheckBox("MCA");
54)     cb2.setBounds(220,190,60,30);
55)     cb3=new JCheckBox("PHD");
56)     cb3.setBounds(290,190,60,30);
57)
58)     rb1=new JRadioButton("Male");
59)     rb1.setBounds(150,240,80,30);
60)     rb2=new JRadioButton("Female");
61)     rb2.setBounds(250,240,80,30);
62)     ButtonGroup bg=new ButtonGroup();
```



```
63)    bg.add(rb1);
64)    bg.add(rb2);
65)
66)    String[] techs={"C","C++","JAVA"};
67)    l=new JList(techs);
68)    l.setBounds(150,280,60,60);
69)
70)    String[] prof={"Student","Business","Teacher"};
71)    cb=new JComboBox(prof);
72)    cb.setBounds(150,340,80,30);
73)
74)    ta=new JTextArea(5,25);
75)    ta.setBounds(150,380,100,40);
76)
77)    b=new JButton("Registration");
78)    b.setBounds(50,450,110,40);
79)    b.addActionListener(this);
80)
81)    c.add(l1);c.add(tf);
82)    c.add(l2);c.add(pf);
83)    c.add(l3);c.add(cb1);c.add(cb2);c.add(cb3);
84)    c.add(l4);c.add(rb1);c.add(rb2);
85)    c.add(l5);c.add(l);
86)    c.add(l6);c.add(cb);
87)    c.add(l7);c.add(ta);
88)    c.add(b);
89) }
90) public void actionPerformed(ActionEvent ae)
91) {
92)    uname=tf.getText();
93)    upwd=pf.getText();
94)    if(cb1.isSelected()==true)
95)    {
96)       uqual=uqual+cb1.getLabel()+" ";
97)    }
98)    if(cb2.isSelected()==true)
99)    {
100)       uqual=uqual+cb2.getLabel()+" ";
101)    }
102)    if(cb3.isSelected()==true)
103)    {
104)       uqual=uqual+cb3.getLabel()+" ";
105)    }
106)    if(rb1.isSelected()==true)
107)    {
```



```
108)         ugen=rb1.getLabel();
109)     }
110)     if(rb2.isSelected()==true)
111)     {
112)         ugen=rb2.getLabel();
113)     }
114)     Object[] techs=l.getSelectedValues();
115)     for(int i=0;i<techs.length;i++)
116)     {
117)         utech=utech+techs[i]+" ";
118)     }
119)     uprof=(String)cb.getSelectedItem();
120)     uaddr=ta.getText();
121)
122)     class DisplayFrame extends JFrame
123)     {
124)         DisplayFrame()
125)         {
126)             this.setVisible(true);
127)             this.setSize(500,500);
128)             this.setBackground(Color.pink);
129)         }
130)         public void paint(Graphics g)
131)         {
132)             Font f=new Font("arial",Font.BOLD,25);
133)             g.setFont(f);
134)             g.drawString("User Name :"+uname,50,100);
135)             g.drawString("Password :"+upwd,50,150);
136)             g.drawString("Qualification :"+uqual,50,200);
137)             g.drawString("User Gender :"+ugen,50,250);
138)             g.drawString("Technologies :"+utech,50,300);
139)             g.drawString("Proffession :"+uprof,50,350);
140)             g.drawString("Address :"+uaddr,50,400);
141)         }
142)     }
143)     DisplayFrame df=new DisplayFrame();
144) }
145)
146) class RegistrationApp {
147)     public static void main(String[] args)
148)     {
149)         RegistrationFrame rf=new RegistrationFrame();
150)     }
151) }
```



## JColorChooser:

It is a swing specific GUI component, it able to display grids of colors to select.

To represent this GUI component, SWING has provided a predefined class in the form of `javax.swing.JColorChooser`.

To create `JColorChooser` class object we have to use the following constructor.

```
public JColorChooser()
```

To get the current `SelectionModel` from `JColorChooser` we have to use the following method.

```
public SelectionModel getSelectionModel()
```

In GUI applications, `JColorChooser` is able to rise `ChangeEvent` and it will be handled by `ChangeListener` by executing the following method.

```
public void stateChanged(ChangeEvent ce)
```

To get the selected color from `JColorChooser` we have to use the following method.

```
public Color getColor()
```

EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) import javax.swing.*;
4) import javax.swing.event.*;
5) class ColorChooserFrame extends JFrame implements ChangeListener
6) {
7)     JColorChooser cc;
8)     Container c;
9)     ColorChooserFrame()
10)    {
11)        this.setVisible(true);
12)        this.setSize(500,500);
13)        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
14)        c=this.getContentPane();
15)
16)        cc=new JColorChooser();
17)        cc.getSelectionModel().addChangeListener(this);
18)        c.setLayout(new FlowLayout());
19)        c.add(cc);
20)    }
21)    public void stateChanged(ChangeEvent ce)
22)    {
23)        Color clr=cc.getColor();
24)        JFrame f=new JFrame();
```



```
25) f.setVisible(true);
26) f.setSize(300,300);
27) f.getContentPane().setBackground(clr);
28) }
29) }
30) class ColorChooserEx
31) {
32) public static void main(String[] args)
33) {
34) ColorChooserFrame cc=new ColorChooserFrame();
35) }
36) }
```

## JFileChooser:

It is a swing specific GUI component, it able to display all the file system which is available in our system hard disk in order to select.

To represent this GUI component, SWING has provided a predefined class in the form of `javax.swing.JFileChooser`.

To create `JFileChooser` class object we have to use the following constructor.  
`public JFileChooser()`

To get Selected file from `JFileChooser` we have to use the following method.  
`public File getSelectedFile()`

In GUI applications, `JFileChooser` is able to rise `ActionEvent` and it would be handled by `ActionListener` by executing "`actionPerformed(ActionEvent ae)`".

EX:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) import javax.swing.*;
4) import javax.swing.event.*;
5) import java.io.*;
6) class FileChooserFrame extends JFrame implements ActionListener
7) {
8) JLabel l;
9) JTextField tf;
10) JButton b;
11) Container c;
12) JFileChooser fc;
13) FileChooserFrame()
14) {
```



```
15)    this.setVisible(true);
16)    this.setSize(500,500);
17)    this.setTitle("File Chooser Example");
18)    this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
19)    c=this.getContentPane();
20)    c.setBackground(Color.pink);
21)    c.setLayout(new FlowLayout());
22)
23)    l=new JLabel("Select File");
24)    tf=new JTextField(20);
25)    b=new JButton("Browse");
26)    b.addActionListener(this);
27)    c.add(l);c.add(tf);c.add(b);
28} 
29) public void actionPerformed(ActionEvent ae)
30) {
31)     class FileDialogFrame extends JFrame implements ActionListener
32)     {
33)         FileDialogFrame()
34)         {
35)             this.setVisible(true);
36)             this.setSize(500,500);
37)             fc=new JFileChooser();
38)             fc.addActionListener(this);
39)             this.getContentPane().add(fc);
40)         }
41)         public void actionPerformed(ActionEvent ae)
42)         {
43)             File f=fc.getSelectedFile();
44)             String path=f.getAbsolutePath();
45)             tf.setText(path);
46)             this.setVisible(false);
47)         }
48)     }
49)     FileDialogFrame ff=new FileDialogFrame();
50) }
51}
52) class FileChooserEx
53) {
54)     public static void main(String[] args)
55)     {
56)         FileChooserFrame f=new FileChooserFrame();
57)     }
58}
```



## JTable

It is a SWING specific GUI component, it able to display data in rows and columns.

To represent this GUI component, SWING has provided a predefined class in the form of `javax.swing.JTable`.

To create `JTable` class object we have to use the following constructor.

`public JTable(Object[][] body, Object[] header)`

EX:

```
1) import javax.swing.*;
2) import javax.swing.table.*;
3) import java.awt.*;
4) class TableEx
5) {
6)     public static void main(String[] args)
7)     {
8)         String[] header={"EID","ENAME","ESAL","EADDR"};
9)         Object[][] body={{"e1","AAA","5000","Hyderabad"}, {"e2","BBB","6000",
 "Secbad"}, {"e3","CCC","7000","Vijayawada"}, {"e4","DDD","8000","Warngal"}};
10)        JFrame f=new JFrame();
11)        f.setVisible(true);
12)        f.setSize(500,500);
13)        Container c=f.getContentPane();
14)        c.setLayout(new BorderLayout());
15)        JTable t=new JTable(body,header);
16)        JTableHeader th=t.getTableHeader();
17)        c.add(th,BorderLayout.NORTH);
18)        c.add(t,BorderLayout.CENTER);
19)    }
20} }
```



# Internationalization (I18N)



---

Designing Java applications w.r.t the local users is called as Internationalization.

Java is able to provide very good Internationalization support due to UNICODE representations in java.

UNICODE is one of the character representations following by JAVA, it able to represent all the alphabet from all the natural languages.

In java applications, if we want to provide Internationalization services, first, we have to represent a group of local users in java programs, to represent group of local users, first we have to divide all the users in no of groups.

To divide users into no of groups, we have to use the following parameters:

- Language: It will be represented in the form of two lower case letters  
EX: en, hi, it,.....
- Country: It will be represented in the form of two upper case letters.  
EX: US, IN, IT,.....
- System Variant [OS]: It will be represented in the form of three lower case letters.  
EX: win, lin, uni,.....

To represent a group of Local users in java applications, JAVA has provided a predefined class in the form of "java.util.Locale".

To create Objects for java.util.Locale class we have to use the following constructors.

- public Locale(String lang)
- public Locale(String lang, String country)
- public Locale(String lang, String country, String Sys\_Var)

EX: Locale l = new Locale("en");  
Locale l = new Locale("en", "US");  
Locale l = new Locale("it", "IT", "win");

Java is able to provide Internationalization support in the form of the following three services.

- Number Formations
- Date Formations
- Message Formations



## • **Number Formations:**

- From language to language, country to country number representations are varied.  
To represent numbers w.r.t a particular Locale JAVA has provided a predefined class in the form of "java.text.NumberFormat"
- To represent number w.r.t a particular Locale by using NumberFormat class we have to use the following steps.

## • **Create Locale Class Object:**

Locale l = new Locale("en", "US", "win");

## • **Create NumberFormat Class Object:**

- To create NumberFormat class object we have to use the following static Factory Method from java.text.NumberFormat class.
- public static NumberFormat getInstance(Locale l)  
EX: NumberFormat nf = NumberFormat.getInstance();

## • **Format a particular Number w.r.t a particular Locale:**

- To format a number w.r.t a particular Local we have to use the following method from java.text.NumberFormat class.
- public String format(xxx num)  
Where xxx may be byte, short, int,.....  
EX: String data = nf.format(123456.2345f);

EX:

```
1) import java.util.*;
2) import java.text.*;
3) class I18NEx
4) {
5)     public static void main(String[] args) throws Exception
6)     {
7)         Locale l=new Locale("it","IT","win");
8)         NumberFormat nf=NumberFormat.getInstance(l);
9)         System.out.println(nf.format(1234567.3456));
10)    }
11) }
```



- **Date Formations:**

- From language to language, country to country Date representations are varied. To represent Dates w.r.t a particular Locale JAVA has provided a predefined class in the form of "java.text.DateFormat"
- To represent Date w.r.t a particular Locale by using DateFormat class we have to use the following steps.

- **Create Locale Class Object:**

Locale l = new Locale("en", "US", "win");

- **Create DateFormat Class Object:**

To create DateFormat class object we have to use the following static factory method from java.text.DateFormat class.

```
public static DateFormat getDateInstance(int date_Style, Locale l)
Where date_Style may be 0,1,2 and 3.  
EX: DateFormat df = DateFormat.getDateInstance(0, l);
```

- **Format current System Date w.r.t a particular Locale:**

To format current System Date w.r.t a particular Locale we have to use the following method from java.text.DateFormat class.

```
public String format(Date d)  
EX: String date = df.format(new Date());
```

EX:

```
1) import java.util.*;  
2) import java.text.*;  
3) class I18NEx  
4) {  
5)   public static void main(String[] args)throws Exception  
6)   {  
7)     Locale l=new Locale("it","IT","win");  
8)     DateFormat df=DateFormat.getDateInstance(3,l);  
9)     System.out.println(df.format(new Date()));  
10)    }  
11) }
```



- **Message Formations:**

- From language to language, from country to country, messages representations are varied, if we want to represent message w.r.t a particular Locale then we have to use `java.util.ResourceBundle` class provided by JAVA.
- To represent messages w.r.t a particular Locale by using `ResourceBundle` class we have to use the following steps.

- **Prepare Properties Files for each and every Locale:**

In properties files we have to provide locale respective messages in the form of Key-Value pairs, where keys must be in English but values must be in local respective messages.

We have to provide properties files names in the following format.

`File_name_<lang>_<country>_<sys_Var>.properties`

**EX 1:** `abc_en_US.properties`

welcome = Welcome to en US Users.

**EX 2:** `abc_it_IT.properties`

welcome = Welcomeo Toe it IT Userse.

**EX 3:** `abc_hi_IN.properties`

welcome = Aap ka swagath hai.

- **In Java program, Create Locale object:**

`Locale l = new Locale("en", "US");`

- **Prepare ResourceBundle class Object:**

The main intention of `ResourceBundle` object is to store the content of a particular properties file.

To create `ResourceBundle` Object we have to use the following method.

`public static ResourceBundle getBundle(String base_Name, Locale l)`

**EX:** `ResourceBundle rb = ResourceBundle.getBundle("abc",l);`



- **Get Message from ResourceBundle:**

To get Message from ResourceBundle object we have to use the following method.

```
public String getString(String key)
```

```
String message = rb.getString("welcome");
```

**EX:**

**abc\_en\_US.properties**

welcome = Welcome to en US Users.

**abc\_it\_IT.properties**

welcome = Welcomeo To it IT Users.

**abc\_hi\_IN.properties**

welcome = Aap ka swagath hai.

## I18NEx.java

```
1) import java.util.*;  
2) class I18NEx  
3) {  
4)     public static void main(String[] args)throws Exception  
5)     {  
6)         Locale l=new Locale("hi","IN");  
7)         ResourceBundle rb=ResourceBundle.getBundle("abc",l);  
8)         System.out.println(rb.getString("welcome"));  
9)     }  
10) }
```



# Reflection API



- Introduction
- Class
- Method
- Field
- Constructor

## **Reflection:**

The process of analyzing all the capabilities of a particular class at runtime is called as "Reflection".

Reflection API is a set of predefined classes provided by Java to perform reflection over a particular class.

Reflection API is not useful in projects development, reflection API is useful in products development like Compilers, JVM's, Server's, IDE's, FrameWork's.....

Java has provided the complete predefined library for Reflection API in the form of "java.lang" package and "java.lang.reflect" package.

Java.lang  
Class  
java.lang.reflect  
Field  
Method  
Constructor  
Package  
Modifier  
---  
---

## **java.lang.Class:**

This class can be used to represent a particular class metadata which includes class name, super class details, implemented interfaces details, access modifiers.....

To get the above class metadata, first we have to create Class Object for the respective class.

There are three ways to create object for java.lang.Class

### **Using forName(--) Method:**

forName() method is defined in java.lang.Class to load a particular class byte code to the memory and to get metadata of the respective loaded class in the form of Class object.

```
public static Class forName(String class_Name) throws ClassNotFoundException  
class c = Class.forName("Employee");
```



When JVM encounter the above instruction, JVM will perform the following actions.

- JVM will take provided class from `forName()` method.
- JVM will search for its .class file at current location, at java predefined library and at the locations reoffered by "classpath" environment variable.
- If the required .class file is not available at all the above locations then JVM will rise an exception like "java.lang.ClassNotFoundException".
- If the required .class file is available at either of the above locations then JVM will load its byte code to the memory.
- After loading its byte code to the memory, JVM will collect the respective class metadata and stored in the form of `java.lang.Class` object.

## • Using `getClass()` Method:

- In Java applications, if we create object for any class then JVM will load the respective class byte code to the memory, JVM will get metadata of the respective class and stored in the form of `java.lang.Class` object.
- In the above context, to get the generated Class object, we have to use the following method from `java.lang.Object` class.

`public Class getClass()`

EX:

```
Employee e = new Employee();
Class c = e.getClass();
```

## • Using .class File Name:

- In Java, every .class file is representing a `java.lang.Class` Object, it will manage the metadata of the respective class.

EX: `Class c = Employee.class`

- To get name of the class, we have to use the following method.

`public String getName()`

- To get super class metadata in the form of Class object, we have to use the following method.

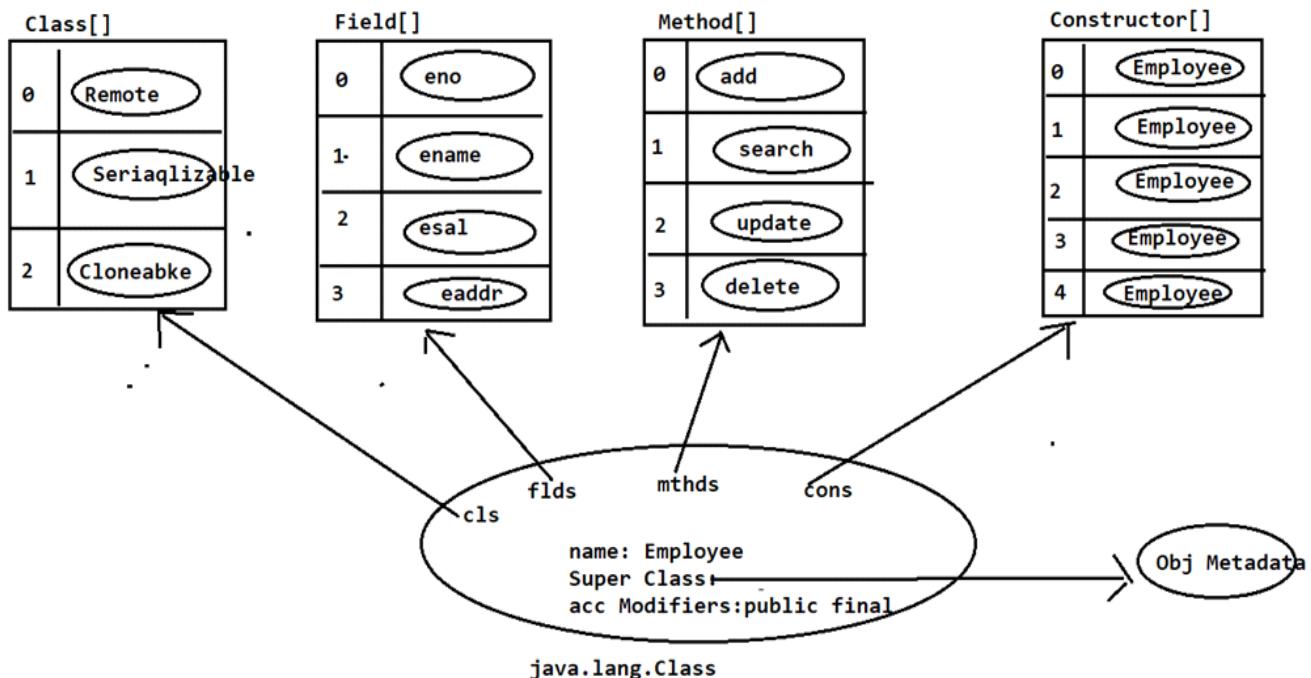
`public Class getSuperclass()`

- To get implemented interfaces metadata in the form of `Class[]` we have to use the following method.

`public Class[] getInterfaces()`

- To get the specified access modifiers list, we have to use the following method.

`public int getModifiers()`



## EX:

```
1) int val = c.getModifiers()
2) String modifiers = Modifier.toString(val);
3) import java.lang.reflect.*;
4) public abstract class Employee implements java.io.Serializable,java.lang.Cloneable
{
5)
6) class Test {
7)     public static void main(String args[]) throws Exception {
8)
9)         Class c1 = Clas.forName("Employee");
10)        System.out.println(c1.getName());
11)        Employee e = new Employee();
12)
13)        Class c2 = e.getClass();
14)        System.out.println(c2.getName());
15)
16)        Class c3 = Employee.class;
17)        System.out.println(c3.getName());
18)
19)        Class c = Class.forName("Employee");
20)        System.out.println("Class Name :" +c.getName());
21)        System.out.println("Super Class :" +c.getSuperclass().getName());
22)        Class[] cls = c.getInterfaces();
23)        System.out.println("Interfaces :");
24)
```



```
25)         for(int i=0; i<cls.length; i++) {  
26)             Class cl = cls[i];  
27)             System.out.println(cl.getName()+" ");  
28)         }  
29)         System.out.println();  
30)         int val = c.getModifiers();  
31)         System.out.println("Modifiers :" +Modifier.toString(val));  
32)     }  
33} }
```

## java.lang.reflect.Field:

- This class can be used to get the metadata of a particular variable which contains the name of the variable, data type of the variable, access modifiers of the variable and value of the variable.
- If we want to get all the variables metadata of a particular class first we have to get `java.lang.Class` object. After getting `Class` object, we have to get all the variables metadata in the form `Field[]`.
- To get all the variables metadata in the form of `Field[]`, we have to use the following two methods.

- **public Field[] getFields()**

This method will return all the variables metadata which are declared as public in the respective class and in the super class.

- **public Field[] getDeclaredFields()**

This method will return all the variables metadata which are available in the respective class only irrespective of public declaration.

- To get name of the variable, we have to use the following method.

`public String getName()`

- To get datatype of the variables, we have to use the following method.

`public Class getType()`

- To get value of the variable, we have to use the following method.

`public Object get(Field f)`

- To get access modifiers of a variable, we have to use the following method.

`public int getModifiers()`

`public static String toString(int val)`



```
1) import java.lang.reflect.*;
2) class Employee {
3)     public static String eid = "E-111";
4)     public static String ename = "Durga";
5)     public static String eaddr = "Hyd";
6) }
7) class Test {
8)     public static void main(String args[]) throws Exception {
9)         Class c = Employee.class;
10)        Field[] flds = c.getDeclaredFields();
11)        for(int i=0; i<flds.length; i++) {
12)            Field f = flds[i];
13)            System.out.println("Field Name:" + f.getName());
14)            System.out.println("data Type:" + f.getType().getName());
15)            System.out.println("value:" + f.get(f));
16)            int val = f.getModifiers();
17)            System.out.println("Modifiers :" + Modifier.toString(val));
18)            System.out.println("-----");
19)        }
20)    }
21} }
```

## java.lang.Method:

- This class can be used to represent the metadata of a particular method, which includes method name, return type, parameter types, access modifiers and exception types
- If we want to get all the methods metadata in the form of method[], first we have to get java.lang.Class object then we have to use either of the following methods.  
`public Method[] getMethods()`
- It can be used to get all the methods metadata which are declared as public in the respective class and in the super class.

`public Method[] getDecalredMethods()`

It can be used to get all the methods metadata which are available in the respective class irrespective of public declaration.

- To get method name, we have to use the following method.  
`public String getName()`
- To get method return type, we have to use the following method.  
`public Class getReturnType()`
- To get parameter data types, we have to use the following method.  
`public Class[] getParameterTypes()`



- To get Exception types which are specified along with "throws" keyword then we have to use the following method.

```
public Class[] getExceptionTypes()
```

- To get Access modifier of a particular class, we have to use the following method.

```
public int getModifiers()
```

```
public static String toString(int value)
```

```
1) import java.lang.reflect.*;
2) class Employee {
3)     public void create(String eid, String ename, String eaddr) throws
   ClassNotFoundException, NullPointerException {
4)
5) }
6)     public void search(String eid) throws ClassCastException {}
7)     public void delete(String eid) throws ClassCastException,
   ClassNotFoundException {}
8)
9) }
10) class Test {
11)     public static void main(String args[]) throws Exception {
12)         Class c = Employee.class;
13)         Method[] mthds = c.getDeclaredMethods();
14)
15)         for(int i=0; i<mthds.length; i++) {
16)             Method m = mthds[i];
17)             System.out.println("Method Name :" +m.getName());
18)             System.out.println("Return Type :" +m.getReturnType());
19)             int val = m.getModifiers();
20)             System.out.println("Modifiers :" +modifier.toString(val));
21)             Class[] cls = m.getParameterTypes();
22)             System.out.println("Parametes :");
23)
24)             for(int j=0; j<cls.length; j++) {
25)                 Class cl = cls[j];
26)                 System.out.println(cl.getName() + " ");
27)             }
28)
29)             System.out.println();
30)             Class[] cls1 = m.getExceptionTypes();
31)             System.out.println("Exception Types :");
32)
33)             for(int j=0; j<cls1.length; j++) {
34)                 Class cl1 = cls1[j];
35)                 System.out.println(cl1.getName() + " ");
36)             }
37)
```



```
38)         System.out.println();
39)         System.out.println("-----");
40)     }
41)
42} }
```

## java.lang.reflect.Constructor:

This class can be used to get metadata of a particular constructor.

To get all the constructors metadata of a particular class first we have to get Class object then we have to use the following methods.

`public Constructor[] getConstructors()`

It will return only public constructors details from the respective class.

`public Constructor[] getDeclaredConstructors()`

It will return all the constructors metadata of the respective class irrespective of their public declaration.

Constructor class has provided the following methods to get constructor data.

- 1) `public String getName()`
- 2) `public Class[] getParameterTypes()`
- 3) `public Class[] getExceptionTypes()`
- 4) `public int getModifiers()`
- 5) `public static String toString(int val)`

Program is same as `java.lang.reflect.Method` program with the above methods

```
1) import java.lang.reflect.*;
2) class Employee {
3)     public Employee(String eid, String ename, String eaddr) throws
   ClassNotFoundException, NullPointerException {
4)     }
5)     public Employee(String eid, String ename) throws ClassCastException {}
6)     public Employee(String eid) throws ClassCastException,
   ClassNotFoundException {}
7)
8)
9) }
10) class Test {
11)     public static void main(String args[]) throws Exception {
12)         Class c = Employee.class;
13)         Constructor[] con = c.getDeclaredConstructors();
14)         for(int i=0; i<con.length; i++) {
15)             Constructor constructor = con[i];
16)             System.out.println("Constructor Name:" constructor.getName());
17)             int val = constructor.getModifiers();
```



```
18)         System.out.println("Modifiers:"+Modifier.toString(val));
19)         Class[] cls = constructor.getParameterTypes();
20)         System.out.println("Parametes :");
21)         for(int j=0; j<cls.length; j++) {
22)             Class cl = cls[j];
23)             System.out.println(cl.getName()+" ");
24)         }
25)         System.out.println();
26)         Class[] cls1 = constructor.getExceptionTypes();
27)         System.out.println("Exception Types :");
28)         for(int j=0; j<cls1.length; j++) {
29)             Class cl1 = cls1[j];
30)             System.out.println(cl1.getName()+" ");
31)         }
32)         System.out.println();
33)         System.out.println("-----");
34)     }
35) }
36} }
```



# Annotations



- Introduction
- Comment Vs Annotations
- XML Vs Annotations
- Types of Annotations
- Standard Annotations
- Custom Annotations

## **Annotation:**

Annotation is a Java Feature provided by JDK 5.0 version, it can be used to represent metadata in Java applications.

### **Q) In Java Applications, to describe Metadata we have already Comments then what is the Requirement to go for Annotations?**

- In Java applications, if we provide metadata by using comments then "Lexical Analyzer" will remove comments metadata from Java program as part of Compilation.
- As per the application requirement, if we want to bring metadata up to .java file, up to .class file and up to RUNTIME of our application then we have to use "Annotations".

### **Q) In Java Applications, to provide Metadata at Runtime of Java Applications we are able to use XML Documents then what is the Requirement to go for "Annotations"?**

In Java applications, if we provide metadata by using XML documents then we are able to get the following problems.

- Every time we have to check whether XML documents are located properly or not.
- Every time we have to check whether XML documents are formatted properly or not.
- Every time we have to check whether we are using right parsing mechanism or not to access the data from XML document.
- First Developers must aware XML tech.

To overcome all the above problems, we have to use Java alternative that is "Annotation".

### **EX: Servlet Configuration with XML Document**

#### web.xml

- 1) <web-app>
- 2)
- 3)   <servlet>
- 4)     <servlet-name>ls</servlet-name>
- 5)     <servlet-class>LoginServlet</servlet-class>
- 6)   </servlet>



```
7) <servlet-mapping>
8)     <servlet-name>ls</servlet-name>
9)     <url-pattern>/login</url-pattern>
10)    </servlet-mapping>
11)
12) </web-app>
```

## Servlet Configuration with Annotation

```
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
}
```

| XML-Based Tech  | Annotation Based Tech |
|-----------------|-----------------------|
| JDK 1.4         | JDK 5.0               |
| JDBC 3.0        | JDBC 4.0              |
| Servlets 2.5    | Servlets 3.0          |
| Struts 1.x      | Struts 2.x            |
| JSF 1.x         | JSF 2.x               |
| Hibernate 3.2.4 | Hibernate 3.5         |
| EJBs 2.x        | EJBs 3.x              |
| Spring 2.x      | Spring 3.x            |

**NOTE:** Annotations are not the complete alternative for XML tech, with in a Java application we can use annotations as an alternative for XML documents but when we go for distributed applications where applications are designed on different tech standards there annotations are not possible, only we have to use XML tech.

To process the annotations, Java has provided a predefined tool in the form of "Annotation Processing tool"[APT], it was managed by Java up to JAVA 7 version, it was removed from Java in JAVA8 version.

## Syntax to declare Annotation:

```
@interface Annotation_Name {
    data_Type member_Name() [default] value;
}
```

## Syntax to use Annotation:

```
@Annotation_Name(member_name1=value1,member_Name2=value2.....)
```

In Java, all the annotations are interfaces by default.

In Java, the common and default super type for all the annotations is "java.lang.annotation.Annotation".

As per the number of members inside annotations, there are three types of annotations.



- **Marker Annotations:**

It is an annotation without members.

EX: @interface Override {  
    }

- **Single-Valued Annotation:**

It is an Annotation with exactly one single member

EX: @interface SuppressWarnings {  
    String value();  
}

- **Multi-Valued Annotation:**

It is an annotation with more than one member.

EX: @interface WebServlet {  
    int loadOnStartup();  
    String[] urlPatterns();  
    ----  
}

In Java Annotations are divided into the following two types as per their nature.

- Standard Annotations
- Custom Annotations

- **Standard Annotations:**

These are predefined Annotations provided by Java along with Java software.

There are two types of Standard Annotations

- General Purpose Annotations
- Meta Annotations

- **General Purpose Annotations:**

These Annotations are commonly used Annotations in Java applications

These Annotations are provided by Java as part of java.lang package.

EX: @Override  
    @Deprecated  
    @SuppressWarnings  
    @FunctionalInterface [JAVA 8]



- **Meta Annotations:**

These Annotations can be used to define another Annotations.

These Annotations are provided by Java as part of `java.lang.annotation` package.

EX: `@Inherited`  
`@Documented`  
`@Target`  
`@Retention`

- **@Override:**

- In Java applications if we want to perform method overriding then we have to provide a method in subclass with the same prototype of super class method.
- While performing method overriding, if we are not providing the same super class method at sub class method then compiler will not rise any error and JVM will not rise any exception, JVM will provide super class method output.
- In the above context, if we want to get an error about to describe failure case of method overriding from compiler then we have to use `@Override`

EX:

```
1) class JdbcApp {  
2)     public void getDriver() {  
3)         System.out.println("Type-1 Driver");  
4)     }  
5) }  
6) class New_JdbcApp extends JdbcApp {  
7)     @Override  
8)     public void getdriver() {  
9)         System.out.println("Type-4 Driver");  
10)    }  
11) }  
12) class Test {  
13)     public static void main(String args[]) {  
14)         JdbcApp app = new New_JdbcApp();  
15)         app.getDriver();  
16)     }  
17) }
```

If we compile the above program then compiler will rise an error like "method does not override or implement a method from a SuperType".

**NOTE:** If we compile the above programme, compiler will recognize `@Override` annotation, compiler will take sub class method name which is marked with `@Override` annotation, compiler will compare sub class method name with all the super class method names. If any method name is matched then compiler will not rise any error.



If no super class method name is matched with sub class method name then compiler will rise an error.

- **@Deprecated:**

The main intention of this annotation is to make a method as deprecated method and to provide deprecation message when we access that deprecated method.

In general, Java has provided Deprecation support for only predefined methods, if we want to get deprecation support for the user defined methods we have to use `@Deprecated` annotation.

**NOTE:** Deprecated method is an outdated method introduced in the initial versions of Java and having alternative methods in the later versions of Java.

```
1) class Employee {  
2)     @Deprecated  
3)     public void gen_Salary(int basic, float hq) {  
4)         System.out.println("Salary is calculated on the basis of basic amount,hq");  
5)     }  
6)     public void gen_Salary(int basic, float hq, int ta, float pf) {  
7)         System.out.println("Salary is calculated on the basis of basic amount,hq,ta,pf");  
8)     }  
9) }  
10) class Test {  
11)     public static void main(String args[]) {  
12)         Empoloyee emp = new Employee();  
13)         emp.gen_Salary(25000, 20.0f);  
14)     }  
15) }
```

**Note:** If we compile the above code then compiler will provide the following deprecation message.

**Note:** Java uses or overrides a deprecated API"



## • @SuppressWarnings(--):

In Java applications, when we perform unchecked or unsafe operations then compiler will rise some warning messages. In this context, to remove the compiler generated warning messages we have to use `@SuppressWarnings("unchecked")` annotation.

EX:

```
1) import java.util.*;
2) class Bank {
3)     @SuppressWarnings("unchecked")
4)     public ArrayList listCustomers() {
5)         ArrayList al = new ArrayList();
6)         al.add("chaitu");
7)         al.add("Mahesh");
8)         al.add("Jr NTR");
9)         al.add("Pavan");
10)        return al;
11)    }
12)
13) class Test {
14)     public static void main(String args[]) {
15)         Bank b = new Bank();
16)         List l = b.listCustomers();
17)         System.out.println(l);
18)     }
19)}
```

## @FunctionalInterface:

- \* In Java, if we provide any interface without abstract methods then that interface is called as "Marker Interface".
- \* In Java, if we provide any interface with exactly one abstract method then that interface is called as "Functional Interface".
- \* To make any interface as Functional Interface and to allow exactly one abstract method in any interface then we have to use "`@FunctionalInterface`" annotation.
- \* NOTE: This annotation is a new annotation provided by JAVA8 version.

```
1) @FunctionalInterface
2) interface Loan {
3)     void getLoan();
4)
5) class GoldLoan implements Loan {
6)     public void getLoan() {
7)         System.out.println("GoldLoan");
8)     }
9) }
```



```
10) class Test {  
11)     public static void main(String args[]) {  
12)         Loan l = new GoldLoan();  
13)         l.getLoan();  
14)     }  
15) }
```

## @Inherited:

In general all the annotations are not inheritable by default.

If we want to make/prepare any annotation as Inheritable annotation then we have to declare that annotation as Inheritable annotation then we have to declare that annotation with `@Inherited` annotation.

EX:

```
1) @interface Persistable  
2) {  
3) }  
4) @Persistable  
5) class Employee  
6) {  
7) }  
8) class Manager extends Employee{  
9) }
```

In the above example, only Employee class objects are Persistable i.e eligible to store in database.

```
1) @Inherited  
2) @interface Persistable  
3) {  
4) }  
5) @Persistable  
6) class Employee {  
7) }  
8) class Manager extends Employee {  
9) }
```

**NOTE:** In the above example, both Employee class objects and manager class objects are Persistable i.e eligible to store in database.



## • @Documented:

- In Java, by default all the annotations are not documentable.
- In Java applications, if we want to make any annotation as documentable annotation then we have to prepare the respective annotation with `@Documented` annotation.

EX:

```
1) @interface Persistable
2) {
3) }
4) @Persistable
5) class Employee
6) {
7) }
8) javadoc Employee.java
```

If we prepare html documentation for Employee class by using "javadoc" tool then  
`@Persistable` annotation is not listed in the html documentation.

```
1) @Documented
2) @interface Persistable
3) {
4) }
5) @Persistable
6) class Employee
7) {
8) }
```

If we prepare html documentation for Employee class by using "javadoc" tool then  
`@Persistable` annotation will be listed in the html documentation.

## • @Target:

The main intention of this annotation is to define a list of target elements to which we are applying the respective annotation.

**Syntax:** `@Target(--list of constants from ElementType enum--)`

Where `ElementType` enum contains FIELD, CONSTRUCTOR, METHOD,  
TYPE [Classes, abstract classes, interfaces]

EX:

```
1) @Target(ElementType.TYPE, ElementType.FIELD, ElementType.METHOD)
2) @interface persistable
3) {
4) }
5) @Persistable
```



```
6) class Employee  
7) {  
8)     @Persistable  
9)     Account acc;  
10)  
11)    @Persistable  
12)    public Address getAddress(){  
13) }
```

## @Retention:

The main intention of the annotation is to define life time of the respective annotation in Java application.

**Syntax:** `@Retention(...a constant from RestentionPolicy enum...)`

Where RestentionPolicy enum contains the constants like SOURCE, CLASS, RUNTIME

```
1) @Retention(RetentionPolicy.RUNTIME)  
2) @interface Persistable  
3) {  
4) }  
5) @Persistable  
6) class Employee  
7) {  
8) }
```

## Custom Annotations:

These are the annotations defined by the developers as per their application requirements.

To use custom annotations in java applications, we have to use the following steps.

- **Declare User defined Annotation:**

### Bank.java

```
1) import java.lang.annotation.*;  
2) @Inherited  
3) @Documented  
4) @Target(ElementType.TYPE)  
5) @Retention(RetentionPolicy.RUNTIME)  
6) @interface Bank {  
7)     String name() default "ICICI Bank";  
8)     String branch() default "S R Nagar";  
9)     String phone() default "040-123456";  
10) }
```



## 11) Utilize User defined Annotations in Java Applications:

### Account.java

```
1) @Bank(name="Axis Bank",phone="040-987654")
2) public class Account {
3)     String accNo;
4)     String accName;
5)     String accType;
6)     public Account(String accNo, String accName, String accType) {
7)         this.accNo = accNo;
8)         this.accName = accName;
9)         this.accType = accType;
10)    }
11)    public void getAccountDetails() {
12)        System.out.println("Account Details");
13)        System.out.println("-----");
14)        System.out.println("Account Number:" +accNo);
15)        System.out.println("Account Name :" +accName);
16)        System.out.println("Account Type :" +accType);
17)    }
18} }
```

### Access the Data from User-Defined Annotation in Main Application:

- If the Annotation is Class Level Annotation then use the following Steps
  - Get `java.lang.Class` object of the respective class
  - Get Annotation object by using `getAnnotation(Class c)` method
- If the annotation is Field level annotation then use the following steps:
  - Get `java.lang.Class` object of the respective class
  - Get `java.lang.reflect.Field` class object of the respective variable from `java.lang.Class` by using  `getField(String field_name)` method
  - Get Annotation object by using `getAnnotation(Class c)` method from `java.lang.reflect.Field` class.



- **If the Annotation is Method Level Annotation then use the following Steps**
  - Get `java.lang.Class` object of the respective class
  - Get `java.lang.reflect.Method` class object of the respective Method from `java.lang.Class` by using `getMethod(String method_name)` method
  - Get Annotation object by using `getAnnotation(Class c)` method from `java.lang.reflect.Method` class.

## MainApp.java

```
1) import java.lang.annotation.*;
2) import java.lang.reflect.*;
3) public class MainApp {
4)     public static void main(String args[]) throws Exception {
5)         Account acc = new Account("abc123","Durga","Hyd");
6)         acc.getAccountDetails();
7)         System.out.println();
8)         Class c = acc.getClass();
9)         Annotation ann = c.getAnnotation(Bank.class);
10)        Bank b = (Bank)ann;
11)        System.out.println("Bank Details");
12)        System.out.println("-----");
13)        System.out.println("Bank Name :" +b.name());
14)        System.out.println("Branch Name :" +b.branch());
15)        System.out.println("Phone " +b.phone());
16)    }
17} }
```

**NOTE:** class Level Annotation

## Method Level Annotation

### Course.java

```
1) import java.lang.annotation.*;
2) @Inherited
3) @Documented
4) @Target(ElementType.METHOD)
5) @Retention(RetentionPolicy.RUNTIME)
6) @interface Course {
7)     String cid() default "C-111";
8)     String cname() default "Java";
9)     int ccost() default 5000;
10} }
```



## Student.java

```
1) public class Student {  
2)     String sid;  
3)     String sname;  
4)     String saddr;  
5)     public Student(String sid, String sname, String saddr) {  
6)         this.sid = sid;  
7)         this.sname = sname;  
8)         this.saddr = saddr;  
9)     }  
10)    @Course(cid = "C-222", cname = ".NET")  
11)    public void getStudentDetails() {  
12)        System.out.println("Student Details");  
13)        System.out.println("-----");  
14)        System.out.println("Student Id :" + sid);  
15)        System.out.println("Student Name :" + sname);  
16)        System.out.println("Student Address:" + saddr);  
17)    }  
18) }
```

## ClientApp.java

```
1) import java.lang.annotation.*;  
2) import java.lang.reflect.*;  
3) public class ClientApp {  
4)     public static void main(String args[]) throws Exception {  
5)         Student std = new Student("S-111", "Durga", "Hyd");  
6)         std.getStudentDetails();  
7)         System.out.println();  
8)         Class cl = std.getClass();  
9)         Method m = cl.getMethod("getStudentDetails");  
10)        Annotation ann = m.getAnnotation(Course.class);  
11)        Course c = (Course)ann;  
12)        System.out.println("Course Details");  
13)        System.out.println("-----");  
14)        System.out.println("Course Id :" + c.cid());  
15)        System.out.println("Course Name :" + c.cname());  
16)        System.out.println("Course Cost :" + c.ccost());  
17)    }  
18) }
```



# Regular Expressions



---

Regular Expression is an expression which represents a group of Strings according to a particular pattern.

**EX:**

- We can write a Regular Expression to represent all valid mail ids.
- We can write a Regular Expression to represent all valid mobile numbers.

The main important application areas of Regular Expression are:

- To implement validation logic.
- To develop Pattern matching applications.
- To develop translators like compilers, interpreters etc.
- To develop digital circuits.
- To develop communication protocols like TCP/IP, UDP etc.

To represent and use Regular Expressions in Java applications, JAVA has provided predefined library in the form of a package like "java.util.regex" in JDK1.4 version.

To prepare and Use Regular Expressions in java applications we have to use the following actions.

- **Create java.util.regex.Pattern Class Object with a Regular Expression:**

- A Pattern object represents "compiled version of Regular Expression".
- To create Regular Expression in the form Pattern object we have to use the following method from java.util.regex.Pattern class.
- `public static Pattern compile(String reg_Ex)`

**EX:** `Pattern p=Pattern.compile("ab");`

- **Create java.util.regex.Matcher Class Object with a String which we want to compare with Regular Expression:**

- A Matcher object can be used to match character sequences against a Regular Expression.
- We can create a Matcher object by using `matcher()` method of Pattern class.  
`public Matcher matcher(String target);`

**EX:** `Matcher m=p.matcher("abbbabbaba");`



## • Get Matched Strings from Matcher Object:

To get details about the Matches identified in Matcher object we have to use the following methods.

- **public boolean find();**

It attempts to find next match and returns true if it is available otherwise returns false.

- **public int start();**

Returns the start index of the match.

- **public int end();**

Returns the offset (equalize) after the last character matched OR Returns the "end+1" index of the matched.

- **public String group();**

Returns the matched Pattern.

EX:

```
1) importjava.util.regex.*;
2) class RegularExpressionDemo {
3)     public static void main(String[] args) {
4)         int count = 0;
5)         Pattern p = Pattern.compile("ab");
6)         Matcher m = p.matcher("abbabbaba");
7)         while(m.find()) {
8)             count++;
9)             System.out.println(m.start()+"-----"+m.end()+"-----"+m.group());
10)        }
11)        System.out.println("The no of occurrences:"+count);
12)    }
13) }
```

OUTPUT:

```
0-----2-----ab
4-----6-----ab
7-----9-----ab
```

The no of occurrences: 3

To prepare Regular Expressions, we have to use the following elements.

- Character Classes
- Quantifiers



## • Character Classes:

Character classes are used to specify alphabets and digits in Regular Expressions.

- [abc] ----- Either 'a' or 'b' or 'c'
- [^abc] ----- Except 'a' and 'b' and 'c'
- [a-z] ----- Any lower case alphabet symbol
- [A-Z] ----- Any upper case alphabet symbol
- [a-zA-Z] ----- Any alphabet symbol
- [0-9] ----- Any digit from 0 to 9
- [a-zA-Z0-9] ----- Any alphanumeric character
- [^a-zA-Z0-9] ----- Any special character

## Predefined Character Classes:

\s----space character

\d----Any digit from 0 to 9 [0-9]

\w----Any word character [a-zA-Z0-9]

. ----Any character including special characters.

\S----any character except space character

\D----any character except digit

\W----any character except word character (special character)

### EX:

```
1) importjava.util.regex.*;
2) class RegularExpressionDemo {
3)     public static void main(String[] args) {
4)         Pattern p = Pattern.compile("x");
5)         Matcher m = p.matcher("a1b7@z#");
6)         while(m.find()) {
7)             System.out.println(m.start()+"-----"+m.group());
8)         }
9)     }
10) }
```

### EX:

```
1) importjava.util.regex.*;
2) class RegularExpressionDemo {
3)     public static void main(String[] args) {
4)         Pattern p = Pattern.compile("x");
5)         Matcher m = p.matcher("a1b7 @z#");
6)         while(m.find()) {
7)             System.out.println(m.start()+"-----"+m.group());
8)         }
9)     }
10) }
```



## Quantifiers:

Quantifiers can be used to specify no of characters to match.

- a ----- Exactly one 'a'
- a+ ----- At least one 'a'
- a\* ----- Any number of a's including zero number
- a? ----- At most one 'a'

### EX:

```
1) importjava.util.regex.*;
2) class RegularExpressionDemo {
3)     public static void main(String[] args) {
4)         Pattern p = Pattern.compile("x");
5)         Matcher m = p.matcher("abaabaaab");
6)         while(m.find()) {
7)             System.out.println(m.start()+"-----"+m.group());
8)         }
9)     }
10) }
```

## Pattern Class split() Method:

Pattern class contains split() method to split the given string against a regular expression.

### EX 1:

```
1) importjava.util.regex.*;
2) class RegularExpressionDemo {
3)     public static void main(String[] args) {
4)         Pattern p = Pattern.compile("\s");
5)         String[] s = p.split("Durga software solutions");
6)         for(String s1:s) {
7)             System.out.println(s1);
8)         }
9)     }
10) }
```

### EX 2:

```
1) importjava.util.regex.*;
2) class RegularExpressionDemo {
3)     public static void main(String[] args) {
4)         Pattern p = Pattern.compile("\\.| // (or) [.]");
5)         String[] s = p.split("www.dugrajobs.com");
6)         for(String s1:s) {
7)             System.out.println(s1);
8)         }
9)     }
10) }
```



```
9)    }
10) }
```

## String Class split() Method:

String class also contains split() method to split the given string against a regular expression.

### Example:

```
1) importjava.util.regex.*;
2) class RegularExpressionDemo {
3)     public static void main(String[] args) {
4)         String s = "www.durgajobs.com";
5)         String[] s1 = s.split("\\.");
6)         for(String s2:s1) {
7)             System.out.println(s2);
8)         }
9)     }
10) }
```

NOTE: String class split() method can take regular expression as argument where as pattern class split() method can take target string as the argument.

## StringTokenizer:

This class present in java.util package.

It is a specially designed class to perform string tokenization.

### EX 1:

```
1) importjava.util.*;
2) classRegularExpressionDemo
3) {
4)     public static void main(String[] args)
5)     {
6)         StringTokenizerst=new StringTokenizer("Durga Software Solutions");
7)         while(st.hasMoreTokens())
8)         {
9)             System.out.println(st.nextToken());
10)        }
11)    }
12) }
```

The default regular expression for the StringTokenizer is space.



## Example 2:

```
1) importjava.util.*;
2) classRegularExpressionDemo
3) {
4)     public static void main(String[] args)
5)     {
6)         StringTokenizerst=new StringTokenizer("1,99,988","","");
7)         while(st.hasMoreTokens())
8)         {
9)             System.out.println(st.nextToken());
10)        }
11)    }
12) }
```

## Requirement:

Write a regular expression to represent all valid identifiers in java language.

## Rules:

The allowed characters are:

- a to z, A to Z, 0 to 9, -, #
- The 1st character should be alphabet symbol only.
- The length of the identifier should be at least 2.

## Program:

```
1) importjava.util.regex.*;
2) class RegularExpressionDemo
3) {
4)     public static void main(String[] args)
5)     {
6)         Pattern p=Pattern.compile("[a-zA-Z][a-zA-Z0-9-#]+"); (or)
7)         Pattern p=Pattern.compile("[a-zA-Z][a-zA-Z0-9-#][a-zA-Z0-9-#]*");
8)         Matcher m=p.matcher(args[0]);
9)         if(m.find()&&m.group().equals(args[0]))
10)        {
11)            System.out.println("valid identifier");
12)        }
13)        else
14)        {
15)            System.out.println("invalid identifier");
16)        }
17)    }
18) }
```



## Requirement:

Write a regular expression to represent all mobile numbers.

- Should contain exactly 10 digits.
- The 1<sup>st</sup> digit should be 7 to 9.

## Program:

```
1) import java.util.regex.*;
2) class RegularExpressionDemo {
3)     public static void main(String[] args) {
4)         Pattern p = Pattern.compile(
5)             "[7-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9]");
6)         //Pattern p = Pattern.compile("[7-9][0-9]{9}");
7)         Matcher m = p.matcher(args[0]);
8)         if(m.find() && m.group(0).equals(args[0])) {
9)             System.out.println("valid number");
10)        }
11)        else {
12)            System.out.println("invalid number");
13)        }
14)    }
15) }
```

## Analysis:

10 digits mobile:

[7-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9][0-9] (or)  
[7-9][0-9]{9}

## OUTPUT:

```
E:\javaapps>javac RegularExpressionDemo.java
E:\javaapps>java RegularExpressionDemo 9989123456
Valid number
```

```
E:\javaapps>java RegularExpressionDemo 6989654321
Invalid number
10 digits (or) 11 digits:
(0?[7-9][0-9]{9})
```

## OUTPUT:

```
E:\javaapps>javac RegularExpressionDemo.java
```

```
E:\javaapps>java RegularExpressionDemo 9989123456
Valid number
```

```
E:\javaapps>java RegularExpressionDemo 09989123456
Valid number
```



```
E:\javaapps>java RegularExpressionDemo 919989123456
```

Invalid number

10 digits (Or) 11 digit (or) 12 digits:

(0|91)?[7-9][0-9]{9} (or)

(91)?(0?[7-9][0-9]{9})

```
E:\javaapps>javac RegularExpressionDemo.java
```

```
E:\javaapps>java RegularExpressionDemo 9989123456
```

Valid number

```
E:\javaapps>java RegularExpressionDemo 09989123456
```

Valid number

```
E:\javaapps>java RegularExpressionDemo 919989123456
```

Valid number

```
E:\javaapps>java RegularExpressionDemo 69989123456
```

Invalid number

### **Requirement:**

Write a regular expression to represent all Mail Ids.

### **Program:**

```
1) importjava.util.regex.*;
2) class RegularExpressionDemo
3) {
4)     public static void main(String[] args)
5)     {
6)         Pattern p=Pattern.compile(
7)             "[a-zA-Z][a-zA-Z0-9-.]*@[a-zA-Z0-9]+([.][a-zA-Z]+)+");
8)         Matcher m=p.matcher(args[0]);
9)         if(m.find()&&m.group().equals(args[0]))
10)        {
11)            System.out.println("valid mail id");
12)        }
13)        else
14)        {
15)            System.out.println("invalid mail id");
16)        }
17)    }
18} }
```

### **OUTPUT:**

```
E:\javaapps>javac RegularExpressionDemo.java
```

```
E:\javaapps>java RegularExpressionDemo sunmicrosystem@gmail.com
```

Valid mail id

```
E:\javaapps>java RegularExpressionDemo 999sunmicrosystem@gmail.com
```

Invalid mail id



---

```
E:\javaapps>java RegularExpressionDemo 999sunmicrosystem@gmail.co9
Invalid mail id
```

### Requirement:

Write a program to extract all valid mobile numbers from a file.

### Program:

```
1) import java.util.regex.*;
2) import java.io.*;
3) class RegularExpressionDemo
4) {
5)     public static void main(String[] args) throws IOException
6)     {
7)         PrintWriter out=new PrintWriter("output.txt");
8)         BufferedReader br=new BufferedReader(new FileReader("input.txt"));
9)         Pattern p=Pattern.compile("(0|91)?[7-9][0-9]{9}");
10)        String line=br.readLine();
11)        while(line!=null)
12)        {
13)            Matcher m=p.matcher(line);
14)            while(m.find())
15)            {
16)                out.println(m.group());
17)            }
18)            line=br.readLine();
19)        }
20)        out.flush();
21)    }
22} }
```

### Requirement:

Write a program to extract all Mail IDs from the File.

**NOTE:** In the above program replace mobile number regular expression with MAIL ID regular expression.

### Requirement:

Write a program to display all .txt file names present in E:\javaapps folder.



## Program:

```
1) import java.util.regex.*;
2) import java.io.*;
3) classRegularExpressionDemo
4) {
5)     public static void main(String[] args) throws IOException
6)     {
7)         int count=0;
8)         Pattern p=Pattern.compile("[a-zA-Z0-9-$.]+[.]txt");
9)         File f=new File("E:\\javaapps");
10)        String[] s=f.list();
11)        for(String s1:s)
12)        {
13)            Matcher m=p.matcher(s1);
14)            if(m.find()&&m.group().equals(s1))
15)            {
16)                count++;
17)                System.out.println(s1);
18)            }
19)        }
20)        System.out.println(count);
21)    }
22} }
```

## OUTPUT:

input.txt  
output.txt  
outut.txt  
3

Write a regular expressions to represent valid Gmail mail id's:

[a-zA-Z0-9][a-zA-Z0-9-.]\*@gmail[.]com



### Write a regular expressions to represent all Java language identifiers:

#### Rules:

- The length of the identifier should be at least two.
- The allowed characters are

a-z  
A-Z  
0-9  
#  
\$

- The first character should be lower case alphabet symbol k-z , and second character should be a digit divisible by 3  
 $[k-z][0369][a-zA-Z0-9#\$]^*$

### Write a regular expressions to represent all names starts with 'a'

$[aA][a-zA-Z]^*$

To represent all names starts with 'A' ends with 'K'

$[aA][a-zA-Z]^*[kK]$



# GARBAGE COLLECTION



- Introduction
- Approaches to make an Object Eligible for Garbage Collection
- Approaches for requesting garbage Collector to Run Garbage Collector
- Finalization
- Java Heap Memory and its Memory Areas
- Garbage Collection in Young Generation
- Garbage Collection in Old generation
- Types of Garbage Collectors for Old generation

## Introduction:

- In old languages like C++ programmer is responsible for both creation and destruction of objects. Usually programmer is taking very much care while creating object and neglect destruction of useless objects .Due to his negligence at certain point of time for creation of new object sufficient memory may not be available and entire application may be crashed due to memory problems.
- But in java programmer is responsible only for creation of new object and his not responsible for destruction of objects.
- Sun people provided one assistant which is always running in the background for destruction at useless objects. Due to this assistant the chance of failing java program is very rare because of memory problems.
- This assistant is nothing but garbage collector. Hence the main objective of GC is to destroy useless objects.

## The Ways To Make An Object Eligible For GC:

- Even though programmer is not responsible for destruction of objects but it is always a good programming practice to make an object eligible for GC if it is no longer required.
- An object is eligible for GC if and only if it does not have any references.

The following are various possible ways to make an object eligible for GC:

### • Nullifying the Reference Variable:

If an object is no longer required then we can make eligible for GC by assigning "null" to all its reference variables.

```
Student std1=new Student();
Student std2=new Student();
```

-----

-----

```
std1=null;
std2=null;
```

-----

-----



- **Reassign the Reference Variable:**

If an object is no longer required then reassign all its reference variables to some other objects then old object is by default eligible for GC.

```
Student std1 = new Student();
Student std2 = new Student();
---
---
std1 = new Student();
---
---
std2=std1;
---
---
```

- **Objects created inside a Method:**

Objects created inside a method are by default eligible for GC once method completes.

**EX 1:**

```
1) class Test {
2)     public static void main(String[] args) {
3)         m1();
4)     }
5)     static void m1() {
6)         Student std1 = new Student();
7)         Student std2 = new Student();
8)     }
9) }
```

**Island of Isolation:**

If objects are not having explicit references or the Objects are having internal references are eligible for GC.

```
1) class Test {
2)     Test t;
3)     public static void main(String[] args) {
4)         Test t1 = new Test();
5)         Test t2 = new Test();
6)         Test t3 = new Test();
7)
8)         t1.t = t2;
9)         t2.t = t3;
10)        t3.t = t1;
```



```
11)
12)     t1 = null; //No Object is eligible for GC.
13)     t2 = null; //No Object is eligible for GC.
14)     t3 = null; //All Objects are eligible GC, because, all Explicit references
15)             are nullified.
16) }
17} }
```

## The Methods for requesting JVM to Run GC:

- Once we made an object eligible for GC it may not be destroyed immediately by the GC. Whenever JVM runs GC then only object will be destroyed by the GC. But when exactly JVM runs GC we can't expect, it is vendor dependent.
- We can request JVM to run garbage collector programmatically, but whether JVM accept our request or not there is no guaranty. But most of the times JVM will accept our request.

The following are various ways for requesting JVM to run GC:

- By System Class:**

System class contains a static method GC for this purpose.

EX: System.gc();

- By Runtime class:**

A Java application can communicate with JVM by using Runtime object.

Runtime class is a singleton class present in java.lang. Package.

We can create Runtime object by using factory method getRuntime().

EX: Runtime r = Runtime.getRuntime();

Once we got Runtime object we can call the following methods on that object.

freeMemory(): returns the free memory present in the heap.

totalMemory(): returns total memory of the heap.

gc(): for requesting JVM to run gc.

EX:

```
1) importjava.util.Date;
2) class RuntimeDemo {
3)     public static void main(String args[]) {
4)         Runtime r = Runtime.getRuntime();
5)         System.out.println("total memory of the heap :" +r.totalMemory());
6)         System.out.println("free memory of the heap :" +r.freeMemory());
7)         for(int i=0; i<10000; i++) {
8)             Date d = new Date();
9)             d = null;
```



```
10)    }
11)    System.out.println("free memory of the heap :" + r.freeMemory());
12)    r.gc();
13)    System.out.println("free memory of the heap :" + r.freeMemory());
14) }
15) }
```

### OUTPUT:

Total memory of the heap: 5177344  
Free memory of the heap: 4994920  
Free memory of the heap: 4743408  
Free memory of the heap: 5049776

NOTE: Runtime class is a singleton class so not create the object to use constructor.

### Which of the following are valid ways for requesting JVM to Run GC?

System.gc(); (valid)  
Runtime.gc(); (invalid)  
(new Runtime).gc(); (invalid)  
Runtime.getRuntime().gc(); (valid)

NOTE: gc() method present in System class is static, where as it is instance method in Runtime class.

NOTE: Over Runtime class gc() method , System class gc() method is recommended to use.

NOTE: In java it is not possible to find size of an object and address of an object.

## Finalization:

- Just before destroying any object GC always calls finalize() method to perform cleanup activities.
- If the corresponding class contains finalize() method then it will be executed otherwise Object class finalize() method will be executed.  
which is declared as follows.  
`protected void finalize() throws Throwable`

### Case 1:

Just before destroying any object GC calls finalize() method on the object which is eligible for GC then the corresponding class finalize() method will be executed.

For Example if String object is eligible for GC then String class finalize() method is executed but not Test class finalize()method.



**EX:**

```
1) class Test {  
2)     public static void main(String args[]) {  
3)         String s = new String("Durga Software Solutions");  
4)         Test t = new Test();  
5)         s = null;  
6)         System.gc();  
7)         System.out.println("End of main.");  
8)     }  
9)     public void finalize() {  
10)         System.out.println("finalize() method is executed");  
11)     }  
12} }
```

**OUTPUT:** End of main.

In the above program String class finalize() method got executed. Which has empty implementation.

If we replace String object with Test object then Test class finalize() method will be executed .

The following program is an Example of this.

**EX:**

```
1) class Test {  
2)     public static void main(String args[]) {  
3)         String s = new String("Durga Software Solutions");  
4)         Test t = new Test();  
5)         t = null;  
6)         System.gc();  
7)         System.out.println("End of main.");  
8)     }  
9)     public void finalize() {  
10)         System.out.println("finalize() method is executed");  
11)     }  
12} }
```

**OUTPUT:** finalize() method is executed

End of main

**Case 2:**

We can call finalize() method explicitly then it will be executed just like a normal method call and object won't be destroyed. But before destroying any object GC always calls finalize() method.



## EX:

```
1) class Test {  
2)     public static void main(String args[]) {  
3)         Test t = new Test();  
4)         t.finalize();  
5)         t.finalize();  
6)         t = null;  
7)         System.gc();  
8)         System.out.println("End of main.");  
9)     }  
10)    public void finalize() {  
11)        System.out.println("finalize() method called");  
12)    }  
13) }
```

## OUTPUT:

finalize() method called.

finalize() method called.

finalize() method called.

End of main.

In the above program finalize() method got executed 3 times in that 2 times explicitly by the programmer and one time by the gc.

## Case 3:

finalize() method can be call either by the programmer or by the GC .

If the programmer calls explicitly finalize() method and while executing the finalize() method if an exception raised and uncaught then the program will be terminated abnormally.

If GC calls finalize() method and while executing the finalize()method if an exception raised and uncaught then JVM simply ignores that exception and the program will be terminated normally.

## EX:

```
1) class Test {  
2)     public static void main(String args[]) {  
3)         Test t = new Test();  
4)         //t.finalize();-----line(1)  
5)         t = null;  
6)         System.gc();  
7)         System.out.println("End of main.");  
8)     }  
9)     public void finalize() {  
10)        System.out.println("finalize() method called");  
11)        System.out.println(10/0);
```



12) }

If we are not comment line1 then programmer calling finalize() method explicitly and while executing the finalize()method ArithmeticException raised which is uncaught hence the program terminated abnormally.

If we are comment line1 then GC calls finalize() method and JVM ignores ArithmeticException and program will be terminated normally.

## Which of the following is true?

While executing finalize() method JVM ignores every exception(invalid).

While executing finalize() method JVM ignores only uncaught exception(valid).

### Case 4:

On any object GC calls finalize() method only once.

#### EX:

```
1) class FinalizeDemo {  
2)     static FinalizeDemo s;  
3)     public static void main(String args[]) throws Exception {  
4)         FinalizeDemo f = new FinalizeDemo();  
5)         System.out.println(f.hashCode());  
6)         f = null;  
7)         System.gc();  
8)         Thread.sleep(5000);  
9)         System.out.println(s.hashCode());  
10)        s = null;  
11)        System.gc();  
12)        Thread.sleep(5000);  
13)        System.out.println("end of main method");  
14)    }  
15)    public void finalize() {  
16)        System.out.println("finalize method called");  
17)        s = this;  
18)    }  
19) }
```

#### OUTPUT:

```
D:\Enum>java FinalizeDemo  
4072869  
finalize method called  
4072869  
End of main method
```



## NOTE:

The behavior of the GC is vendor dependent and varied from JVM to JVM hence we can't expect exact answer for the following.

- What is the algorithm followed by GC.
- Exactly at what time JVM runs GC.
- In which order GC identifies the eligible objects.
- In which order GC destroys the object etc.
- Whether GC destroys all eligible objects or not.

Whenever the program runs with low memory then the JVM runs GC, but we can't except exactly at what time.

## Memory Leaks:

In Java applications, if any object is not eligible for Garbage Collection and not utilized in java applications then that Objects are called as "Memory Leaks".

## Heap Memory Structure:

Heap Memory is mainly divided into the following two parts.

- Young Generation
  - Eden Space
  - Survivor Space
    - Survivor Space
    - Survivor Space
- Old Generation

When we create object newly, that object will come to Young Generation, in Young Generation, that Object will come to Eden Space, at Eden space Minor Garbage Collection will be performed, if any object is dereference object then Minor Garbage Collection will remove that object. In Eden Space, if any object is still live then Garbage Collector will move that live objects to Survivor space, in Survivor space , initial objects will come to S0 survivor space, there also Minor Garbage Collection will be performed, where if any object is dereferenced then Garbage Collector will remove that objects and the remaining objects are moved to S1 Survivor space, there again Minor Garbage Collection is going on, still any object is live then that object will be moved to Old Generation. In Old Generation, Major Garbage Collection will be performed.

In Old Generation, we will perform Garbage Collection by using either of the following Garbage Collectors.

- Serial Garbage Collector
- Parallel Garbage Collector
- Parallel Old Garbage Collector
- Concurrent Mark and Sweep [CMS] Garbage Collector



- Garbage First [G1] Garbage Collector

## • **Serial Garbage Collector:**

This Garbage Collector will follow the following algorithm

- Mark the Surviving objects in Old Generation.
- Keep all marked objects at front end of the Heap and keep all the unmarked Objects at another end [Sweep].
- Free the memory for unmarked objects [Compact]

It will perform Garbage Collection by using Single Thread.

- While performing garbage Collection, it will pause all application threads which are running in java applications.
- It is suitable for Single Thread Model.
- It is not suitable for multi threaded applications.
- It is suitable for Standalone applications.
- It is not suitable for Server side applications.
- To run this Garbage Collector, it will take less memory.
- To activate this Garbage Collector we have to use the following Command.  
D:\java9>java -XX:+UseSerialGC Test

## • **Parallel Garbage Collector / Throughput Garbage Collector:**

- It will use the following algorithm to perform Garbage Collection
  - Mark the Surviving objects in Old Generation.
  - Keep all marked objects at front end of the Heap and keep all the unmarked Objects at another end [Sweep].
  - Free the memory for unmarked objects [Compact]
- It is default Garbage Collector in JVM.
- It will use more than one thread to perform garbage Collection.
- It will be used in Multi Threaded Environment.
- It is suitable in Server side programming.
- It will pause all the threads while performing Garbage Collection.
- It requires more memory to perform garbage Collection when compared with Serial Garbage Collector.
- To activate this Garbage Collector we will use the following command.  
D:\java9>java -XX:+UseParallelGC Test

## • **Parallel Old Garbage Collector:**

- It was introduced in JDK5.0 version.
- It is same as Parallel Garbage Collector, but, it will use "Mark-Summary-Compact" Algorithm.
  - Mark Survivor objects in Old Generation.
  - Identifies all Survivor objects from the areas where Garbage Collection was performed previously.



- Delete Unmarked Objects, that is, Compact.
- To activate this Garbage Collector we have to use the following command.  
D:\java9>java -XX:+UseParallelOldGC Test
- **Concurrent Mark And Sweep [CMS] Garbage Collector:**
- It allows multiple threads to perform garbage Collection.
- It will not freeze all the application threads while performing Garbage Collection except in the following two situations.
  - While performing marking the survivor objects.
  - Any changes in the heap memory while performing garbage Collection
- It will provide very good performance when compared with Parallel Garbage Collector.
- It is more complicated Garbage Collector and it needs more system Memory.
- To activate this Garbage Collector we will use the following command.  
D:\java9>java -XX:+UseConcMarkSweepGC Test
- **Garbage First [G1] Garbage Collector:**
- It was introduced in JAVA7 version.
- In this mechanism, it will divide heap memory into no of regions, each region contains no of Grids, where each grid contains Objects, Here Garbage Collection will be performed over all the Grids that is Objects , if any useless object is identified then it will be removed.
- This mechanism is very simple and fast.
- To activate this mechanism we have to use the following command.  
D:\java9>java -XX:+UseG1GC Test

## Garbage Collection Optimization Options:

-Xms ----> Initial Heap Size.  
-Xmx ----> Max Heap Size  
-Xmn ----> Size of Young generation  
-XX:PermSize ---> Initial Permanent Generation Size  
-XX:MaxPermSize ---> Maximum Permanent Generation Size

EX:

D:\java9>java -Xmx12m -Xms3m -Xmn1m -XX:PermSize=20m  
-XX:MaxPermSize=20m -XX:+UseSerialGC Test



# JVM



## Virtual Machine: JVM

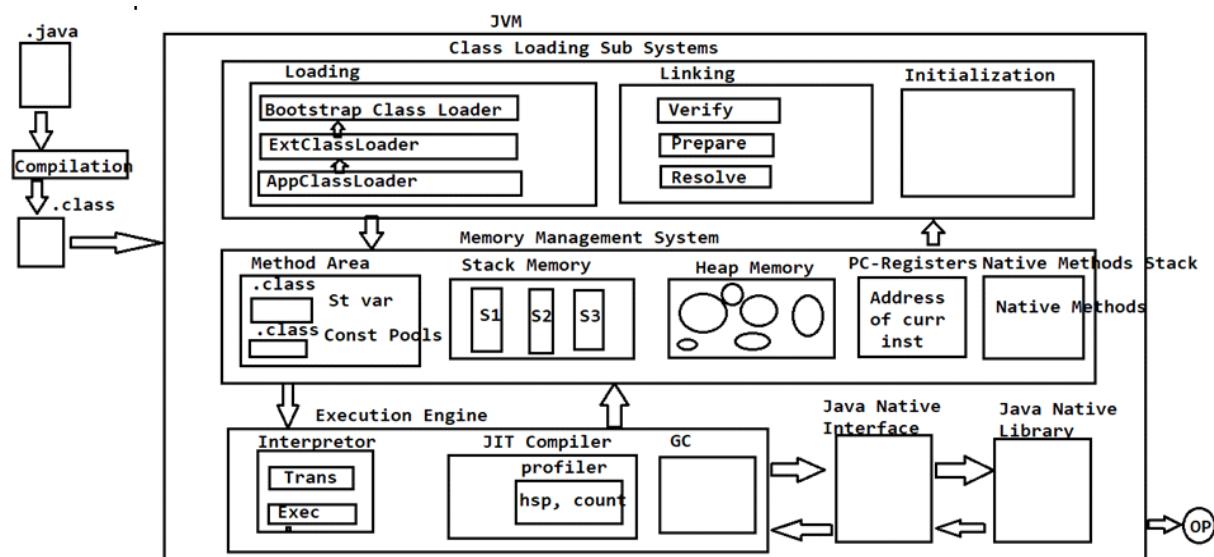
It is a Software Simulation of a Machine which can Perform Operations Like a Physical Machine.

EX:

- JVM Acts as Runtime Engine to Run Java Applications
- PVM (Parrot VM) Acts as Runtime Engine to Run Scripting Languages Like PEARL.
- CLR (Common Language Runtime) Acts as Runtime Engine to Run .Net Based Application
- KVM (Kernel Based Virtual Machine) for Linux Systems

## JVM

- JVM is the Part of JRE.
- JVM is Responsible to Load and Run Java Applications.
- JVM Runs Java Byte Code by creating 5 Identical Runtime Areas to execute Class Members.
  - Class Loader Sub System
  - Memory Management System
  - Execution Engine
  - PC-Registers
  - Native Methods Stack





## ClassLoader Sub System:

ClassLoader Sub System is Responsible for the following 3 Activities.

- Loading
- Linking
- Initialization

### • Loading:

- Loading Means Reading Class Files and Store Corresponding Binary Data in Method Area.
- For Each Class File JVM will Store the following Information in Method Area.
  - Fully Qualified Name of the Loaded Class OR Interface OR enum.
  - Fully Qualified Name of its Immediate Parent Class OR Interface OR enum.
  - Whether .class File is related to Class OR Interface OR enum.
  - The Modifiers Information
  - Variable OR Fields Information
  - Method Information
  - Constant Pool Information and so on.

After loading .class File Immediately JVM will Creates an Object of the Type class Class to Represent Class Level Binary Information on the Heap Memory.

The Class Object used by Programmer to got Class Level Information Like Fully Qualified Name of the Class, Parent Name, Method and Variable Information Etc.

**NOTE:** For Every Loader Time Only One Class Object will be Created Even though we are using Class Multiple Times in Our Application.

### • Linking:

Linking Consists of 3 Activities

- Verification
- Preparation
- Resolution

### • Verification:

- It is the Process of ensuring that Binary Representation of a Class is structurally Correct OR Not.
- That is JVM will Check whether .class File generated by Valid Compiler OR Not and whether .class File is Properly Formatted OR Not.
- Internally Byte Code Verifier which is Part of ClassLoader Sub System is Responsible for this Activity.
- If Verification Fails then we will get Runtime Exception Saying `java.lang.VerifyError`.



- **Preparation:**

In this Phase JVM will Allocate Memory for the Class Level Static Variables and Assign Default Values (But Not Original Values Assign to the Variable).

NOTE: Original Values will be assigned in Initialization Phase.

- **Resolution:**

- It is the Process of Replaced Symbolic References used by the Loaded Type with Original References.
- Symbolic References are Resolved into Direct References by searching through Method Area to Locate the Referenced Entity.

- **Initialization:**

In this Phase All Static Variables Assignments with Original Values and Static Block Execution will be performed from Parent Class to Child Class.

NOTE: While Loading, Linking and Initialization if any Error Occurs then we will get Runtime Exception Saying LinkageError.

## Types of ClassLoaders:

Every ClassLoader Sub System contains the following 3 ClassLoaders.

- Bootstrap ClassLoader OR Primordial ClassLoader
- Extension ClassLoader
- Application ClassLoader OR System ClassLoader

- **Bootstrap ClassLoader:**

- This ClassLoader is Responsible for loading 4 Java API Classes.
- That is the Classes Present in rt.jar (runtime.jar).  
Location: %JAVA\_HOME%\jre\lib\rt.jar
- This Location is Called Bootstrap Class Path.
- That is Bootstrap ClassLoader is Responsible to Load Classes from Bootstrap Class Path.  
Bootstrap ClassLoader is by Default Available with the JVM.
- It is implemented in Native Languages Like C and C++.

- **Extension ClassLoader:**

- It is the Child of Bootstrap ClassLoader.
- This ClassLoader is Responsible to Load Classes from Extension Class Path.  
Location: %JAVA\_HOME%\jre\lib\ext
- This ClassLoader is implemented in Java and the corresponding .class File Name is sun.misc.Launcher\$ExtClassLoader.class

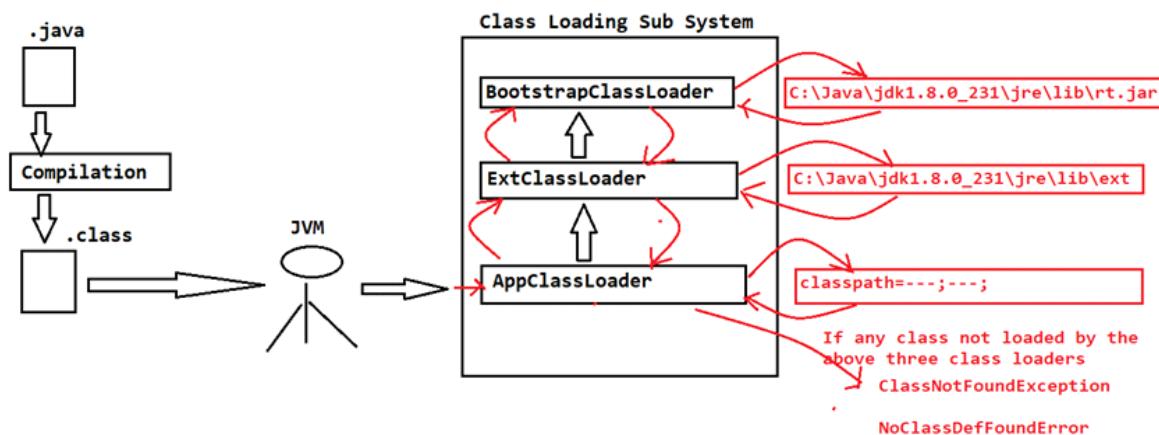


## • Application ClassLoader OR System ClassLoader:

- It is the Child of Extension ClassLoader.
- This ClassLoader is Responsible to Load Classes from Application Class Path (Current Working Directory).
- It Internally Uses Environment Variable Class Path.
- Application ClassLoader is implemented in Java and the corresponding .class File Name is sun.misc.Launcher\$AppClassLoader.class

## How ClassLoader will Work?

- ClassLoader follows Delegation Hierarchy Principle.
- Whenever JVM Come Across a Particular Class 1st it will Check whether the corresponding Class is Already Loaded OR Not.
- If it is Already Loaded in Method Area then JVM will Use that Loaded Class.
- If it is Not Already Loaded then JVM Requests ClassLoader Sub System to Load that Particular Class.
- Then ClassLoader Sub System Handovers the Request to Application ClassLoader.
- Application ClassLoader Delegates that Request to Extension ClassLoader and ExtensionClassLoader in-turn Delegates that Request to Bootstrap ClassLoader.
- Bootstrap ClassLoader Searches in Bootstrap Class Path for the required .class File (jdk/jre/lib)
- If the required .class is Available, then it will be Loaded. Otherwise Bootstrap ClassLoader Delegates that Request to Extension ClassLoader.
- Extension ClassLoader will Search in Extension Class Path (jdk/jre/lib/ext). If the required .class File is Available then it will be Loaded, Otherwise it Delegates that Request to Application ClassLoader.
- Application ClassLoader will Search in Application Class Path (Current Working Directory). If the specified .class is Already Available, then it will be Loaded Otherwise we will get Runtime Exception Saying ClassNotFoundException OR NoClassDefFoundError.





## Memory Management System:

- While Loading and Running a Java Program JVM required Memory to Store Several Things Like Byte Code, Objects, Variables, Etc.
- Total JVM Memory organized in the following 5 Categories:
  - Method Area
  - Heap Area OR Heap Memory
  - Java Stacks Area
  - PC Registers Area
  - Native Method Stacks Area

### **Method Area:**

- Method Area will be Created at the Time of JVM Start - Up.
- It will be Shared by All Threads (Global Memory).
- This Memory Area Need Not be Continuous.
- Method area shows runtime constant pool.
- Total Class Level Binary Information including Static Variables Stored in Method Area.

### **Heap Area:**

- Programmer Point of View Heap Area is Consider as Important Memory Area.
- Heap Area will be Created at the Time of JVM Start - Up.
- Heap Area can be accessed by All Threads (Global OR Sharable Memory).
- Heap Area Need not be Continuous.
- All Objects and corresponding Instance Variables will be stored in the Heap Area.
- Every Array in Java is an Object and Hence Arrays also will be stored in Heap Memory Only.
- We are able to get Heap memory calculations by using `java.lang.Runtime` class.
- Runtime Class Present in `java.lang` Package and it is a Singleton Class.
- We can Create Runtime Object by using  
`Runtime r = Runtime.getRuntime();`
- Once we got Runtime Object we can call the following Methods on that Object.
  - **maxMemory():** Returns Number of Bytes of Max Memory allocated to the Heap.
  - **totalMemory():** Returns Number of Bytes of Total (Initial) Memory allocated to the Heap.
  - **freeMemory():** Returns Number of Bytes of Free Memory Present in Heap.



**EX:**

```
1) class A
2) {
3) }
4) class Test
5) {
6)     public static void main(String[] args) throws Exception
7)     {
8)         A a1=new A();
9)         A a2=new A();
10)        A a3=new A();
11)
12)        Runtime rt=Runtime.getRuntime();
13)        System.out.println(rt.maxMemory());
14)        System.out.println(rt.totalMemory());
15)        System.out.println(rt.freeMemory());
16)        System.out.println(rt.totalMemory()-rt.freeMemory());
17)    }
18} }
```

## **Stack Memory:**

- ☺ For Every Thread JVM will Create a Separate Runtime Stack.
- ☺ Runtime Stack will be Created Automatically at the Time of Thread Creation.
- ☺ All Method Calls and corresponding Local Variables, Intermediate Results will be stored in the Stack.
- ☺ For Every Method Call a Separate Entry will be Added to the Stack and that Entry is Called Stack Frame OR Activation Record.
- ☺ After completing that Method Call the corresponding Entry from the Stack will be Removed.
- ☺ After completing All Method Calls, Just Before terminating the Thread, the Runtime Stack will be destroyed by the JVM.
- ☺ The Data stored in the Stack can be accessed by Only the corresponding Thread and it is Not Available to Other Threads.

## **PC (Program Counter) Registers Area:**

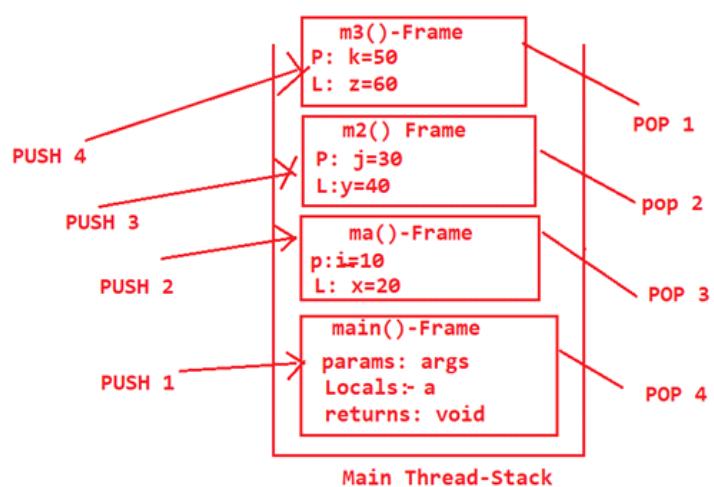
- ☺ For Every Thread a Separate PC Register will be Created at the Time of Thread Creation.
- ☺ PC Registers contains Address of Current executing Instruction.
- ☺ Once Instruction Execution Completes Automatically PC Register will be incremented to Hold Address of Next Instruction.



## **Native Method Stacks:**

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

```
class A
{
    void m1(int i){
        int x = 20;
        m2(30);
    }
    void m2(int j){
        int y = 40;
        m3(50);
    }
    void m3(int k){
        int z = 60;
    }
}
class Test{ Main Thread
    public static void main(S[] args){
        A a = new A();
        a.m1();
    }
}
```



### NOTE:

Method Area, Heap Area and Stack Area are considered as Major Memory Areas with Respect to Programmers Point of View.

Method Area and Heap Area are for JVM. Whereas Stack Area, PC Registers Area and Native Method Stack Area are for Thread. That is

- One Separate Heap for Every JVM
- One Separate Method Area for Every JVM
- One Separate Stack for Every Thread
- One Separate PC Register for Every Thread
- One Separate Native Method Stack for Every Thread

Static Variables will be stored in Method Area whereas Instance Variables will be stored in Heap Area and Local Variables will be stored in Stack Area.

## **Execution Engine:**

- This is the Central Component of JVM.
- Execution Engine is Responsible to Execute Java Class Files.
- Execution Engine contains 2 Components for executing Java Classes.
  - Interpreter
  - JIT Compiler



- **Interpreter:**

- It is Responsible to Read Byte Code and Interpret (Convert) into Machine Code (Native Code) and Execute that Machine Code Line by Line.
- The Problem with Interpreter is it Interprets Every Time Even the Same Method Multiple Times. Which Reduces Performance of the System.
- To Overcome this Problem SUN People Introduced JIT Compilers in 1.1 Version.

- **JIT Compiler:**

- The Main Purpose of JIT Compiler is to Improve Performance.
- Internally JIT Compiler Maintains a Separate Count for Every Method whenever JVM Come Across any Method Call.
- First that Method will be interpreted normally by the Interpreter and JIT Compiler Increments the corresponding Count Variable.
- This Process will be continued for Every Method.
- Once if any Method Count Reaches Threshold (The Starting Point for a New State) Value, then JIT Compiler Identifies that Method Repeatedly used Method (HOT SPOT).
- Immediately JIT Compiler Compiles that Method and Generates the corresponding Native Code. Next Time JVM Come Across that Method Call then JVM Directly Use Native Code and Executes it Instead of interpreting Once Again. So that Performance of the System will be Improved.
- The Threshold Count Value varied from JVM to JVM.
- Profiler which is the Part of JIT Compiler is Responsible to Identify HOT SPOTS.

**NOTE:**

- JVM Interprets Total Program Line by Line at least Once.
- JIT Compilation is Applicable Only for Repeatedly invoked Methods. But Not for Every Method.

## **Java Native Interface (JNI):**

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

## **Java Native Library:**

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



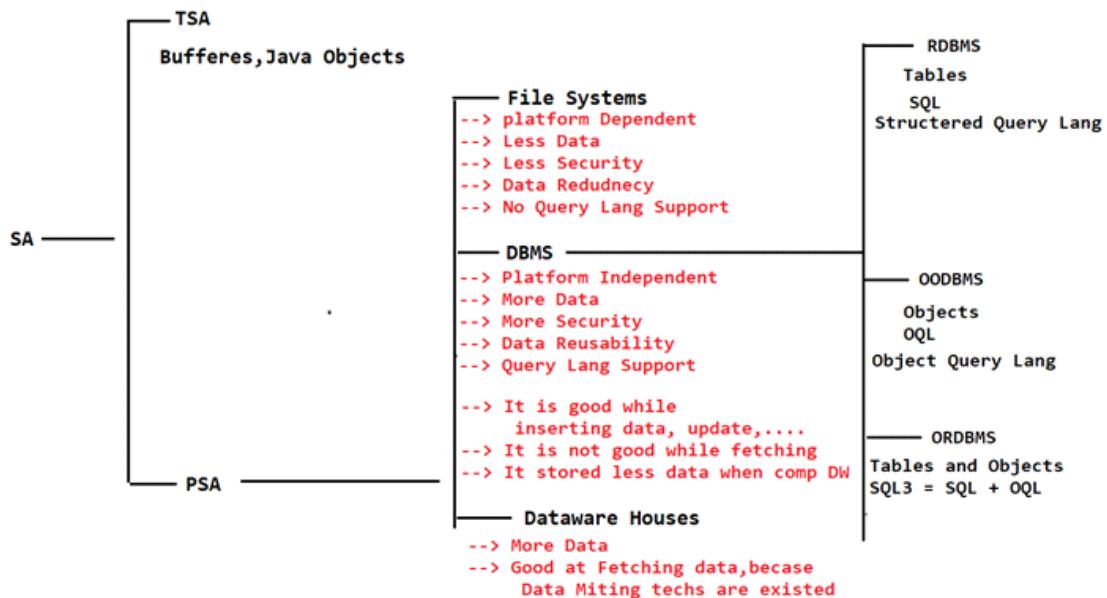
# JDBC



## Storage Areas:

As part of the Enterprise Application development it is essential to manage the organizations data like Employee Details, Customer Details, Products Details..etc

To manage the above specified data in enterprise applications we have to use storage areas (Memory elements). There are two types of Storage areas.



- **Temporary Storage Areas:**

These are the memory elements, which will store the data temporarily

Eg: Buffers, Java Objects

- **Permanent Storage Objects:**

These are the memory elements which will store data permanently.

Eg: FileSystems, DBMS, DataWareHouses.

## File Systems:

It is a System, it will be provided by the local operating System.

Due to the above reason File Systems are not suitable for platform independent technologies like JAVA.

- File Systems are able to store less volume of the data.
- File Systems are able to provide less security.
- File Systems may increases data Redundancy.



In case of File Systems Query Language support is not available. So that all the database operations are complex.

## **DBMS:**

- Database Management System is very good compare to file System but still it able to store less data when compared to DataWareHouses.
- DBMS is very good at the time of storing the data but which is not having Fast Retrieval Mechanisms.

## **DataWareHouses:**

When Compared to File Systems and DBMS it is able to store large and large volumes of data.

Data ware houses having fast retrieval mechanisms in the form of data mining techniques.

## **Q) What is the difference between Database and Database Management System?**

- DataBase is a memory element to store the data.
- Database Management System is a Software System, it can be used to manage the data by storing it on database and retrieving it from Database.
- Database is a collection of interrelated data as a single unit.
- DBMS is a collection of interrelated data and a set of programs to access the data.

There are three types of DBMS:

- RDMS (Relational Database Management Systems)
- OODBMS (Object Oriented Database Management Systems)
- ORDBMS (Object Relational Database Management Systems)

### **• Relational Database Management Systems:**

- It is a DBMS, it can be used to represent the data in the form of tables.
- This DBMS will use SQL3 as a Query Language to perform Database Operations.

### **• Object Oriented Database Management System:**

- It is Database Management System, It will represents the data in the form of Objects.
- This database management system will require OQL (Object Query Language) as Query language to perform database operations.



## **Object Relational Database Management System:**

- It is a Database Management System, it will represent some part of data in the form of Tables and some other part of the data in the form of objects
- This Database Management System will require SQL3 as Query Language to perform database operations.

Where SQL3 is the combination of SQL2 and OQL

$$\text{SQL3} = \text{SQL2+OQL}$$

## **Query Processing System:**

When we submit an SQL Query to the Database then Database Engine will perform the following Steps.

### **Step 1: Query Tokenization**

This Phase will take SQL Query as an Input, divided into number of tokens and Generate Stream of tokens as an output.

### **Step 2: Query Processing**

This phase will take Stream of tokens as an Input, constructs Query Tree with the Tokens, if Query

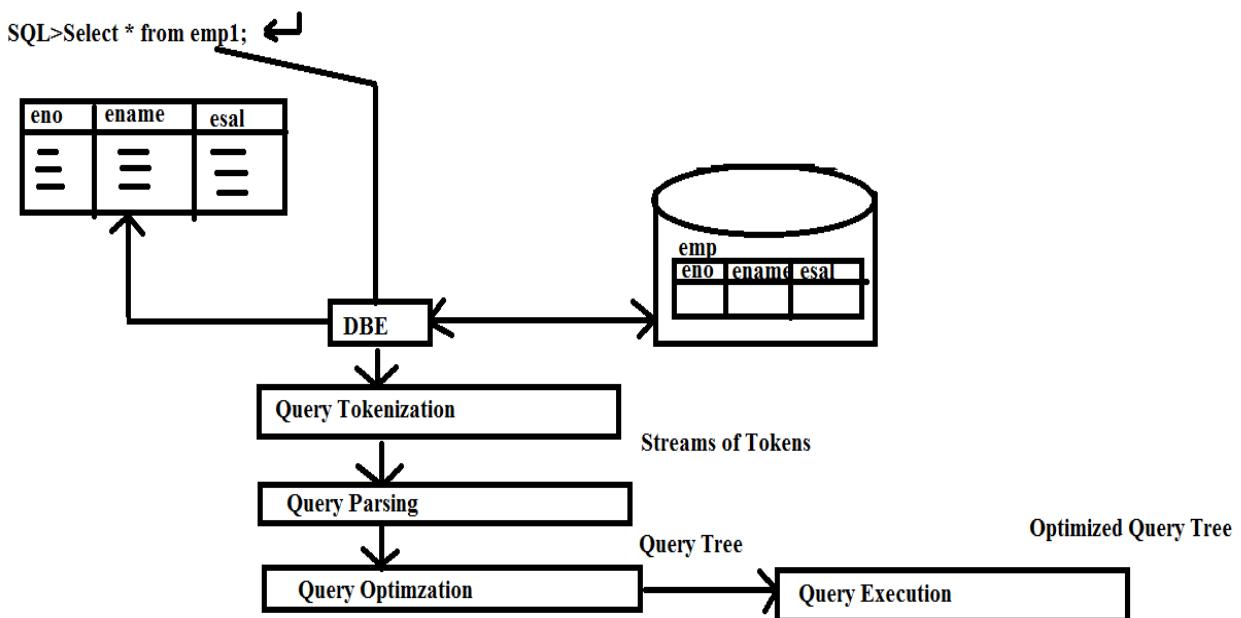
Tree Success then no Syntax error is available in the provided SQL Query. If Query Tree is not Success then there are some syntax errors in the provided SQL Query.

### **Step 3: Query Optimization**

The main purpose of Query Optimization phase is to perform optimization on Query Tree in order to reduce execution time and to optimize memory utilization.

### **Step 4: Query Execution**

This phase will take optimized Query Tree as an input and execute the Query by using interpreters.



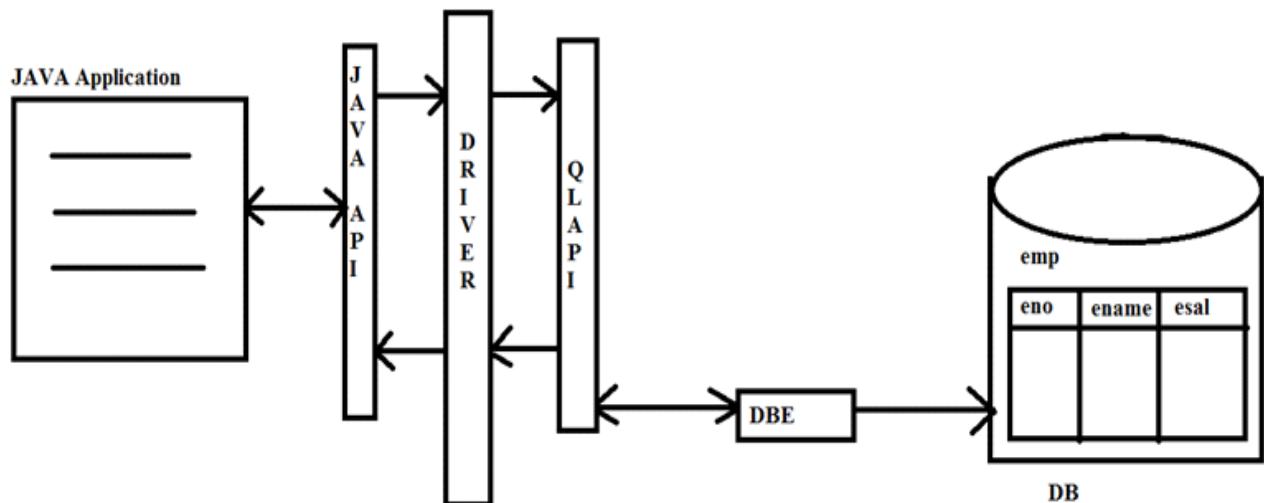
## JDBC (Java DataBase Connectivity):

- The process of interacting with the database from Java Applications is called as JDBC.
- JDBC is an API, which will provide very good predefined library to connect with database from JAVA Applications in order to perform the basic database operations
- In case of JDBC Applications we will define the database logic and Java application and we will send a Java represented database logic to Database Engine. But database engine is unable to execute the Java represented database logic, it should required the database logic in Query Language Representations.
- In the above context, to execute JDBC applications we should require a conversion mechanism to convert the database logic from Java representations to Query language representations and from Query language representations to Java representations.
- In the above situation the required conversion mechanisms are available in the form of software called as "Driver".



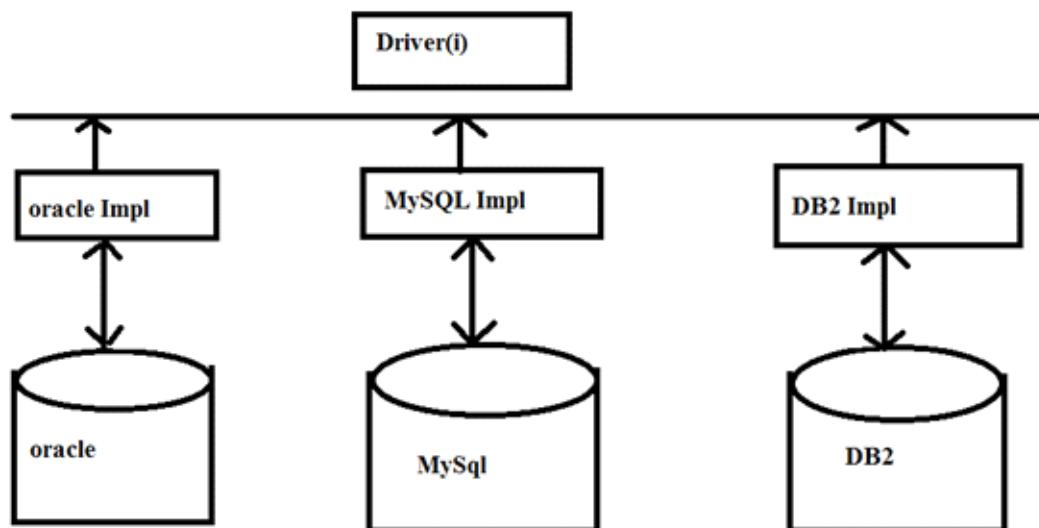
## Driver:

Driver is an interface existed between Java application and database to map Java API calls to Query language API calls and Query language API calls to Java API calls.



To provide driver as a product Sun Micro Systems has provided Driver as an interface and Sun Micro Systems. Lets the database vendors to provide implementation classes to the driver interface as part of their database software's.

If we want to use Drivers in JDBC applications then we have to get Driver implementation from the respective database software's.



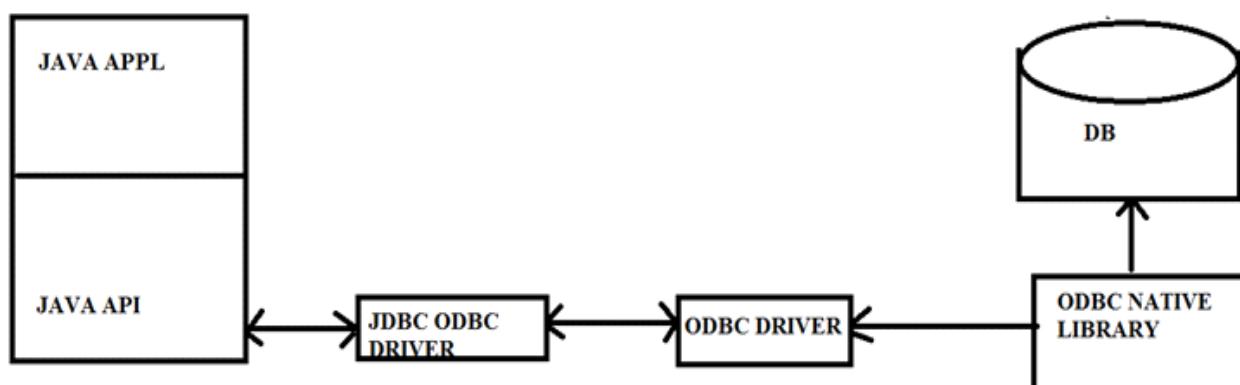


There are 180+ numbers of drivers but all these drivers could be classified into the following Four types.

- Type 1
- Type 2
- Type 3
- Type 4

## • Type 1 Driver:

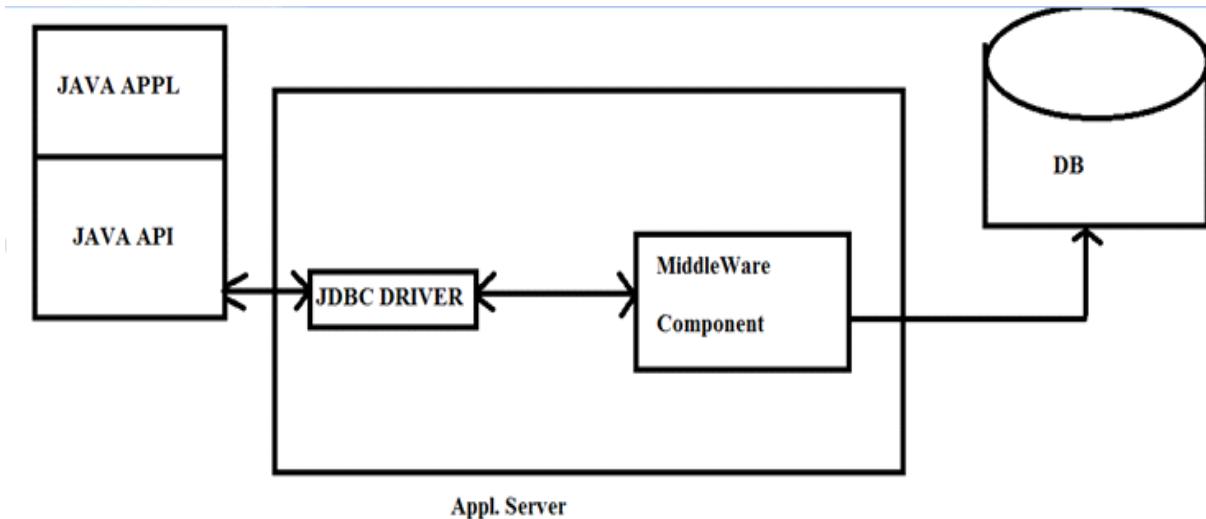
- Type 1 Driver is also called as JDBC-ODBC Driver and Bridge Driver.
- JDBC-ODBC Driver is a driver provided by Sun Micro Systems as an Implementation to Driver Interface.
- Sun Micro Systems has provided JDBC-ODBC Driver with the inter dependent on the Microsoft's product ODBC Driver.
- ODBC Driver is a Open Specification, it will provide very good environment to interact with any type of database from JDBC-ODBC Driver.
- If we want to use JDBC-ODBC Driver in our JDBC Applications first we have to install the MicroSoft Product ODBC Driver native library.
- To interact with the database from Java Application if we use JDBC-ODBC Driver then we should require two types conversions so that JDBC-ODBC Driver is Slower Driver.
- JDBC-ODBC Driver is highly recommended for standalone applications, it is not suitable for web applications, distributed applications and so on.
- JDBC-ODBC Driver is suggestible for Simple JDBC applications, not for complex JDBC applications.
- The portability of the JDBC-ODBC Driver is very less.





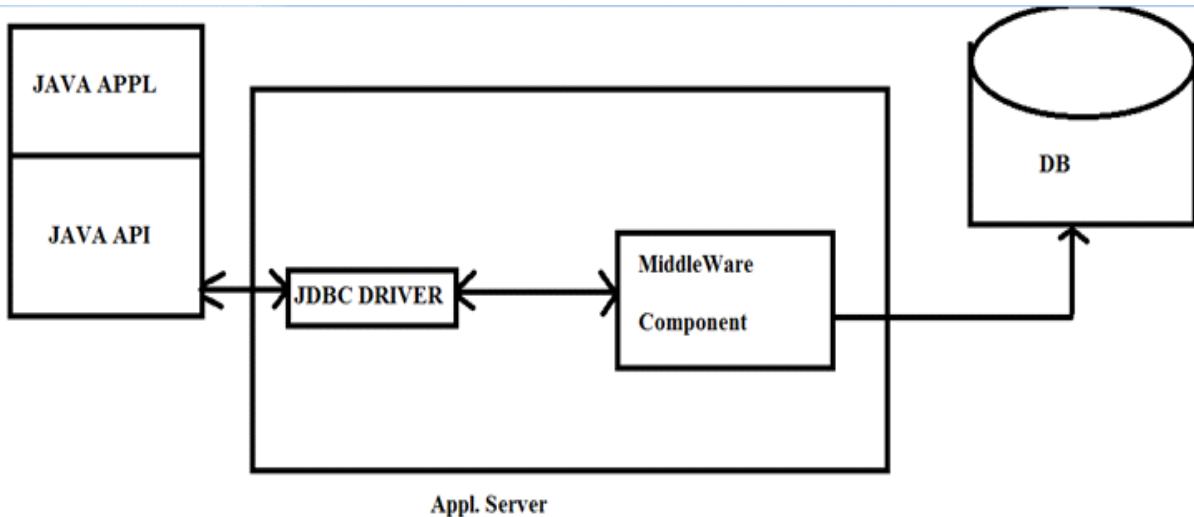
## • Type 2 Driver:

- Type 2 Driver is also called part java, part native driver that is Type 2 Driver was implemented by using Java implementations and the database vendor provided native library.
- When compared to Type1 Driver Type2 Driver is faster Driver because it should not require two times conversions to interact with the Database from Java Applications.
- When compared to Type1 Driver Type2 driver portability is more.
- Type 2 Driver is still recommended for standalone application not suggestible for web applications and Enterprise applications.
- If we want to use Type 2 Driver in our JDBC applications then we have to install the database vendor provided native library.
- Type 2 Driver is cast full Driver among all the drivers.
- Type 2 Driver's portability is not good when compared to Type 3 Driver and Type 4 Driver.



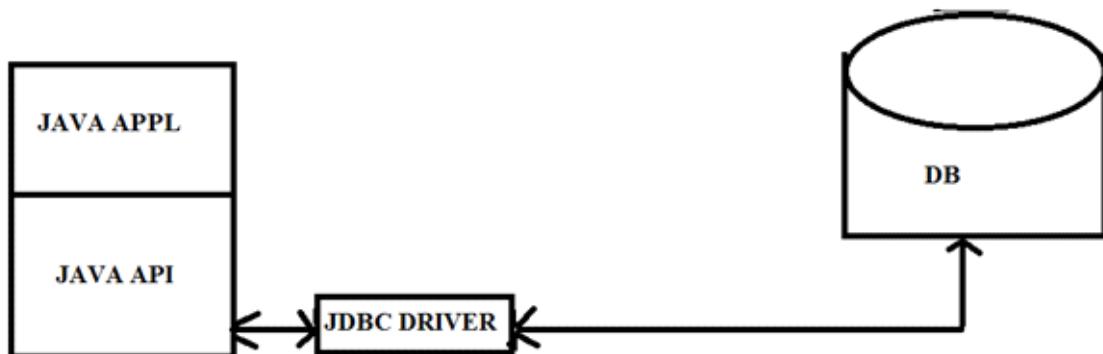
## • Type 3 Driver:

- Type 3 Driver is also called as Middleware Database Server Access Driver and Network Driver.
- Type 3 Driver is purely designed for Enterprise applications it is not suggestible for standalone applications.
- Type 3 Driver portability is very good when compared to Type 1 and Type 2 Driver's.
- Type 3 Driver will provide very good environment to interact with multiple number of databases.
- Type 3 Driver will provide very good environment to switch from one database to another database without having modifications in client applications.
- Type 3 Driver should not require any native library installations, it should require the Compatibility with the application server.
- Type 3 Driver is fastest Driver when compared to all the Drivers.



## **Type 4 Driver:**

- Type 4 Driver is also called as pure Java Driver and Thin Driver because Type 4 Driver was implemented completely by using java implementations.
- Type 4 Driver is the frequent used Driver when compared to all the remaining Drivers.
- Type 4 Driver is recommended for any type application includes standalone applications, Network Applications.
- Type 4 Driver portability is very good when compared to all the remaining Drivers.
- Type 4 driver should not require any native library dependences and it should require one time conversion to interact with database from Java Applications.
- Type 4 is the cheapest Driver among all.



## **Steps to design JDBC Application:**

- Load and register the Driver.
- Establish the connection between Java Application.
- Prepare either Statement or prepared Statement or CallableStatement Objects.
- Write and execute SQL Queries.
- Close the connection



## **Load and Register the Driver:**

- In general Driver is an interface provided by Sun Microsystems and whose implementation classes are provided by the Database Vendors as part of their Database Softwares.
- To load and Register the Driver first we have to make available Driver implementation to JDBC application. For this we have to set classpath environment variable to the location Where we have Driver implementation class.
- If we want to use Type1 Driver provided by Sun MicroSystems in our JDBC applications then it is not required to set classpath environment variable because Type1 Driver was provided by Sun MicroSystems as part of Java Software in the form of sun.jdbc.odbc.JdbcOdbcDriver
- If we want to use Type1 Driver in our JDBC applications then before loading we have to Configure the Microsoft product ODBC Driver.
- To configure Microsoft product ODBC Driver we have to use the following path.

## **To Setting DSN**

- Start
- Control Panel
- System and Security
- Administrative Tools
- Data Sources (ODBC)
- user DSN
- Click on Add button
- Select Microsoft ODBC for the Oracle
- Click on Finish button
- Provide DSN name (provide any name)
- Click on OK

To load and register Driver in our JDBC applications we have to use the following method from class

'Class'  
    Public static Class forName(String class\_Name)

EX: Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");



- When JVM encounter the above instruction JVM will pick up the parameter that is JDBCODbcDriver Class name and JVM will search for its .class file in the current location, if it is not available then JVM will search for it in Java predefined library.
- If JVM identify JDBCODBCDriver.class file in Java pre-defined library (rt.jar) then JVM will load JdbcOdbcDriver class byte code to the memory.
- At the time of loading JdbcOdbcDriver class byte code to the memory JVM will execute a static block, As part of this JVM will execute a method call like DriverManager.registerDriver(); by the execution of registerDriver() method only JDBCODbcDrvier will be available to our JDBC applications.
- In case of Type1 Driver if we use either JDBC 4.0 version or JDK 6.0 version then it is optional.
- To perform loading and register the driver step because JVM will perform Drvier registration automatically at the time of establishing the connection between Java application and Database.

**NOTE:** To prepare JDBC applications Java API has provided the required pre-defined library in the form of java.sql package so that we have to import this package in our Java file.

Import java.sql.\*;

java.sql package includes the following pre-defined library to design JDBC applications.

java.sql package includes the following predefined library.

I Interface              C Class

Driver (I)  
DriverManager (C)  
Connection (I)  
Statement (I)  
PreparedStatement (I)  
ResultSet (I)  
ResultSetMetaData (I)  
DatabaseMetaData (I)  
Savepoint(i)



- **Establish the Connection between Java Application and Database:**

- To establish the connection between Java Application and Database we have to use the following Method from DriverManager Class.

```
public static Connection getConnection  
        (String url, String db_user_name, String db_password)
```

**EX:** Connection con =

```
DriverManager.getConnection("jdbc:odbc:dsnName","system","durga");
```

- When JVM encounter the above instruction JVM will access getConnection method, as part of the getConnection method JVM will access connect() method to establish virtual socket connection between Java application and database as per the URL which we provided.

Where getConnection() method will take three parameters

- Driver URL
- Database username
- Database password

- In general from Driver to Driver Driver class name and Driver URL will varied.
- If we use Type1 Driver then we have to use the following Driver class name and URL  
d-class : sun.jdbc.odbc.JdbcOdbcDriver  
url: jdbc:odbc:dsnName
- In general all the JDBC Drivers should have an URL with the following format.  
main-protocol: sub-protocol
- Where main-protocol name should be JDBC for each and every Driver but the sub protocol name should be varied from Driver to Driver.

Q) **In JDBC Applications getConnection() Method will establish the Connection between Java Application and Database and Return Connection Object but Connection is an Interface how it is possible to Create Connection Object?**

In general in Java technology we are unable to create objects for the interfaces directly, if we want to accommodate interface data in the form of objects then we have to take either an implementation class or Anonymous Inner class.

- If we take implementation class as an alternative then it may allow its own data apart from the Data declared in interface and implementation class object should have its own identity instead of interface identity.



- If we want to create an object with only interface identity and to allow only interface data we have to use Anonymous inner class as an alternative.
- In JDBC Applications getConnection() method will return connection object by returning anonymous Inner class object of connection interface.

**NOTE:** To create connection object taking an implementation class or anonymous inner class is completely depending on the Driver Implementation.

- **Create either Statement OR PreparedStatement OR CallableStatement Objects as per the Requirement:**

- As part of the JDBC applications after establish the connection between Java application and Database.
- We have to prepare SQL Queries, we have to transfer SQL Queries to the databaseEngine and we have to make Database Engine to execute SQL Queries.
- To write and execute SQL Queries we have to use same predefined library from Statement prepared Statement and callableStatement.
- To use the above required predefined library we have to prepare either Statement OR preparedStatement OR CallableStatement objects.

**Q) What is the difference between Statement, PreparedStatement and CallableStatement Objects?**

- In JDBC Applications when we have a requirement to execute all the SQL Queries independently we have to use Statement.
- In JDBC Applications when we have a requirement to execute the same SQL Query in the next Sequence where to improve the performance of JDBC application we will use PreparedStatement.
- In JDBC Applications when we have a requirement to access stored procedures and functions available at Database from Java application we will use CallableStatement object.

To prepare Statement object we have to use the following method from Connection.

public Statement createStatement()

**EX:** Statement st = con.createStatement();

Where createStatement() method will return Statement object by creating Statement interfaces Anonymous inner class object.

- **Write and Execute SQL Queries:**

- ExecuteQuery()
- ExecuteUpdate()
- Execute()



## Q) What are the differences between executeQuery(), executeUpdate() and execute() Method?

- Where executeQuery() method can be used to execute “selection group SQL Queries” in order to fetch(retrieve) data from Database.
- When JVM encounter executeQuery() method with selection group SQL query then JVM will pickup Selection group SQL Query, send to JdbcOdbcDriver, it will send to Connection. Now connection will carry that SQL Query to the Database Engine through ODBC Driver.
- At Database Database Engine will execute the selection group SQL Query by performing Query Tokenization, Query Parsing, Query Optimization and Query Execution.
- By the execution of selection group SQL Query database engine will fetch the data from database and return to Java Application.
- As Java Technology is pure object oriented, Java application will store the fetched data in the form of an object at heap memory called as “ResultSet”.
- As per the predefined implementation of executeQuery method JVM will return the generated ResultSet Object Reference as Return Value from executeQuery() method.

public ResultSet executeQuery(String sql\_Query) throws SQLException

EX: ResultSet rs = st.executeQuery("select \* from emp1");

- Where executeUpdate() method can be used to execute Updation group SQLQueries in order to Perform the Database Operations like create, insert, update, delete, Drop.
- When JVM encounter Updation group SQL Query with executeUpdate() Method the JVM will pickup that SQL Query and send to Database through Database Connection.
- At Database side Database Engine will execute it, perform Updation from Database, identify rowCount value (number of records got updated) and return to Java Application.
- As per the predefined implementation of executeUpdate() method JVM will return rowCount value From executeUpdate() method.

public int executeUpdate(String sql\_Query) throws Exception

EX: int rowCount = st.executeUpdate

("update emp1 set esal=esal+500 where esal<1000");

- In JDBC Applications execute() Method can be used to execute both selection group and Updation group SQL Queries.
- When JVM encounter selection group SQL Query with execute() method then JVM will send selection Group SQL Query to Database Engine, where Database Engine will execute it and send back the fetched Data to Java Application.
- In Java application ResultSet object will be created with the fetched data but as per the predefined implementation of execute() method JVM will return “true” as a Boolean value.
- When JVM encounter Updation group SQL Query as parameter to execute() method then JVM will send it to the database engine, where Database Engine will perform



Updations on database and return row Count value to Java Application. But as per the predefined implementation of execute() method JVM will return "false" as Boolean value from execute() method

public Boolean execute(String sql\_Query) throws SQLException.

EX: boolean b1 = st.execute ("select \* from emp1");

boolean b2 = st.executeUpdate("update emp1 set esal = esal+500 where esal<10000");

- **Close the connection:**

In JDBC Applications after the Database Logic it is convention to close Database Connection for this we have to use the following Method.

public void close() throws SQLException

EX: con.close();

**JdbcApp1:** The following Example demonstrates how to Create a Table on Database through a JDBC Application by taking Table Name as Dynamic Input.

```
1) //import section
2) import java.sql.*;
3) import java.io.*;
4) class JdbcApp1 {
5)     public static void main(String args[]) throws Exception {
6)         //Load a Register Driver
7)         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
8)
9)         //Establish Connection between Java Application and Database
10)        Connection con =
11)            DriverManager.getConnection("jdbc:odbc:nag","system","durga");
12)
13)        //Prepare Statement
14)        Statement st = con.createStatement();
15)
16)        //Create BufferedReader
17)        BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
18)
19)        //Take Table Name as Dynamic Input
20)        System.out.println("Enter table name");
21)        String tname=br.readLine();
22)
23)        //Prepare SQLQuery
24)        String sql =
25) "create table " + tname + "(eno number,ename varchar2(10),esal number)";
26)
27)        //Execute SQL Query
28)        st.executeUpdate(sql);
29)
```



```
30)         System.out.println("table created successfully");
31)         //Close the Connection
32)         con.close();
33)     }
34} }
```

**JdbcApp2:** The following Example demonstrates how to Insert Number of Records on Database Table by taking Records Data as Dynamic Input

```
1) import java.io.*;
2) import java.sql.*;
3) public class JdbcApp2 {
4)     public static void main(String[] args) throws Exception {
5)         Class.forName("oracle.jdbc.OracleDriver");
6)         Connection con = DriverManager.getConnection
7)             ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
8)         Statement st = con.createStatement();
9)         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
10)        while(true) {
11)            System.out.print("Employee Number:");
12)            int eno = Integer.parseInt(br.readLine());
13)            System.out.print("Employee Name:");
14)            String ename = br.readLine();
15)            System.out.print("Employee Salary:");
16)            float esal = Float.parseFloat(br.readLine());
17)            System.out.print("Employee Address:");
18)            String eaddr = br.readLine();
19)            st.executeUpdate
20)                ("insert into emp1 values("+eno+"','"+ename+"','"+esal+"','"+eaddr+"')");
21)            System.out.println("Employee Inserted Successfully");
22)            System.out.print("Onemore Employee[Yes/No]? :");
23)            String option = br.readLine();
24)            if(option.equals("No")) {
25)                break;
26)            }
27)        }
28)        con.close();
29)    }
30} }
```

In JDBC Applications if we want to used Type1 Driver provided by Sun Micro Systems then we have to use the following Driver cClass and URL

driver.class:sun.jdbc.odbc.JdbcOdbcDriver  
url:jdbc:odbc:dsnName



Similarly if we want to use Type4 Driver provided by oracle we have to use the following Driver class and URL

driver\_class:oracle.jdbc.driver.OracleDriver  
url:jdbc:oracle:thin:@localhost:1521:xe

- Oracle Driver is a class provided by oracle software in the form of Ojdbc14.jar file
- Oracle Software has provided ojdbc14.jar file at the following location  
C:\oracleexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc.jar
- If we want to use Type4 Driver provided by oracle in our JDBC Applications we have to set Classpath Environment Variable to the Location where we have ojdbc14.jar

D:\jdbc4>set classpath =  
%classpath%;C:\oracleexe\app\oracle\product\10.2.0\server\jdbc\lib\ojdbc14.jar

## JdbcApp3: The following Example Demonstrate how to perform Updations on Database Table through JDBC Application

```
1) import java.io.*;  
2) import java.sql.*;  
3) public class JdbcApp3 {  
4)     public static void main(String[] args) throws Exception {  
5)         //Class.forName("oracle.jdbc.OracleDriver");  
6)         Connection con = DriverManager.getConnection  
7)             ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");  
8)         Statement st = con.createStatement();  
9)         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));  
10)        System.out.print("Bonus Amount :");  
11)        int bonus_Amt = Integer.parseInt(br.readLine());  
12)        System.out.print("Salary Range :");  
13)        float sal_Range = Float.parseFloat(br.readLine());  
14)        int rowCount = st.executeUpdate  
15)            ("update emp1 set esal=esal+" + bonus_Amt + " where esal<" + sal_Range);  
16)        System.out.println("Employees Updated :" + rowCount);  
17)        con.close();  
18)    }  
19} }
```



## JdbcApp4: The following Example demonstrates how to Delete Number of Records from Database Table through a JDBC Application

```
1) import java.io.*;
2) import java.sql.*;
3) public class JdbcApp4 {
4)     public static void main(String[] args) throws Exception {
5)         DriverManager.registerDriver(new oracle.jdbc.OracleDriver());
6)         Connection con = DriverManager.getConnection
7)             ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");
8)         Statement st = con.createStatement();
9)         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
10)        System.out.print("Salary Range :");
11)        float sal_Range = Float.parseFloat(br.readLine());
12)        int rowCount = st.executeUpdate("delete from emp1 where esal<" + sal_Range);
13)        System.out.println("Records Deleted :" + rowCount);
14)        con.close();
15)    }
16} }
```

In JDBC Application we will use `executeUpdate()` method to execute the Updation group SQL Queries like create, insert, update, delete, drop, alter and so on.

- If we execute the SQL Queries like insert, update and delete then really some number of record will be updated on database table then that number will be return as `rowCount` value from `executeUpdate()`.
- If we execute the SQL Queries like create, alter, drop with `executeUpdate()` method then records manipulation is not available on database, in this context the return value from `executeUpdate()` method is completely depending on the type of Driver which we used in JDBC application.
- In the above context if we use type1 Driver provided by SunMicroSystems the `executeUpdate()` method will return "-1" as `rowCount` value.
- For the above requirement if we use Type4 Driver provided by oracle then `executeUpdate()` method will return "0" as `rowCount` value.



## JdbcApp5:

```
1) import java.sql.*;  
2) public class JdbcApp5 {  
3)     public static void main(String[] args)throws Exception {  
4)         Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");  
5)         Connection con =  
6)             DriverManager.getConnection("jdbc:odbc:nag","system","durga");  
7)         Statement st = con.createStatement();  
8)         int rowCount1 = st.executeUpdate("create table emp1(eno number)");  
9)         System.out.println(rowCount1);  
10)        int rowCount2 = st.executeUpdate("drop table emp1");  
11)        System.out.println(rowCount2);  
12)        con.close();  
13)    }  
14} }
```

## JdbcApp6:

```
1) import java.sql.*;  
2) public class JdbcApp6 {  
3)     public static void main(String[] args)throws Exception {  
4)         Class.forName("oracle.jdbc.OracleDriver");  
5)         Connection con = DriverManager.getConnection  
6)             ("jdbc:oracle:thin:@localhost:1521:xe","system","durga");  
7)         Statement st = con.createStatement();  
8)         int rowCount1 = st.executeUpdate("create table emp1(eno number)");  
9)         System.out.println(rowCount1);  
10)        int rowCount2 = st.executeUpdate("drop table emp1");  
11)        System.out.println(rowCount2);  
12)        con.close();  
13)    }  
14} }
```



# Java 8 Features



# Table of Contents

|                                     |     |
|-------------------------------------|-----|
| 1) Lambda Expressions .....         | 610 |
| 2) Functional Interfaces .....      | 612 |
| 3) Default methods .....            | 621 |
| 4) Predicates .....                 | 626 |
| 5) Functions .....                  | 629 |
| 6) Double colon operator (::) ..... | 631 |
| 7) Stream API .....                 | 634 |
| 8) Date and Time API .....          | 640 |



---

Java 7 – July 28<sup>th</sup> 2011  
2 Years 7 Months 18 Days

Java 8 - March 18<sup>th</sup> 2014

Java 9 - September 22<sup>nd</sup> 2016

Java 10 - 2018

After Java 1.5 version, Java 8 is the next major version.

Before Java 8, sun people gave importance only for objects but in 1.8 version oracle people gave the importance for functional aspects of programming to bring its benefits to Java. i.e it doesn't mean Java is functional oriented programming language.

## Java 8 New Features:

- 1) Lambda Expression
- 2) Functional Interfaces
- 3) Default methods
- 4) Predicates
- 5) Functions
- 6) Double colon operator (::)
- 7) Stream API
- 8) Date and Time API
- Etc.....



# Lambda ( $\lambda$ ) Expressions

- ✿ Lambda calculus is a big change in mathematical world which has been introduced in 1930. Because of benefits of Lambda calculus slowly this concepts started using in programming world. "LISP" is the first programming which uses Lambda Expression.
- ✿ The other languages which uses lambda expressions are:
  - C#.Net
  - C Objective
  - C
  - C++
  - Python
  - Ruby etc.and finally in Java also.
- ✿ The Main Objective of Lambda Expression is to bring benefits of functional programming into Java.

## What is Lambda Expression ( $\lambda$ ):

- Lambda Expression is just an anonymous (nameless) function. That means the function which doesn't have the name, return type and access modifiers.
- Lambda Expression also known as anonymous functions or closures.

### Ex: 1

```
public void m1() { }           0 → {  
    sop("hello"); }           sop("hello");  
}                           }  
                           0 → { sop("hello"); }  
                           0 → sop("hello");
```

### Ex:2

```
public void add(int a, int b) { }           (int a, int b) → sop(a+b);  
    sop(a+b); }  
}
```

- If the type of the parameter can be decided by compiler automatically based on the context then we can remove types also.
- The above Lambda expression we can rewrite as  $(a,b) \rightarrow sop(a+b)$ ;



## Ex: 3

```
public String str(String str) {  
    return str;  
} } (String str) → return str;  
                                ↓  
                                (str) → str;
```

## Conclusions:

- 1) A lambda expression can have zero or more number of parameters (arguments).

Ex:

0 → sop("hello");  
(int a) → sop(a);  
(int a, int b) → return a+b;

- 2) Usually we can specify type of parameter. If the compiler expects the type based on the context then we can remove type. i.e., programmer is not required.

Ex:

(int a, int b) → sop(a+b);  
 ↓  
(a,b) → sop(a+b);

- 3) If multiple parameters present then these parameters should be separated with comma (,).

- 4) If zero number of parameters available then we have to use empty parameter [ like ()].

Ex: 0 → sop("hello");

- 5) If only one parameter is available and if the compiler can expect the type then we can remove the type and parenthesis also.

Ex:

(int a) → sop(a);  
 ↓  
(a) → sop(a);  
 ↓  
A → sop(a);

- 6) Similar to method body lambda expression body also can contain multiple statements. If more than one statements present then we have to enclose inside within curly braces. If one statement present then curly braces are optional.

- 7) Once we write lambda expression we can call that expression just like a method, for this functional interfaces are required.



# Functional Interfaces

If an interface contain only one abstract method, such type of interfaces are called functional interfaces and the method is called functional method or single abstract method (SAM).

Ex:

- 1) Runnable → It contains only run() method
- 2) Comparable → It contains only compareTo() method
- 3) ActionListener → It contains only actionPerformed()
- 4) Callable → It contains only call() method

Inside functional interface in addition to single Abstract method (SAM) we write any number of default and static methods.

Ex:

```
1) interface Interf {  
2)     public abstract void m1();  
3)     default void m2() {  
4)         System.out.println ("hello");  
5)     }  
6) }
```

In Java 8, Sun Micro System introduced @Functional Interface annotation to specify that the interface is Functional Interface.

Ex:

```
@Functional Interface  
Interface Interf {  
    public void m1();  
}
```

This code compiles without any compilation errors.

Inside Functional Interface we can take only one abstract method, if we take more than one abstract method then compiler raise an error message that is called we will get compilation error.

Ex:

```
@Functional Interface {  
    public void m1();  
    public void m2();  
}
```

This code gives compilation error.

Inside Functional Interface we have to take exactly only one abstract method.If we are not declaring that abstract method then compiler gives an error message.



Ex:

```
@Functional Interface {  
    interface Interface { } } compilation error  
}
```

## Functional Interface with respect to Inheritance:

If an interface extends Functional Interface and child interface doesn't contain any abstract method then child interface is also Functional Interface

Ex:

```
1) @Functional Interface  
2) interface A {  
3)     public void methodOne();  
4) }  
5) @Functional Interface  
6) Interface B extends A {  
7) }
```

In the child interface we can define exactly same parent interface abstract method.

Ex:

```
1) @Functional Interface  
2) interface A {  
3)     public void methodOne();  
4) }  
5) @Functional Interface  
6) interface B extends A {  
7)     public void methodOne();  
8) }
```

No Compile Time Error

In the child interface we can't define any new abstract methods otherwise child interface won't be Functional Interface and if we are trying to use @Functional Interface annotation then compiler gives an error message.

```
1) @Functional Interface {  
2) interface A {  
3)     public void methodOne();  
4) }  
5) @Functional Interface  
6) interface B extends A {  
7)     public void methodTwo();  
8) }
```

Compiletime Error



Ex:

```
@Functional Interface
interface A {
    public void methodOne();
}

interface B extends A {
    public void methodTwo();
}
```

No compile time error

This's Normal interface so that code compiles without error

In the above example in both parent & child interface we can write any number of default methods and there are no restrictions. Restrictions are applicable only for abstract methods.

## Functional Interface Vs Lambda Expressions:

Once we write Lambda expressions to invoke it's functionality, then Functional Interface is required. We can use Functional Interface reference to refer Lambda Expression.

Where ever Functional Interface concept is applicable there we can use Lambda Expressions

### Ex:1 Without Lambda Expression

```
1) interface Interf {
2)     public void methodOne() {}
3)     public class Demo implements Interface {
4)         public void methodOne() {
5)             System.out.println("method one execution");
6)         }
7)         public class Test {
8)             public static void main(String[] args) {
9)                 Interfi = new Demo();
10)                i.methodOne();
11)            }
12} }
```

### Above code With Lambda expression

```
1) interface Interf {
2)     public void methodOne() {}
3)     class Test {
4)         public static void main(String[] args) {
5)             Interfi = () → System.out.println("MethodOne Execution");
6)             i.methodOne();
7)         }
8)     }
```



## Without Lambda Expression

```
1) interface Interf {  
2)     public void sum(int a,int b);  
3) }  
4) class Demo implements Interf {  
5)     public void sum(int a,int b) {  
6)         System.out.println("The sum:" +(a+b));  
7)     }  
8) }  
9) public class Test {  
10)    public static void main(String[] args) {  
11)        Interfi = new Demo();  
12)        i.sum(20,5);  
13)    }  
14) }
```

## Above code With Lambda Expression

```
1) interface Interf {  
2)     public void sum(int a, int b);  
3) }  
4) class Test {  
5)     public static void main(String[] args) {  
6)         Interfi = (a,b) → System.out.println("The Sum:" +(a+b));  
7)         i.sum(5,10);  
8)     }  
9) }
```

## Without Lambda Expressions

```
1) interface Interf {  
2)     public int square(int x);  
3) }  
4) class Demo implements Interf {  
5)     public int square(int x) {  
6)         return x*x; OR (int x) → x*x  
7)     }  
8) }  
9) class Test {  
10)    public static void main(String[] args) {  
11)        Interfi = new Demo();  
12)        System.out.println("The Square of 7 is: " +i.square(7));  
13)    }  
14) }
```



## Above code with Lambda Expression

```
1) interface Interf {  
2)     public int square(int x);  
3) }  
4) class Test {  
5)     public static void main(String[] args) {  
6)         Interfi = x → x*x;  
7)         System.out.println("The Square of 5 is:"+i.square(5));  
8)     }  
9) }
```

## Without Lambda expression

```
1) class MyRunnable implements Runnable {  
2)     public void main() {  
3)         for(int i=0; i<10; i++) {  
4)             System.out.println("Child Thread");  
5)         }  
6)     }  
7) }  
8) class ThreadDemo {  
9)     public static void main(String[] args) {  
10)         Runnable r = new myRunnable();  
11)         Thread t = new Thread(r);  
12)         t.start();  
13)         for(int i=0; i<10; i++) {  
14)             System.out.println("Main Thread")  
15)         }  
16)     }  
17) }
```

## With Lambda expression

```
1) class ThreadDemo {  
2)     public static void main(String[] args) {  
3)         Runnable r = () → {  
4)             for(int i=0; i<10; i++) {  
5)                 System.out.println("Child Thread");  
6)             }  
7)         };  
8)         Thread t = new Thread(r);  
9)         t.start();  
10)        for(i=0; i<10; i++) {  
11)            System.out.println("Main Thread");  
12)        }  
13)    }  
14) }
```



## Anonymous inner classes vs Lambda Expressions

Wherever we are using anonymous inner classes there may be a chance of using Lambda expression to reduce length of the code and to resolve complexity.

### Ex: With anonymous inner class

```
1) class Test {  
2)     public static void main(String[] args) {  
3)         Thread t = new Thread(new Runnable() {  
4)             public void run() {  
5)                 for(int i=0; i<10; i++) {  
6)                     System.out.println("Child Thread");  
7)                 }  
8)             };  
9)         t.start();  
10)        for(int i=0; i<10; i++) {  
11)            System.out.println("Main thread");  
12)        }  
13)    }  
14) }
```

### With Lambda expression

```
1) class Test {  
2)     public static void main(String[] args) {  
3)         Thread t = new Thread(() -> {  
4)             for(int i=0; i<10; i++) {  
5)                 System.out.println("Child Thread");  
6)             }  
7)         );  
8)         t.start();  
9)         for(int i=0; i<10; i++) {  
10)            System.out.println("Main Thread");  
11)        }  
12)    }  
13) }
```

## What are the advantages of Lambda expression?

- We can reduce length of the code so that readability of the code will be improved.
- We can resolve complexity of anonymous inner classes.
- We can provide Lambda expression in the place of object.
- We can pass lambda expression as argument to methods.



## Note:

- Anonymous inner class can extend concrete class, can extend abstract class, can implement interface with any number of methods but
- Lambda expression can implement an interface with only single abstract method (Functional Interface).
- Hence if anonymous inner class implements Functional Interface in that particular case only we can replace with lambda expressions. Hence wherever anonymous inner class concept is there, it may not possible to replace with Lambda expressions.
- Anonymous inner class! = Lambda Expression
  
- Inside anonymous inner class we can declare instance variables.
- Inside anonymous inner class "this" always refers current inner class object(anonymous inner class) but not related outer class object

## Ex:

- Inside lambda expression we can't declare instance variables.
- Whatever the variables declare inside lambda expression are simply acts as local variables
- Within lambda expression 'this' keyword represents current outer class object reference (that is current enclosing class reference in which we declare lambda expression)

## Ex:

```
1) interface Interf {  
2)     public void m10;  
3) }  
4) class Test {  
5)     int x = 777;  
6)     public void m20 {  
7)         Interfi = ()→ {  
8)             int x = 888;  
9)             System.out.println(x); 888  
10)            System.out.println(this.x); 777  
11)        };  
12)        i.m10;  
13)    }  
14)    public static void main(String[] args) {  
15)        Test t = new Test();  
16)        t.m20;  
17)    }  
18) }
```

- From lambda expression we can access enclosing class variables and enclosing method variables directly.



- ★ The local variables referenced from lambda expression are implicitly final and hence we can't perform re-assignment for those local variables otherwise we get compile time error.

Ex:

```
1) interface Interf {  
2)     public void m10;  
3) }  
4) class Test {  
5)     int x = 10;  
6)     public void m20 {  
7)         int y = 20;  
8)         Interfi = () → {  
9)             System.out.println(x); 10  
10)            System.out.println(y); 20  
11)            x = 888;  
12)            y = 999; //CE  
13)        };  
14)        i.m10;  
15)        y = 777;  
16)    }  
17)    public static void main(String[] args) {  
18)        Test t = new Test();  
19)        t.m20;  
20)    }  
21) }
```



## Differences between anonymous inner classes and Lambda expression

| Anonymous Inner class                                                                                                             | Lambda Expression                                                                                                                          |
|-----------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------|
| It's a class without name                                                                                                         | It's a method without name (anonymous function)                                                                                            |
| Anonymous inner class can extend Abstract and concrete classes                                                                    | lambda expression can't extend Abstract and concrete classes                                                                               |
| Anonymous inner class can implement An interface that contains any number of Abstract methods                                     | lambda expression can implement an Interface which contains single abstract method (Functional Interface)                                  |
| Inside anonymous inner class we can Declare instance variables.                                                                   | Inside lambda expression we can't Declare instance variables, whatever the variables declared are simply acts as local variables.          |
| Anonymous inner classes can be Instantiated                                                                                       | lambda expressions can't be instantiated                                                                                                   |
| Inside anonymous inner class "this" Always refers current anonymous Inner class object but not outer class Object.                | Inside lambda expression "this" Always refers current outer class object. That is enclosing class object.                                  |
| Anonymous inner class is the best choice If we want to handle multiple methods.                                                   | Lambda expression is the best Choice if we want to handle interface With single abstract method (Functional Interface).                    |
| In the case of anonymous inner class At the time of compilation a separate Dot class file will be generated (outerclass\$1.class) | At the time of compilation no dot Class file will be generated for Lambda expression. It simply converts in to private method outer class. |
| Memory allocated on demand Whenever we are creating an object                                                                     | Reside in permanent memory of JVM (Method Area).                                                                                           |



# Default Methods

- Until 1.7 version onwards inside interface we can take only public abstract methods and public static final variables (every method present inside interface is always public and abstract whether we are declaring or not).
- Every variable declared inside interface is always public static final whether we are declaring or not.
- But from 1.8 version onwards in addition to these, we can declare default concrete methods also inside interface, which are also known as defender methods.
- We can declare default method with the keyword “default” as follows

```
1) default void m10{  
2)     System.out.println ("Default Method");  
3) }
```

- Interface default methods are by-default available to all implementation classes. Based on requirement implementation class can use these default methods directly or can override.

Ex:

```
1) interface Interf {  
2)     default void m10 {  
3)         System.out.println("Default Method");  
4)     }  
5) }  
6) class Test implements Interf {  
7)     public static void main(String[] args) {  
8)         Test t = new Test();  
9)         t.m10;  
10)    }  
11) }
```

- Default methods also known as defender methods or virtual extension methods.
- The main advantage of default methods is without effecting implementation classes we can add new functionality to the interface (backward compatibility).

**Note:** We can't override object class methods as default methods inside interface otherwise we get compile time error.



Ex:

```
1) interface Interf {  
2)     default int hashCode() {  
3)         return 10;  
4)     }  
5) }
```

CompileTimeError

Reason: Object class methods are by-default available to every Java class hence it's not required to bring through default methods.

## Default method vs multiple inheritance

Two interfaces can contain default method with same signature then there may be a chance of ambiguity problem (diamond problem) to the implementation class. To overcome this problem compulsory we should override default method in the implementation class otherwise we get compile time error.

```
1) Eg 1:  
2) interface Left {  
3)     default void m1() {  
4)         System.out.println("Left Default Method");  
5)     }  
6) }  
7)  
8) Eg 2:  
9) interface Right {  
10)    default void m1() {  
11)        System.out.println("Right Default Method");  
12)    }  
13) }  
14)  
15) Eg 3:  
16) class Test implements Left, Right {}
```

## How to override default method in the implementation class?

In the implementation class we can provide complete new implementation or we can call any interface method as follows.

interfacename.super.m1();



Ex:

```
1) class Test implements Left, Right {  
2)     public void m10() {  
3)         System.out.println("Test Class Method"); OR Left.super.m10();  
4)     }  
5)     public static void main(String[] args) {  
6)         Test t = new Test();  
7)         t.m10();  
8)     }  
9) }
```

## Differences between interface with default methods and abstract class

Even though we can add concrete methods in the form of default methods to the interface, it won't be equal to abstract class.

| Interface with Default Methods                                                                             | Abstract Class                                                                                           |
|------------------------------------------------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------|
| Inside interface every variable is Always public static final and there is No chance of instance variables | Inside abstract class there may be a Chance of instance variables which Are required to the child class. |
| Interface never talks about state of Object.                                                               | Abstract class can talk about state of Object.                                                           |
| Inside interface we can't declare Constructors.                                                            | Inside abstract class we can declare Constructors.                                                       |
| Inside interface we can't declare Instance and static blocks.                                              | Inside abstract class we can declare Instance and static blocks.                                         |
| Functional interface with default Methods Can refer lambda expression.                                     | Abstract class can't refer lambda Expressions.                                                           |
| Inside interface we can't override Object class methods.                                                   | Inside abstract class we can override Object class methods.                                              |

## Interface with default method != abstract class

## Static methods inside interface:

- ✿ From 1.8 version onwards in addition to default methods we can write static methods also inside interface to define utility functions.
- ✿ Interface static methods by-default not available to the implementation classes hence by using implementation class reference we can't call interface static



- methods. We should call interface static methods by using interface name.

### Ex:

```
1) interface Interf {  
2)     public static void sum(int a, int b) {  
3)         System.out.println("The Sum:"+ (a+b));  
4)     }  
5) }  
6) class Test implements Interf {  
7)     public static void main(String[] args) {  
8)         Test t = new Test();  
9)         t.sum(10, 20); //CE  
10)        Test.sum(10, 20); //CE  
11)        Interf.sum(10, 20);  
12)    }  
13) }
```

- As interface static methods by default not available to the implementation class, overriding concept is not applicable.
- Based on our requirement we can define exactly same method in the implementation class, it's valid but not overriding.

### Ex:1

```
1) interface Interf {  
2)     public static void m1() {}  
3) }  
4) class Test implements Interf {  
5)     public static void m1() {}  
6) }
```

It's valid but not overriding

### Ex:2

```
1) interface Interf {  
2)     public static void m1() {}  
3) }  
4) class Test implements Interf {  
5)     public void m1() {}  
6) }
```

This's valid but not overriding

### Ex3:



```
1) class P {  
2)     private void m1() {}  
3) }  
4) class C extends P {  
5)     public void m1() {}  
6) }
```

This's valid but not overriding

From 1.8 version onwards we can write main() method inside interface and hence we can run interface directly from the command prompt.

Ex:

```
1) interface Interf {  
2)     public static void main(String[] args) {  
3)         System.out.println("Interface Main Method");  
4)     }  
5) }
```

At the command prompt:

Javac Interf.java

JavalInterf



# Predicates

- A predicate is a function with a single argument and returns boolean value.
- To implement predicate functions in Java, Oracle people introduced Predicate interface in 1.8 version (i.e., Predicate<T>).
- Predicate interface present in *Java.util.function* package.
- It's a functional interface and it contains only one method i.e., test()

Ex:

```
interface Predicate<T> {  
    public boolean test(T t);  
}
```

As predicate is a functional interface and hence it can refers lambda expression

Ex:1 Write a predicate to check whether the given integer is greater than 10 or not.

Ex:

```
public boolean test(Integer I) {  
    if (I > 10) {  
        return true;  
    }  
    else {  
        return false;  
    }  
}
```



```
(Integer I) → {  
    if(I > 10)  
        return true;  
    else  
        return false;  
}
```



```
I → (I > 10);
```



```
predicate<Integer> p = I →(I > 10);  
System.out.println (p.test(100)); true  
System.out.println (p.test(7)); false
```



## Program:

```
1) import Java.util.function;
2) class Test {
3)     public static void main(String[] args) {
4)         predicate<Integer> p = i → (i>10);
5)         System.out.println(p.test(100));
6)         System.out.println(p.test(7));
7)         System.out.println(p.test(true)); //CE
8)     }
9) }
```

# 1 Write a predicate to check the length of given string is greater than 3 or not.

```
Predicate<String> p = s → (s.length() > 3);
System.out.println (p.test("rvkb")); true
System.out.println (p.test("rk")); false
```

#-2 write a predicate to check whether the given collection is empty or not.

```
Predicate<collection> p = c → c.isEmpty();
```

## Predicate joining

It's possible to join predicates into a single predicate by using the following methods.

```
and()
or()
negate()
```

these are exactly same as logical AND ,OR complement operators

## Ex:

```
1) import Java.util.function.*;
2) class test {
3)     public static void main(string[] args) {
4)         int[] x = {0, 5, 10, 15, 20, 25, 30};
5)         predicate<integer> p1 = i->i>10;
6)         predicate<integer> p2=i -> i%2==0;
7)         System.out.println("The Numbers Greater Than 10:");
8)         m1(p1, x);
9)         System.out.println("The Even Numbers Are:");
10)        m1(p2, x);
11)        System.out.println("The Numbers Not Greater Than 10:");
12)        m1(p1.negate(), x);
13)        System.out.println("The Numbers Greater Than 10 And Even Are: ");
14)        m1(p1.and(p2), x);
15)        System.out.println("The Numbers Greater Than 10 OR Even: ");
16)        m1(p1.or(p2), x);
17)    }
18)    public static void m1(predicate<integer>p, int[] x) {
```



```
19)     for(int x1:x) {  
20)         if(p.test(x1))  
21)             System.out.println(x1);  
22)     }  
23) }  
24) }
```



# Functions

- Functions are exactly same as predicates except that functions can return any type of result but function should (can) return only one value and that value can be any type as per our requirement.
- To implement functions oracle people introduced Function interface in 1.8version.
- Function interface present in *Java.util.function* package.
- Functional interface contains only one method i.e., apply()

```
interface function(T,R) {  
    public R apply(T t);  
}
```

**Assignment:** Write a function to find length of given input string.

**Ex:**

```
1) import Java.util.function.*;  
2) class Test {  
3)     public static void main(String[] args) {  
4)         Function<String, Integer> f = s ->s.length();  
5)         System.out.println(f.apply("Durga"));  
6)         System.out.println(f.apply("Soft"));  
7)     }  
8) }
```

**Note:** Function is a functional interface and hence it can refer lambda expression.



## Differences between predicate and function

| Predicate                                                                                   | Function                                                                                                                            |
|---------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------|
| To implement conditional checks<br>We should go for predicate                               | To perform certain operation And to return some result we Should go for function.                                                   |
| Predicate can take one type Parameter which represents Input argument type.<br>Predicate<T> | Function can take 2 type Parameters. First one represent Input argument type and Second one represent return Type.<br>Function<T,R> |
| Predicate interface defines only one method called test()                                   | Function interface defines only one Method called apply().                                                                          |
| public boolean test(T t)                                                                    | public R apply(T t)                                                                                                                 |
| Predicate can return only boolean value.                                                    | Function can return any type of value                                                                                               |

**Note:** Predicate is a boolean valued function and(), or(), negate() are default methods present inside Predicate interface.



# Method And Constructor References by using :: (Double Colon) Operator

- Functional Interface method can be mapped to our specified method by using :: (double colon) operator. This is called method reference.
- Our specified method can be either static method or instance method.
- Functional Interface method and our specified method should have same argument types, except this the remaining things like return type, methodname, modifiers etc are not required to match.

## Syntax

If our specified method is static method  
Classname::methodName

If the method is instance method  
Objref::methodName

- Functional Interface can refer lambda expression and Functional Interface can also refer method reference. Hence lambda expression can be replaced with method reference. Hence method reference is alternative syntax to lambda expression.

## Ex: With Lambda Expression

```
1) class Test {  
2)     public static void main(String[] args) {  
3)         Runnable r = () → {  
4)             for(int i=0; i<=10; i++) {  
5)  
6)                 System.out.println("Child Thread");  
7)             }  
8)         };  
9)         Thread t = new Thread(r);  
10)        t.start();  
11)        for(int i=0; i<=10; i++) {  
12)            System.out.println("Main Thread");  
13)        }  
14)    }
```

## With Method Reference

```
1) class Test {  
2)     public static void m1() {  
3)         for(int i=0; i<=10; i++) {  
4)             System.out.println("Child Thread");  
5)         }  
6)     }  
7)     public static void main(String[] args) {  
8)         Runnable r = Test:: m1;
```



```
9)     Thread t = new Thread(r);
10)    t.start();
11)    for(int i=0; i<=10; i++) {
12)        System.out.println("Main Thread");
13)    }
14) }
```

In the above example Runnable interface run() method referring to Test class static method m1().  
Method reference to Instance method:

Ex:

```
1) interface Interf {
2)     public void m1(int i);
3) }
4) class Test {
5)     public void m2(int i) {
6)         System.out.println("From Method Reference:"+i);
7)     }
8)     public static void main(String[] args) {
9)         Interf f = I ->sop("From Lambda Expression:"+i);
10)        f.m1(10);
11)        Test t = new Test();
12)        Interf i1 = t::m2;
13)        i1.m1(20);
14)    }
15) }
```

In the above example functional interface method m1() referring to Test class instance method m2().

The main advantage of method reference is we can use already existing code to implement functional interfaces (code reusability).

## Constructor References

We can use :: ( double colon )operator to refer constructors also

Syntax: classname :: new

Ex:

```
Interf f = sample :: new;
```

functional interface f referring sample class constructor



Ex:

```
1) class Sample {  
2)     private String s;  
3)     Sample(String s) {  
4)         this.s = s;  
5)         System.out.println("Constructor Executed:"+s);  
6)     }  
7) }  
8) interface Interf {  
9)     public Sample get(String s);  
10} }  
11) class Test {  
12)     public static void main(String[] args) {  
13)         Interf f = s -> new Sample(s);  
14)         f.get("From Lambda Expression");  
15)         Interf f1 = Sample :: new;  
16)         f1.get("From Constructor Reference");  
17)     }  
18) }
```

**Note:** In method and constructor references compulsory the argument types must be matched.



# STREAMS

To process objects of the collection, in 1.8 version Streams concept introduced.

### What is the differences between Java.util.streams and Java.io streams?

java.util streams meant for processing objects from the collection. i.e, it represents a stream of objects from the collection but Java.io streams meant for processing binary and character data with respect to file. i.e it represents stream of binary data or character data from the file .hence Java.io streams and Java.util streams both are different.

### What is the difference between collection and stream?

- If we want to represent a group of individual objects as a single entity then We should go for collection.
- If we want to process a group of objects from the collection then we should go for streams.
- We can create a stream object to the collection by using stream() method of Collection interface. stream() method is a default method added to the Collection in 1.8 version.

`default Stream stream()`

Ex: `Stream s = c.stream();`

- Stream is an interface present in `java.util.stream`. Once we got the stream, by using that we can process objects of that collection.
- We can process the objects in the following 2 phases

- 1.Configuration
- 2.Processing



## 1) Configuration:

We can configure either by using filter mechanism or by using map mechanism.

### Filtering:

We can configure a filter to filter elements from the collection based on some boolean condition by using filter() method of Stream interface.

```
public Stream filter(Predicate<T> t)
```

here (Predicate<T> t) can be a boolean valued function/lambda expression

Ex:

```
Stream s = c.stream();
Stream s1 = s.filter(i -> i%2==0);
```

Hence to filter elements of collection based on some Boolean condition we should go for filter() method.

### Mapping:

If we want to create a separate new object, for every object present in the collection based on our requirement then we should go for map() method of Stream interface.

```
public Stream map (Function f);
```

→ It can be lambda expression also

Ex:

```
Stream s = c.stream();
Stream s1 = s.map(i-> i+10);
```

Once we performed configuration we can process objects by using several methods.

## 2) Processing

processing by collect() method  
Processing by count() method  
Processing by sorted() method  
Processing by min() and max() methods  
forEach() method  
toArray() method  
Stream.of() method



## Processing by collect() method

This method collects the elements from the stream and adding to the specified to the collection indicated (specified) by argument.

**Ex 1:** To collect only even numbers from the array list

### Approach-1: Without Streams

```
1) import Java.util.*;
2) class Test {
3)     public static void main(String[] args) {
4)         ArrayList<Integer> l1 = new ArrayList<Integer>();
5)         for(int i=0; i<=10; i++) {
6)             l1.add(i);
7)         }
8)         System.out.println(l1);
9)         ArrayList<Integer> l2 = new ArrayList<Integer>();
10)        for(Integer i:l1) {
11)            if(i%2 == 0)
12)                l2.add(i);
13)        }
14)        System.out.println(l2);
15)    }
16} }
```

### Approach-2: With Streams

```
1) import Java.util.*;
2) import Java.util.stream.*;
3) class Test {
4)     public static void main(String[] args) {
5)         ArrayList<Integer> l1 = new ArrayList<Integer>();
6)         for(int i=0; i<=10; i++) {
7)             l1.add(i);
8)         }
9)         System.out.println(l1);
10)        List<Integer> l2 = l1.stream().filter(i -> i%2==0).collect(Collectors.toList());
11)        System.out.println(l2);
12)    }
13} }
```



## Ex: Program for map() and collect() Method

```
1) import Java.util.*;
2) import Java.util.stream.*;
3) class Test {
4)     public static void main(String[] args) {
5)         ArrayList<String> l = new ArrayList<String>();
6)         l.add("rvk"); l.add("rk"); l.add("rvki"); l.add("rvkir");
7)         System.out.println(l);
8)         List<String> l2 = l.stream().map(s -
>s.toUpperCase()).collect(Collectors.toList());
9)         System.out.println(l2);
10)    }
11) }
```

## II.Processing by count()method

This method returns number of elements present in the stream.

```
public long count()
```

Ex:

```
long count = l.stream().filter(s ->s.length()==5).count();
sop("The number of 5 length strings is:"+count);
```

## III.Processing by sorted()method

If we sort the elements present inside stream then we should go for sorted() method.  
the sorting can either default natural sorting order or customized sorting order specified by comparator.

sorted()- default natural sorting order

sorted(Comparator c)-customized sorting order.

Ex:

```
List<String> l3=l.stream().sorted().collect(Collectors.toList());
sop("according to default natural sorting order:"+l3);
```

```
List<String> l4=l.stream().sorted((s1,s2) -> -s1.compareTo(s2)).collect(Collectors.toList());
sop("according to customized sorting order:"+l4);
```



## IV.Processing by min() and max() methods

**min(Comparator c)**

returns minimum value according to specified comparator.

**max(Comparator c)**

returns maximum value according to specified comparator

**Ex:**

```
String min=l.stream().min((s1,s2) -> s1.compareTo(s2)).get();
sop("minimum value is:"+min);
```

```
String max=l.stream().max((s1,s2) -> s1.compareTo(s2)).get();
sop("maximum value is:"+max);
```

## V.forEach() method

This method will not return anything.

This method will take lambda expression as argument and apply that lambda expression for each element present in the stream.

**Ex:**

```
l.stream().forEach(s->sop(s));
l3.stream().forEach(System.out:: println);
```

**Ex:**

```
1) import Java.util.*;
2) import Java.util.stream.*;
3) class Test1 {
4)     public static void main(String[] args) {
5)         ArrayList<Integer> l1 = new ArrayList<Integer>();
6)         l1.add(0); l1.add(15); l1.add(10); l1.add(5); l1.add(30); l1.add(25); l1.add(20);
7)         System.out.println(l1);
8)         ArrayList<Integer> l2=l1.stream().map(i-> i+10).collect(Collectors.toList());
9)         System.out.println(l2);
10)        long count = l1.stream().filter(i->i%2==0).count();
11)        System.out.println(count);
12)        List<Integer> l3=l1.stream().sorted().collect(Collectors.toList());
13)        System.out.println(l3);
14)        Comparator<Integer> comp=(i1,i2)->i1.compareTo(i2);
15)        List<Integer> l4=l1.stream().sorted(comp).collect(Collectors.toList());
16)        System.out.println(l4);
17)        Integer min=l1.stream().min(comp).get();
18)        System.out.println(min);
19)        Integer max=l1.stream().max(comp).get();
20)        System.out.println(max);
21)        l3.stream().forEach(i->sop(i));
22)        l3.stream().forEach(System.out:: println);
```



```
23)
24)    }
25) }
```

## VI.toArray() method

We can use **toArray()** method to copy elements present in the stream into specified array

```
Integer[] ir = l1.stream().toArray(Integer[] :: new);
for(Integer i: ir) {
    sop(i);
}
```

## VII.Stream.of()method

We can also apply a stream for group of values and for arrays.

Ex:

```
Stream s=Stream.of(99,999,9999,99999);
s.forEach(System.out:: println);
```

```
Double[] d={10.0,10.1,10.2,10.3};
Stream s1=Stream.of(d);
s1.forEach(System.out :: println);
```



# Date and Time API: (Joda-Time API)

Until Java 1.7 version the classes present in Java.util package to handle Date and Time (like Date, Calendar, TimeZone etc) are not up to the mark with respect to convenience and performance.

To overcome this problem in the 1.8 version oracle people introduced Joda-Time API. This API developed by joda.org and available in Java in the form of Java.time package.

# program for to display System Date and time.

```
1) import java.time.*;
2) public class DateTime {
3)     public static void main(String[] args) {
4)         LocalDate date = LocalDate.now();
5)         System.out.println(date);
6)         LocalTime time=LocalTime.now();
7)         System.out.println(time);
8)     }
9) }
```

#### Output:

2015-11-23  
12:39:26.587

Once we get LocalDate object we can call the following methods on that object to retrieve Day, month and year values separately.

#### Ex:

```
1) import java.time.*;
2) class Test {
3)     public static void main(String[] args) {
4)         LocalDate date = LocalDate.now();
5)         System.out.println(date);
6)         int dd = date.getDayOfMonth();
7)         int mm = date.getMonthValue();
8)         int yy = date.getYear();
9)         System.out.println(dd+"..."+mm+"..."+yy);
10)        System.out.printf("\n%d-%d-%d",dd,mm,yy);
11)    }
12) }
```

Once we get LocalTime object we can call the following methods on that object.



Ex:

```
1) import java.time.*;
2) class Test {
3)     public static void main(String[] args) {
4)         LocalTime time = LocalTime.now();
5)         int h = time.getHour();
6)         int m = time.getMinute();
7)         int s = time.getSecond();
8)         int n = time.getNano();
9)         System.out.printf("\n%d:%d:%d:%d",h,m,s,n);
10)    }
11) }
```

If we want to represent both Date and Time then we should go for LocalDateTime object.

```
LocalDateTime dt = LocalDateTime.now();
System.out.println(dt);
```

Output: 2015-11-23T12:57:24.531

We can represent a particular Date and Time by using LocalDateTime object as follows.

Ex:

```
LocalDateTime dt1 = LocalDateTime.of(1995, Month.APRIL, 28, 12, 45);
System.out.println(dt1);
```

Ex:

```
LocalDateTime dt1=LocalDateTime.of(1995, 04, 28, 12, 45);
System.out.println(dt1);
System.out.println("After six months:"+dt1.plusMonths(6));
System.out.println("Before six months:"+dt1.minusMonths(6));
```

## To Represent Zone:

ZonedDateTime object can be used to represent Zone.

Ex:

```
1) import java.time.*;
2) class ProgramOne {
3)     public static void main(String[] args) {
4)         ZoneId zone = ZoneId.systemDefault();
5)         System.out.println(zone);
6)     }
7) }
```

We can create ZoneId for a particular zone as follows



**Ex:**

```
ZonedDateTime la = ZonedDateTime.of("America/Los_Angeles");
ZonedDateTime zt = ZonedDateTime.now(la);
System.out.println(zt);
```

## **Period Object:**

Period object can be used to represent quantity of time

**Ex:**

```
LocalDate today = LocalDate.now();
LocalDate birthday = LocalDate.of(1989,06,15);
Period p = Period.between(birthday,today);
System.out.printf("age is %d year %d months %d
days",p.getYears(),p.getMonths(),p.getDays());
```

## **# write a program to check the given year is leap year or not**

```
1) import java.time.*;
2) public class Leapyear {
3)     int n = Integer.parseInt(args[0]);
4)     Year y = Year.of(n);
5)     if(y.isLeap())
6)         System.out.printf("%d is Leap year",n);
7)     else
8)         System.out.printf("%d is not Leap year",n);
9) }
```



# Java 9 Features



# Table of Contents

|                                                                |     |
|----------------------------------------------------------------|-----|
| 1) Private Methods in Interfaces .....                         | 645 |
| 2) Try With Resources Enhancements .....                       | 652 |
| 3) Diamond Operator Enhancements .....                         | 658 |
| 4) SafeVarargs Annotation Enhancements .....                   | 665 |
| 5) Factory Methods for Creating unmodifiable Collections ..... | 669 |
| 6) Stream API Enhancements .....                               | 676 |
| 7) The Java Shell (RPEL) .....                                 | 685 |
| 8) The Java Platform Module System (JPMS) .....                | 734 |
| 9) JLINK (JAVA LINKER) .....                                   | 779 |
| 10) Process API Updates .....                                  | 783 |
| 11) HTTP/2 Client .....                                        | 790 |



# Private Methods in Interfaces

## Need Of Default Methods inside interfaces:

Prior to Java 8, Every method present inside interface is always public and abstract whether we are declaring or not.

```
1) interface Prior2Java8Interf  
2) {  
3)   public void m1();  
4)   public void m2();  
5) }
```

Assume this interface is implemented by 1000s of classes and each class provided implementation for both methods.

```
1) interface Prior2Java8Interf  
2) {  
3)   public void m1();  
4)   public void m2();  
5) }  
6) class Test1 implements Prior2Java8Interf  
7) {  
8)   public void m1(){  
9)     public void m2(){  
10) }  
11) class Test2 implements Prior2Java8Interf  
12) {  
13)   public void m1(){  
14)     public void m2(){  
15) }  
16) class Test3 implements Prior2Java8Interf  
17) {  
18)   public void m1(){  
19)     public void m2(){  
20) }  
21) class Test1000 implements Prior2Java8Interf  
22) {  
23)   public void m1(){  
24)     public void m2(){  
25) }
```

It is valid because all implementation classes provided implementation for both m1() and m2().

Assume our programming requirement is we have to extend the functionality of this interface by adding a new method m3().



```
1) interface Prior2Java8Interf
2) {
3)     public void m1();
4)     public void m2();
5)     public void m3();
6) }
```

If we add new method m3() to the interface then all the implementation classes will be effected and won't be compiled,because every implementation class should implement all methods of interface.

```
1) class Test1 implements Prior2Java8Interf
2) {
3)     public void m1(){}
4)     public void m2(){}
5) }
```

CE: Test1 is not abstract and does not override abstract method m3() in PriorJava8Interf

Hence prior to java 8,it is impossible to extend the functionality of an existing interface without effecting implementation classes. JDK 8 Engineers addresses this issue and provides solution in the form of Default methods, which are also known as Defender methods or Virtual Extension Methods.

## How to Declare Default Methods inside interfaces:

In Java 8, inside interface we can define default methods with implementation as follows.

```
1) interface Java8Interf
2) {
3)     public void m1();
4)     public void m2();
5)     default void m3()
6)     {
7)         //Default Implementation
8)     }
9) }
```

Interface default methods are by-default available to all implementation classes.Based on requirement implementation class can ignore these methods or use these default methods directly or can override.

Hence the main advantage of Default Methods inside interfaces is, without effecting implementation classes we can extend functionality of interface by adding new methods (Backward compatibility).



## Need of private Methods inside interface:

If several default methods having same common functionality then there may be a chance of duplicate code (Redundant Code).

Eg:

```
1) public interface Java8DBLogging
2) {
3)     //Abstract Methods List
4)     default void logInfo(String message)
5)     {
6)         Step1: Connect to DataBase
7)         Setp2: Log Info Message
8)         Setp3: Close the DataBase connection
9)     }
10)    default void logWarn(String message)
11)    {
12)        Step1: Connect to DataBase
13)        Setp2: Log Warn Message
14)        Setp3: Close the DataBase connection
15)    }
16)    default void logError(String message)
17)    {
18)        Step1: Connect to DataBase
19)        Setp2: Log Error Message
20)        Setp3: Close the DataBase connection
21)    }
22)    default void logFatal(String message)
23)    {
24)        Step1: Connect to DataBase
25)        Setp2: Log Fatal Message
26)        Setp3: Close the DataBase connection
27)    }
28) }
```

In the above code all log methods having some common code, which increases length of the code and reduces readability. It creates maintenance problems also. In Java8 there is no solution for this.

## How to declare private Methods inside interface:

JDK 9 Engineers addresses this issue and provided private methods inside interfaces. We can separate that common code into a private method and we can call that private method from every default method which required that functionality.



```
1) public interface Java9DBLogging
2) {
3)     //Abstract Methods List
4)     default void logInfo(String message)
5)     {
6)         log(message,"INFO");
7)     }
8)     default void logWarn(String message)
9)     {
10)        log(message,"WARN");
11)    }
12)    default void logError(String message)
13)    {
14)        log(message,"ERROR");
15)    }
16)    default void logFatal(String message)
17)    {
18)        log(message,"FATAL");
19)    }
20)    private void log(String msg,String logLevel)
21)    {
22)        Step1: Connect to DataBase
23)        Step2: Log Message with the Provided logLevel
24)        Step3: Close the DataBase Connection
25)    }
26) }
```

## Demo Program for private instance methods inside interface:

private instance methods will provide code reusability for default methods.

```
1) interface Java9Interf
2) {
3)     default void m1()
4)     {
5)         m3();
6)     }
7)     default void m2()
8)     {
9)         m3();
10)    }
11)    private void m3()
12)    {
13)        System.out.println("common functionality of methods m1 & m2");
14)    }
15) }
16) class Test implements Java9Interf
17) {
18)     public static void main(String[] args)
```



```
19) {
20)     Test t = new Test();
21)     t.m1();
22)     t.m2();
23)     //t.m3(); ==>CE
24) }
25) }
```

## Output:

```
D:\java9\durga>java Test
common functionality of methods m1 & m2
common functionality of methods m1 & m2
```

Inside Java 8 interfaces, we can take public static methods also. If several static methods having some common functionality, we can separate that common functionality into a private static method and we can call that private static method from public static methods wherever it is required.

## Demo Program for private static methods:

private static methods will provide code reusability for public static methods.

```
1) interface Java9Interf
2) {
3)     public static void m1()
4)     {
5)         m3();
6)     }
7)     public static void m2()
8)     {
9)         m3();
10)    }
11)    private static void m3()
12)    {
13)        System.out.println("common functionality of methods m1 & m2");
14)    }
15) }
16) class Test implements Java9Interf
17) {
18)     public static void main(String[] args)
19)     {
20)         Java9Interf.m1();
21)         Java9Interf.m2();
22)     }
23) }
```

## Output:

```
D:\durga_classes>java Test
common functionality of methods m1 & m2
```



---

common functionality of methods m1 & m2

**Note:** Interface static methods should be called by using interface name only even in implementation classes also.

## Advantages of private Methods inside interfaces:

The main advantages of private methods inside interfaces are:

1. Code Reusability
2. We can expose only intended methods to the API clients (Implementation classes), because interface private methods are not visible to the implementation classes.

**Note:**

1. private methods cannot be abstract and hence compulsory private methods should have the body.
2. private method inside interface can be either static or non-static.

## JDK 7 vs JDK 8 vs JDK9:

1. Prior to java 8, we can declare only public-abstract methods and public-static-final variables inside interfaces.

```
1) interface Prior2Java8Interface  
2) {  
3)   public-static-final variables  
4)   public-abstract methods  
5) }
```

2. In Java 8 ,we can declare default and public-static methods also inside interface.

```
1) interface Java8Interface  
2) {  
3)   public-static-final variables  
4)   public-abstract methods  
5)   default methods with implementation  
6)   public static methods with implementation  
7) }
```

3. In Java 9, We can declare private instance and private static methods also inside interface.

```
1) interface Java9Interface  
2) {  
3)   public-static-final variables  
4)   public-abstract methods  
5)   default methods with implementation  
6)   public static methods with implementation
```



- 7) **private** instance methods with implementation
- 8) **private static** methods with implementation
- 9) }

**Note:** The main advantage of private methods inside interface is Code Reusability without effecting implemenation classes.



# Try with Resources Enhancements

## Need of Try with Resources:

Until 1.6 version it is highly recommended to write finally block to close all resources which are open as part of try block.

```
1) BufferedReader br=null;
2) try
3) {
4)   br=new BufferedReader(new FileReader("abc.txt"));
5)   //use br based on our requirements
6) }
7) catch(IOException e)
8) {
9)   // handling code
10) }
11) finally
12) {
13)   if(br != null)
14)     br.close();
15) }
```

## Problems in this Approach:

1. Compulsory programmer is required to close all opened resources which increases the complexity of the programming
2. Compulsory we should write finally block explicitly which increases length of the code and reduces readability.

To overcome these problems Sun People introduced "try with resources" in 1.7 version.

## Try with Resources:

The main advantage of "try with resources" is the resources which are opened as part of try block will be closed automatically Once the control reaches end of the try block either normally or abnormally and hence we are not required to close explicitly so that the complexity of programming will be reduced. It is not required to write finally block explicitly and hence length of the code will be reduced and readability will be improved.

```
1) try(BufferedReader br=new BufferedReader(new FileReader("abc.txt")))
2) {
3)   use be based on our requirement, br will be closed automatically , One control reaches
   end of try either normally or abnormally and we are not required to close explicitly
4) }
5) catch(IOException e)
```



```
6) {  
7) // handling code  
8) }
```

## Conclusions:

1. We can declare any number of resources but all these resources should be separated with ; (semicolon)

```
1) try(R1 ; R2 ; R3)  
2) {  
3) -----  
4) }
```

2. All resources should be AutoCloseable resources. A resource is said to be auto closable if and only if the corresponding class implements the *java.lang.AutoCloseable* interface either directly or indirectly.

All database related, network related and file io related resources already implemented AutoCloseable interface. Being a programmer we should aware this point and we are not required to do anything extra.

3. AutoCloseable interface introduced in Java 1.7 Version and it contains only one method: close()

4. All resource reference variables should be final or effectively final and hence we can't perform reassignment within the try block.

```
1) try(BufferedReader br=new BufferedReader(new FileReader("abc.txt")))  
2) {  
3)     br=new BufferedReader(new FileReader("abc.txt"));  
4) }
```

CE : Can't reassign a value to final variable br

5. Until 1.6 version try should be followed by either catch or finally but in 1.7 version we can take only try with resource without catch or finally

```
1) try(R)  
2) {  
3)     //valid  
4) }
```

6. The main advantage of "try with resources" is finally block will become dummy because we are not required to close resources of explicitly.



## Problems with JDK 7 Try with Resources:

1. The resource reference variables which are created outside of try block cannot be used directly in try with resources.

```
1) BufferedReader br=new BufferedReader(new FileReader("abc.txt"))
2) try(br)
3) {
4)     // Risky code
5) }
```

This syntax is invalid in until java 1.8V.

We should create the resource in try block primary list or we should declare with new reference variable in try block. i.e. Resource reference variable should be local to try block.

## Solution-1: Creation of Resource in try block primary list

```
1) try(BufferedReader br=new BufferedReader(new FileReader("abc.txt")))
2) {
3)     // It is valid syntax in java 1.8V
4) }
```

## Solution-2: Assign resource with new reference variable

```
1) BufferedReader br=new BufferedReader(new FileReader("abc.txt"))
2) try(BufferedReader br1=br)
3) {
4)     // It is valid syntax in java 1.8V
5) }
```

But from JDK 9 onwards we can use the resource reference variables which are created outside of try block directly in try block resources list. i.e. The resource reference variables need not be local to try block.

```
1) BufferedReader br=new BufferedReader(new FileReader("abc.txt"))
2) try(br)
3) {
4)     // It is valid in JDK 9 but in valid until JDK 1.8V
5) }
```

But make sure resource(br) should be either final or effectively final. Effectively final means we should not perform reassignment.

This enhancement reduces length of the code and increases readability.



```
1) MyResource r1 = new MyResource();  
2) MyResource r2 = new MyResource();  
3) MyResource r3 = new MyResource();  
4) try(r1,r2,r3)  
5) {  
6) }
```

## Demo Program:

```
1) class MyResource implements AutoCloseable  
2) {  
3)     MyResource()  
4)     {  
5)         System.out.println("Resource Creation...");  
6)     }  
7)     public void doProcess()  
8)     {  
9)         System.out.println("Resource Processing...");  
10)    }  
11)    }  
12)    public void close()  
13)    {  
14)        System.out.println("Resource Closing...");  
15)    }  
16) }  
17) class Test  
18) {  
19)     public static void preJDK7()  
20)     {  
21)         MyResource r=null;  
22)         try  
23)         {  
24)             r=new MyResource();  
25)             r.doProcess();  
26)         }  
27)         catch (Exception e)  
28)         {  
29)             System.out.println("Handling:"+e);  
30)         }  
31)         finally  
32)         {  
33)             try  
34)             {  
35)                 if (r!=null)  
36)                 {  
37)                     r.close();  
38)                 }  
39)             }  
40)         catch (Exception e)
```



```
41)      {
42)          System.out.println("Handling:"+e);
43)      }
44)  }
45) }
46) public static void JDK7()
47) {
48)     try(MyResource r=new MyResource())
49)     {
50)         r.doProcess();
51)     }
52)     catch(Exception e)
53)     {
54)         System.out.println("Handling:"+e);
55)     }
56) }
57) public static void JDK9()
58) {
59)     MyResource r= new MyResource();
60)     try(r)
61)     {
62)         r.doProcess();
63)     }
64)     catch(Exception e)
65)     {
66)         System.out.println("Handling:"+e);
67)     }
68) }
69) public static void multipleJDK9()
70) {
71)     MyResource r1= new MyResource();
72)     MyResource r2= new MyResource();
73)     MyResource r3= new MyResource();
74)     MyResource r4= new MyResource();
75)     try(r1;r2;r3;r4)
76)     {
77)         r1.doProcess();
78)         r2.doProcess();
79)         r3.doProcess();
80)         r4.doProcess();
81)     }
82)     catch(Exception e)
83)     {
84)         System.out.println("Handling:"+e);
85)     }
86) }
87) public static void main(String[] args)
88) {
89)     System.out.println("Program Execution With PreJDK7");
90)     preJDK7();
```



```
91)
92)     System.out.println("Program Execution With JDK7");
93)     JDK7();
94)
95)     System.out.println("Program Execution With JDK9");
96)     JDK9();
97)     System.out.println("Program Execution Multiple Resources With JDK9");
98)     multipleJDK9();
99)
100)    }
```

## Output:

Program Execution With PreJDK7  
Resource Creation...  
Resource Processing...  
Resource Closing...  
Program Execution With JDK7  
Resource Creation...  
Resource Processing...  
Resource Closing...  
Program Execution With JDK9  
Resource Creation...  
Resource Processing...  
Resource Closing...  
Program Execution Multiple Resources With JDK9  
Resource Creation...  
Resource Creation...  
Resource Creation...  
Resource Creation...  
Resource Processing...  
Resource Processing...  
Resource Processing...  
Resource Processing...  
Resource Closing...  
Resource Closing...  
Resource Closing...  
Resource Closing...



# Diamond Operator Enhancements

This enhancement is as the part of Milling Project Coin (JEP 213).

Before understanding this enhancement, we should aware Generics concept, which has been introduced in java 1.5 version.

The main objectives of Generics are:

1. To provide Type Safety
2. To resolve Type Casting Problems.

## Case 1: Type-Safety

Arrays are always type safe. i.e we can give the guarantee for the type of elements present inside array. For example if our programming requirement is to hold String type of objects, it is recommended to use String array. For the string array we can add only string type of objects. By mistake if we are trying to add any other type we will get compile time error.

Eg:

- 1) `String[] s = new String[100];`
- 2) `s[0] = "Durga";`
- 3) `s[1] = "Pavan";`
- 4) `s[2] = new Integer(10); //error: incompatible types: Integer cannot be converted to String`

`String[]` can contains only String type of elements. Hence, we can always give guarantee for the type of elements present inside array. Due to this arrays are safe to use with respect to type. Hence arrays are type safe.

But collections are not type safe that is we can't give any guarantee for the type of elements present inside collection. For example if our programming requirement is to hold only string type of objects, and if we choose ArrayList, by mistake if we are trying to add any other type we won't get any compile time error but the program may fail at runtime.

- 1) `ArrayList l = new ArrayList();`
- 2) `l.add("Durga");`
- 3) `l.add("Pavan");`
- 4) `l.add(new Integer(10));`
- 5) `....`
- 6) `String name1=(String)l.get(0);`
- 7) `String name2=(String)l.get(1);`
- 8) `String name3=(String)l.get(2); //RE: java.lang.ClassCastException`

Hence we can't give any guarantee for the type of elements present inside collections. Due to this collections are not safe to use with respect to type, i.e collections are not Type-Safe.



## Case 2: Type-Casting

In the case of array at the time of retrieval it is not required to perform any type casting.

- 1) String[] s = new String[100];
- 2) s[0] = "Durga";
- 3) s[1] = "Pavan";
- 4) ...
- 5) String name1=s[0];//Type casting is not required.

But in the case of collection at the time of retrieval compulsory we should perform type casting otherwise we will get compile time error.

Eg:

- 1) ArrayList l = new ArrayList();
- 2) l.add("Durga");
- 3) l.add("Pavan");
- 4) ....
- 5) String name1=l.get(0);//error: incompatible types: Object cannot be converted to String
- 6) String name1=(String)l.get(0);//valid

Hence in Collections Type Casting is bigger headache.

To overcome the above problems of collections(type-safety, type casting),Java people introduced Generics concept in 1.5version.Hence the main objectives of Generics are:

1. To provide Type Safety to the collections.
2. To resolve Type casting problems.

## Example for Generic Collection:

To hold only string type of objects, we can create a generic version of ArrayList as follows.

ArrayList<String> l = new ArrayList<String>();

Here String is called Parameter Type.

For this ArrayList we can add only string type of objects. By mistake if we are trying to add any other type then we will get compile time error.

- 1) l.add("Durga");//valid
- 2) l.add("Ravi");//valid
- 3) l.add(new Integer(10));//error: no suitable method found for add(Integer)

Hence, through generics we are getting type safety.

At the time of retrieval it is not required to perform any type casting we can assign elements directly to string type variables.



`String name1 = l.get(0); // valid and Type Casting is not required.`

Hence, through generic syntax we can resolve type casting problems.

## Difference between Generic and Non-Generic Collections

|                                                                                        |                                                                                                                                                      |
|----------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>ArrayList l = new ArrayList();</code>                                            | <code>ArrayList&lt;String&gt; l = new ArrayList&lt;String&gt;();</code>                                                                              |
| <b>It is Non Generic version of ArrayList</b>                                          | <b>It is Generic version of ArrayList</b>                                                                                                            |
| <b>For this ArrayList we can add any type of object and hence it is not Type-Safe.</b> | <b>For this ArrayList we can add only String type of Objects. By mistake if we are trying to add any other type, we will get compile time error.</b> |
| <b>At the time of retrieval compulsory we should perform Type casting.</b>             | <b>At the time of retrieval, we are not required to perform Type casting.</b>                                                                        |

## Java 7 Diamond Operator (<>):

Diamond Operator '<>' was introduced in JDK 7 under project Coin.

The main objective of Diamond Operator is to instantiate generic classes very easily.

Prior to Java 7, Programmer compulsory should explicitly include the Type of generic class in the Type Parameter of the constructor.

`ArrayList<String> l = new ArrayList<String>();`

Whenever we are using Diamond Operator, then the compiler will consider the type automatically based on context, Which is also known as Type inference. We are not required to specify Type Parameter of the Constructor explicitly.

`ArrayList<String> l = new ArrayList<>();`

Hence the main advantage of Diamond Operator is we are not required to speicfy the type parameter in the constructor explicitly, length of the code will be reduced and readability will be improved.

Eg 2:

`List<Map<String, Integer>> l = new ArrayList<Map<String, Integer>>();`

can be wrritten with Diamond operator as follows

`List<Map<String, Integer>> l = new ArrayList<>();`

But until Java 8 version we cannot apply diamond operator for Anonymous Generic classes. But in Java 9, we can use.



## Usage of Diamond Operator for Anonymous Classes:

In JDK 9, Usage of Diamond Operator extended to Anonymous classes also.

### Anonymous class:

Sometimes we can declare classes without having the name, such type of nameless classes are called **Anonymous Classes**.

#### Eg 1:

```
1) Thread t = new Thread()  
2) {  
3) };
```

We are creating a child class that extends Thread class without name(Anonymous class) and we are creating object for that child class.

#### Eg 2:

```
1) Runnable r = new Runnable()  
2) {  
3) };
```

We are creating an implementation class for Runnable interface without name(Anonymous class) and we are creating object for that implementation class.

#### Eg 3:

```
1) ArrayList<String> l = new ArrayList<String>()  
2) {  
3) };
```

We are creating a child class that extends ArrayList class without name(Anonymous class) and we are creating object for that child class.

From JDK 9 onwards we can use Diamond Operator for Anonymous Classes also.

```
1) ArrayList<String> l = new ArrayList<>()  
2) {  
3) };
```

It is valid in Java 9 but invalid in Java 8.



## Demo Program - 1: To demonstrate usage of diamond operator for Anonymous Class

```
1) class MyGenClass<T>
2) {
3)     T obj;
4)     public MyGenClass(T obj)
5)     {
6)         this.obj = obj;
7)     }
8)
9)     public T getObj()
10)    {
11)        return obj;
12)    }
13)    public void process()
14)    {
15)        System.out.println("Processing obj...");
16)    }
17}
18) public class Test
19) {
20)     public static void main(String[] args)
21)     {
22)         MyGenClass<String> c1 = new MyGenClass<String>("Durga")
23)         {
24)             public void process()
25)             {
26)                 System.out.println("Processing... " + getObj());
27)             }
28)         };
29)         c1.process();
30)
31)         MyGenClass<String> c2 = new MyGenClass<>("Pavan")
32)         {
33)             public void process()
34)             {
35)                 System.out.println("Processing... " + getObj());
36)             }
37)         };
38)         c2.process();
39)     }
40} }
```

### Output:

Processing... Durga  
Processing... Pavan



If we compile the above program according to Java 1.8 version, then we will get compile time error

D:\durga\_classes>javac -source 1.8 Test.java

error: cannot infer type arguments for MyGenClass<T>

    MyGenClass<String> c2 = new MyGenClass<>("Pavan")

    ^

reason: cannot use '<>' with anonymous inner classes in -source 1.8

(use -source 9 or higher to enable '<>' with anonymous inner classes)

## Demo Program - 2: To demonstrate usage of diamond operator for Anonymous Class

```
1) import java.util.Iterator;
2) import java.util.NoSuchElementException;
3) public class DiamondOperatorDemo
4) {
5)     public static void main(String[] args)
6)     {
7)         String[] animals = { "Dog", "Cat", "Rat", "Tiger", "Elephant" };
8)         Iterator<String> iter = new Iterator<>()
9)         {
10)             int i = 0;
11)             public boolean hasNext()
12)             {
13)                 return i < animals.length;
14)             }
15)             public String next()
16)             {
17)                 if (!hasNext())
18)                     throw new NoSuchElementException();
19)                 return animals[i++];
20)             }
21)         };
22)         while (iter.hasNext())
23)         {
24)             System.out.println(iter.next());
25)         }
26)     }
27} }
```

### Output:

Dog  
Cat  
Rat  
Tiger  
Elephant



## Note - 1:

```
1) ArrayList<String> preJava7 = new ArrayList<String>();  
2) ArrayList<String> java7 = new ArrayList<>();  
3) ArrayList<String> java9 = new ArrayList<>()  
4) {  
5) };
```

## Note - 2:

Be Ready for Partial Diamond Operator in the next versions of Java, as the part of Project "Amber". Open JDK people already working on this.

Eg: new Test<String, \_>();



# SafeVarargs Annotation Enhancements

This SafeVarargs Annotation was introduced in Java 7.

Prior to Java 9, we can use this annotation for final methods, static methods and constructors. But from Java 9 onwards we can use for private methods also.

To understand the importance of this annotation, first we should aware var-arg methods and heap pollution problem.

## What is var-arg method?

Until 1.4 version, we can't declared a method with variable number of arguments. If there is a change in no of arguments compulsory we have to define a new method. This approach increases length of the code and reduces readability.

But from 1.5 version onwards, we can declare a method with variable number of arguments, such type of methods are called var-arg methods.

```
1) public class Test
2) {
3)     public static void m1(int... x)
4)     {
5)         System.out.println("var-arg method");
6)     }
7)     public static void main(String[] args)
8)     {
9)         m1();
10)        m1(10);
11)        m1(10,20,30);
12)    }
13) }
```

## Output

var-arg method  
var-arg method  
var-arg method

Internally var-arg parameter will be converted into array.

```
1) public class Test
2) {
3)     public static void sum(int... x)
4)     {
5)         int total=0;
6)         for(int x1 : x)
7)         {
8)             total=total+x1;
9)         }
}
```



```
10)    System.out.println("The Sum:"+ total);
11) }
12) public static void main(String[] args)
13) {
14)    sum();
15)    sum(10);
16)    sum(10,20,30);
17) }
18) }
```

## Output

The Sum:0  
The Sum:10  
The Sum:60

## Var-arg method with Generic Type:

If we use var-arg methods with Generic Type then there may be a chance of Heap Pollution.  
At runtime if one type variable trying to point to another type value, then there may be a chance of ClassCastException. This problem is called Heap Pollution.  
In our code, if there is any chance of heap pollution then compiler will generate warnings.

```
1) import java.util.*;
2) public class Test
3) {
4)    public static void main(String[] args)
5)    {
6)       List<String> l1= Arrays.asList("A","B");
7)       List<String> l2= Arrays.asList("C","D");
8)       m1(l1,l2);
9)    }
10)   public static void m1(List<String>... l)//argument will become List<String>[]
11)   {
12)      Object[] a = l;// we can assign List[] to Object[]
13)      a[0]=Arrays.asList(10,20);
14)      String name=(String)l[0].get(0);//String type pointing to Integer type
15)      System.out.println(name);
16)   }
17) }
```

## Compilation:

javac Test.java

Note: Test.java uses unchecked or unsafe operations.



**Note:** Recompile with -Xlint:unchecked for details.

```
javac -Xlint:unchecked Test.java
```

```
warning: [unchecked] unchecked generic array creation for varargs parameter of type
```

```
List<String>[]
```

```
    m1(I1,I2);
```

```
    ^
```

```
warning: [unchecked] Possible heap pollution from parameterized vararg type List<String>
```

```
    public static void m1(List<String>... I)
```

```
    ^
```

2 warnings

## Execution:

```
java Test
```

```
RE: java.lang.ClassCastException: java.base/java.lang.Integer cannot be cast to  
java.base/java.lang.String
```

In the above program at runtime, String type variable name is trying to point to Integer type, which causes Heap Pollution and results ClassCastException.

```
String name = (String)I[0].get(0);
```

## Need of @SafeVarargs Annotation:

Very few Var-arg Methods cause Heap Pollution, not all the var-arg methods. If we know that our method won't cause Heap Pollution, then we can suppress compiler warnings with `@SafeVarargs` annotation.

```
1) import java.util.*;  
2) public class Test  
3) {  
4)     public static void main(String[] args)  
5)     {  
6)         List<String> I1= Arrays.asList("A","B");  
7)         List<String> I2= Arrays.asList("C","D");  
8)         m1(I1,I2);  
9)     }  
10)    @SafeVarargs  
11)    public static void m1(List<String>... I)  
12)    {  
13)        for(List<String> I1: I)  
14)        {  
15)            System.out.println(I1);  
16)        }  
17)    }  
18) }
```



## Output:

[A, B]  
[C, D]

In the program, inside m1() method we are not performing any reassessments. Hence there is no chance of Heap Pollution Problem. Hence we can suppress Compiler generated warnings with @SafeVarargs annotation.

Note: At compile time observe the difference with and without SafeVarargs Annotation.

## Java 9 Enhancements to @SafeVarargs Annotation:

@SafeVarargs Annotation introduced in Java 7.

Until Java 8, this annotation is applicable only for static methods, final methods and constructors. But from Java 9 onwards, we can also use for private instance methods also.

```
1) import java.util.*;
2) public class Test
3) {
4)     @SafeVarargs //valid
5)     public Test(List<String>... l)
6)     {
7)     }
8)     @SafeVarargs //valid
9)     public static void m1(List<String>... l)
10)    {
11)    }
12)    @SafeVarargs //valid
13)    public final void m2(List<String>... l)
14)    {
15)    }
16)    @SafeVarargs //valid in Java 9 but not in Java 8
17)    private void m3(List<String>... l) {
18)    }
19) }
```

javac -source 1.8 Test.java

error: Invalid SafeVarargs annotation. Instance method m3(List<String>...) is not final.  
private void m3(List<String>... l)  
^

javac -source 1.9 Test.java

We won't get any compile time error.

## FAQs:

Q1. For which purpose we can use @SafeVarargs annotation?

Q2. What is Heap Pollution?



## Factory Methods for creating unmodifiable Collections

**List:** An indexed Collection of elements where duplicates are allowed and insertion order is preserved.

**Set:** An unordered Collection of elements where duplicates are not allowed and insertion order is not preserved.

**Map:** A Map is a collection of key-value pairs and each key-value pair is called Entry. Entry is an inner interface present inside Map interface. Duplicate keys are not allowed, but values can be duplicated.

As the part of programming requirement, it is very common to use Immutable Collection objects to improve Memory utilization and performance.

Prior to Java 9, we can create unmodifiable Collection objects as follows

### Eg 1: Creation of unmodifiable List object

```
1) List<String> beers=new ArrayList<String>();  
2) beers.add("KF");  
3) beers.add("FO");  
4) beers.add("RC");  
5) beers.add("FO");  
6) beers =Collections.unmodifiableList(beers);
```

### Eg 2: Creation of unmodifiable Set Object

```
1) Set<String> beers=new HashSet<String>();  
2) beers.add("KF");  
3) beers.add("KO");  
4) beers.add("RC");  
5) beers.add("FO");  
6) beers =Collections.unmodifiableSet(beers);
```

### Eg 3: Creation of unmodifiable Map object

```
1) Map<String,String> map=new HashMap<String,String>();  
2) map.put("A", "Apple");  
3) map.put("B", "Banana");  
4) map.put("C", "Cat");  
5) map.put("D", "Dog");  
6) map =Collections.unmodifiableMap(map);
```

This way of creating unmodifiable Collections is verbose and not convenient. It increases length of the code and reduces readability.



JDK Engineers addresses this problem and introduced several factory methods for creating unmodifiable collections.

## Creation of unmodifiable List (Immutable List) with Java 9 Factory Methods:

Java 9 List interface defines several factory methods for this.

1. static <E> List<E> of()
2. static <E> List<E> of(E e1)
3. static <E> List<E> of(E e1, E e2)
4. static <E> List<E> of(E e1, E e2, E e3)
5. static <E> List<E> of(E e1, E e2, E e3, E e4)
6. static <E> List<E> of(E e1, E e2, E e3, E e4, E e5)
7. static <E> List<E> of(E e1, E e2, E e3, E e4, E e5, E e6)
8. static <E> List<E> of(E e1, E e2, E e3, E e4, E e5, E e6, E e7)
9. static <E> List<E> of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8)
10. static <E> List<E> of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9)
11. static <E> List<E> of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9, E e10)
12. static <E> List<E> of(E... elements)

Upto 10 elements the matched method will be executed and for more than 10 elements internally var-arg method will be called. JDK Engineers identified List of upto 10 elements is the common requirement and hence they provided the corresponding methods. For remaining cases var-arg method will be executed, which is very costly. These many methods just to improve performance.

**Eg:** To create unmodifiable List with Java 9 Factory Methods.

```
List<String> beers = List.of("KF", "KO", "RC", "FO");
```

It is very simple and straight forward way.

### Note:

1. While using these factory methods if any element is null then we will get NullPointerException.

```
List<String> fruits = List.of("Apple", "Banana", null); ➔ NullPointerException
```

2. After creating the List object, if we are trying to change the content (add | remove | replace elements) then we will get UnsupportedOperationException because List is immutable (unmodifiable).

```
List<String> fruits = List.of("Apple", "Banana", "Mango");
fruits.add("Orange"); //UnsupportedOperationException
fruits.remove(1); //UnsupportedOperationException
fruits.set(1, "Orange"); //UnsupportedOperationException
```



## Creation of unmodifiable Set(Immutable Set) with Java 9 Factory Methods:

Java 9 Set interface defines several factory methods for this.

1. static <E> Set<E> of()
2. static <E> Set<E> of(E e1)
3. static <E> Set<E> of(E e1, E e2)
4. static <E> Set<E> of(E e1, E e2, E e3)
5. static <E> Set<E> of(E e1, E e2, E e3, E e4)
6. static <E> Set<E> of(E e1, E e2, E e3, E e4, E e5)
7. static <E> Set<E> of(E e1, E e2, E e3, E e4, E e5, E e6)
8. static <E> Set<E> of(E e1, E e2, E e3, E e4, E e5, E e6, E e7)
9. static <E> Set<E> of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8)
10. static <E> Set<E> of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9)
11. static <E> Set<E> of(E e1, E e2, E e3, E e4, E e5, E e6, E e7, E e8, E e9, E e10)
12. static <E> Set<E> of(E... elements)

**Eg:** To create unmodifiable Set with Java 9 Factory Methods.

```
Set<String> beers = Set.of("KF", "KO", "RC", "FO");
```

### Note:

1. While using these Factory Methods if we are trying to add duplicate elements then we will get **IllegalArgumentException**, because Set won't allow duplicate elements

```
Set<Integer> numbers=Set.of(10,20,30,10);  
RE: IllegalArgumentException: duplicate element: 10
```

2. While using these factory methods if any element is null then we will get **NullPointerException**.

```
Set<String> fruits=Set.of("Apple", "Banana", null); ➔ NullPointerException
```

3. After creating the Set object, if we are trying to change the content (add | remove elements) then we will get **UnsupportedOperationException** because Set is immutable(unmodifiable).

```
Set<String> fruits=Set.of("Apple", "Banana", "Mango");  
fruits.add("Orange"); //UnsupportedOperationException  
fruits.remove("Apple"); //UnsupportedOperationException
```



## Creation of unmodifiable Map (Immutable Map) with Java 9 Factory Methods:

Java 9 Map interface defines of() and ofEntries() Factory methods for this purpose.

1. static <K,V> Map<K,V> of()
2. static <K,V> Map<K,V> of(K k1,V v1)
3. static <K,V> Map<K,V> of(K k1,V v1,K k2,V v2)
4. static <K,V> Map<K,V> of(K k1,V v1,K k2,V v2,K k3,V v3)
5. static <K,V> Map<K,V> of(K k1,V v1,K k2,V v2,K k3,V v3,K k4,V v4)
6. static <K,V> Map<K,V> of(K k1,V v1,K k2,V v2,K k3,V v3,K k4,V v4,K k5,V v5)
7. static <K,V> Map<K,V> of(K k1,V v1,K k2,V v2,K k3,V v3,K k4,V v4,K k5,V v5,K k6,V v6)
8. static <K,V> Map<K,V> of(K k1,V v1,K k2,V v2,K k3,V v3,K k4,V v4,K k5,V v5,K k6,V v6,K k7,V v7)
9. static <K,V> Map<K,V> of(K k1,V v1,K k2,V v2,K k3,V v3,K k4,V v4,K k5,V v5,K k6,V v6,K k7,V v7,K k8,V v8)
10. static <K,V> Map<K,V> of(K k1,V v1,K k2,V v2,K k3,V v3,K k4,V v4,K k5,V v5,K k6,V v6,K k7,V v7,K k8,V v8,K k9,V v9)
11. static <K,V> Map<K,V> of(K k1,V v1,K k2,V v2,K k3,V v3,K k4,V v4,K k5,V v5,K k6,V v6,K k7,V v7,K k8,V v8,K k9,V v9,K k10,V v10)
12. static <K,V> Map<K,V> ofEntries(Map.Entry<? extends K,? extends V>... entries)

### Note:

Up to 10 entries, it is recommended to use of() methods and for more than 10 items we should use ofEntries() method.

Eg: `Map<String, String> map=Map.of("A","Apple","B","Banana","C","Cat","D","Dog");`

### How to use Map.ofEntries() method:

Map interface contains static Method entry() to create immutable Entry objects.

`Map.Entry<String, String> e=Map.entry("A","Apple");`

This Entry object is immutable and we cannot modify its content. If we are trying to change we will get RE: UnsupportedOperationException

`e.setValue("Durga");` ➔ UnsupportedOperationException

By using these Entry objects we can create unmodifiable Map object with Map.ofEntries() method.

### Eg:

```
1) import java.util.*;  
2) class Test  
3) {  
4)     public static void main(String[] args)  
5)     {  
6)         Map.Entry<String, String> e1=Map.entry("A", "Apple");
```



```
7) Map.Entry<String, String> e2=Map.entry("B","Banana");
8) Map.Entry<String, String> e3=Map.entry("C","Cat");
9) Map<String, String> m=Map.ofEntries(e1,e2,e3);
10) System.out.println(m);
11)
12}
```

In Short way we can also create as follows.

```
1) import static java.util.Map.entry;
2) Map<String, String> map=Map.ofEntries(entry("A","Apple"),entry("B","Banana"),entry("C",
,"Cat"),entry("D","Dog"));
```

### Note:

1. While using these Factory Methods if we are trying to add duplicate keys then we will get **IllegalArgumentException**: duplicate key. But values can be duplicated.

```
Map<String, String> map=Map.of("A","Apple","A","Banana","C","Cat","D","Dog");
RE: java.lang.IllegalArgumentException: duplicate key: A
```

2. While using these factory methods if any element is null (either key or value) then we will get **NullPointerException**.

```
Map<String, String> map=Map.of("A",null,"B","Banana"); ==>NullPointerException
Map<String, String> map=Map.ofEntries(entry(null,"Apple"),entry("B","Banana"));
→ NullPointerException
```

3. After creating the Map object, if we are trying to change the content(add|remove|replace elements) then we will get **UnsupportedOperationException** because Map is immutable(unmodifiable).

### Eg:

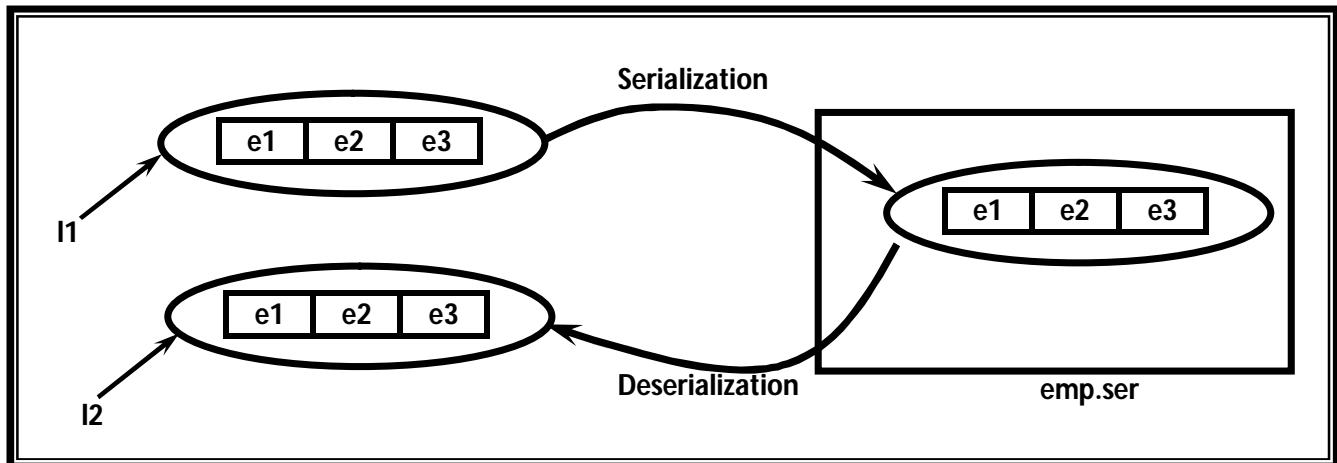
```
Map<String, String> map=Map.ofEntries(entry("A","Apple"),entry("B","Banana"));
map.put("C","Cat"); → UnsupportedOperationException
map.remove("A"); → UnsupportedOperationException
```



## Serialization for unmodifiable Collections:

The immutable collection objects are serializable iff all elements are serializable.

In Brief form, the process of writing state of an object to a file is called **Serialization** and the process of reading state of an object from the file is called **Deserialization**.



```
1) import java.util.*;
2) import java.io.*;
3) class Employee implements Serializable
4) {
5)     private int eno;
6)     private String ename;
7)     Employee(int eno, String ename)
8)     {
9)         this.eno=eno;
10)        this.ename=ename;
11)    }
12)    public String toString()
13)    {
14)        return String.format("%d=%s", eno, ename);
15)    }
16) }
17) class Test
18) {
19)     public static void main(String[] args) throws Exception
20)     {
21)         Employee e1= new Employee(100, "Sunny");
22)         Employee e2= new Employee(200, "Bunny");
23)         Employee e3= new Employee(300, "Chinny");
24)         List<Employee> l1=List.of(e1, e2, e3);
25)         System.out.println(l1);
26)
27)         System.out.println("Serialization of List Object... ");
28)         FileOutputStream fos=new FileOutputStream("emp.ser");
29)         ObjectOutputStream oos=new ObjectOutputStream(fos);
30)         oos.writeObject(l1);
```



```
31)
32) System.out.println("Deserialization of List Object...");
33) FileInputStream fis=new FileInputStream("emp.ser");
34) ObjectInputStream ois=new ObjectInputStream(fis);
35) List<Employee> l2=(List<Employee>)ois.readObject();
36) System.out.println(l2);
37) //l2.add(new Employee(400,"Vinnny")); //UnsupportedOperationException
38}
39}
```

**Output:**

```
D:\durga_classes>java Test
[100=Sunny, 200=Bunny, 300=Chinny]
Serialization of List Object...
Deserialization of List Object...
[100=Sunny, 200=Bunny, 300=Chinny]
```

After deserialization also we cannot modify the content, otherwise we will get UnsupportedOperationException.

**Note:** The Factory Methods introduced in Java 9 are not to create general collections and these are meant for creating immutable collections.



# Stream API Enhancements

Streams concept has been introduced in Java 1.8 version.

The main objective of Streams concept is to process elements of Collection with Functional Programming (Lambda Expressions).

### What is the difference between java.util streams and java.io streams?

java.util streams meant for processing objects from the collection. i.e. it represents a stream of objects from the collection but java.io streams meant for processing binary and character data with respect to file. i.e it represents stream of binary data or character data from the file. Hence java.io streams and java.util streams are different concepts.

### What is the difference between Collections and Streams?

If we want to represent a group of individual objects as a single entity, then we should go for collection. If we want to process a group of objects from the collection then we should go for Streams.

### How to Create Stream Object?

We can create stream object to the collection by using stream() method of Collection interface. stream() method is a default method added to the Collection in 1.8 version.

```
default Stream stream()
```

Ex: Stream s = c.stream(); // c is any Collection object

Note: Stream is an interface present in java.util.stream package.

### How to process Objects of Collection By using Stream:

Once we got the stream, by using that we can process objects of that collection. For this we can use either filter() method or map() method.

#### Processing Objects by using filter() method:

We can use filter() method to filter elements from the collection based on some boolean condition.

```
public Stream filter(Predicate<T> t)
```

Here (Predicate<T > t ) can be a boolean valued function/lambda expression



## Demo Program:

```
1) import java.util.*;
2) import java.util.stream.*;
3) public class Test
4) {
5)     public static void main(String[] args)
6)     {
7)         ArrayList<Integer> l1 = new ArrayList<Integer>();
8)         for(int i=0; i <=10; i++)
9)         {
10)             l1.add(i);
11)         }
12)         System.out.println("Before Filtering:"+l1);
13)         List<Integer> l2=l1.stream().filter(i->i%2==0).collect(Collectors.toList());
14)         System.out.println("After Filtering:"+l2);
15)     }
16) }
```

## Output:

Before Filtering:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

After Filtering:[0, 2, 4, 6, 8, 10]

## Processing Objects by using map() method:

If we want to create a separate new object, for every object present in the collection based on our requirement then we should go for map() method of Stream interface.

**public Stream map (Function f);**

The argument can be lambda expression also

## Demo Program:

```
1) import java.util.*;
2) import java.util.stream.*;
3) public class Test
4) {
5)     public static void main(String[] args) {
6)         ArrayList<Integer> l1 = new ArrayList<Integer>();
7)         for(int i=0; i <=10; i++)
8)         {
9)             l1.add(i);
10)         }
11)         System.out.println("Before using map() method:"+l1);
12)         List<Integer> l2=l1.stream().map(i->i*i).collect(Collectors.toList());
13)         System.out.println("After using map() method:"+l2);
14)     }
15) }
```



## Output:

Before using map() method:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

After using map() method:[0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

## Processing Objects by using flatMap() method:

Both map and flatMap can be applied to a Stream<T> and they both return a Stream<R>. The difference is that the map operation produces one output value for each input value, whereas the flatMap operation produces an arbitrary number (zero or more) values for each input value.

Typical use is for the mapper function of flatMap to return Stream.empty() if it wants to send zero values, or something like Stream.of(x, y, z) if it wants to return several values.

### Demo Program:

```
1) import java.util.*;
2) import java.util.stream.*;
3) public class Test
4) {
5)     public static void main(String[] args)
6)     {
7)         ArrayList<Integer> l1 = new ArrayList<Integer>();
8)         for(int i=0; i <=10; i++)
9)         {
10)             l1.add(i);
11)         }
12)         System.out.println("Before using map() method:"+l1);
13)         List<Integer> l2=l1.stream().flatMap(
14)             i->{ if (i%2 !=0) return Stream.empty();
15)                 else return Stream.of(i);
16)             }).collect(Collectors.toList());
17)         System.out.println("After using flatMap() method:"+l2);
18)
19)         List<Integer> l3=l1.stream().flatMap(
20)             i->{ if (i%2 !=0) return Stream.empty();
21)                 else return Stream.of(i,i*i);
22)             }).collect(Collectors.toList());
23)         System.out.println("After using flatMap() method:"+l3);
24)     }
25) }
```

## Output:

D:\durga\_classes>java Test

Before using map() method:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

After using flatMap() method:[0, 2, 4, 6, 8, 10]

After using flatMap() method:[0, 0, 2, 4, 4, 16, 6, 36, 8, 64, 10, 100]

**Q. What is the difference between map() and flatMap() methods?**



## Java 9 Enhancements for Stream API:

In Java 9 as the part of Stream API, the following new methods introduced.

1. `takeWhile()`
2. `dropWhile()`
  
3. `Stream.iterate()`
4. `Stream.ofNullable()`

**Note:** `takeWhile()` and `dropWhile()` methods are default methods and `iterate()` and `ofNullable()` are static methods of Stream interface.

### 1. `takeWhile()`:

It is the default method present in Stream interface.

```
default Stream takeWhile(Predicate p)
```

It returns the stream of elements that matches the given predicate.  
It is similar to `filter()` method.

### Difference between `takeWhile()` and `filter()`:

`filter()` method will process every element present in the stream and consider the element if predicate is true.

But, in the case of `takeWhile()` method, there is no guarantee that it will process every element of the Stream. It will take elements from the Stream as long as predicate returns true. If predicate returns false, at that point onwards remaining elements won't be processed, i.e rest of the Stream is discarded.

**Eg:** Take elements until we will get even numbers. Once we got odd number then stop and ignore rest of the stream.

### Demo Program:

```
1) import java.util.*;  
2) import java.util.stream.*;  
3) public class Test  
4) {  
5)     public static void main(String[] args)  
6)     {  
7)         ArrayList<Integer> l1 = new ArrayList<Integer>();  
8)         l1.add(2);  
9)         l1.add(4);
```



```
10)    l1.add(1);
11)    l1.add(3);
12)    l1.add(6);
13)    l1.add(5);
14)    l1.add(8);
15)    System.out.println("Initial List:" +l1);
16)    List<Integer> l2=l1.stream().filter(i->i%2==0).collect(Collectors.toList());
17)    System.out.println("After Filtering:" +l2);
18)    List<Integer> l3=l1.stream().takeWhile(i->i%2==0).collect(Collectors.toList());
19)    System.out.println("After takeWhile:" +l3);
20) }
21} }
```

### Output:

Initial List:[2, 4, 1, 3, 6, 5, 8]

After Filtering:[2, 4, 6, 8]

After takeWhile:[2, 4]

### Eg 2:

```
Stream.of("A","AA","BBB","CCC","CC","C").takeWhile(s->s.length()<=2).forEach(System.out::println);
```

### Output:

A

AA

## 2. dropWhile()

It is the default method present in Stream interface.

```
default Stream dropWhile(Predicate p)
```

It is the opposite of takeWhile() method.

It drops elements instead of taking them as long as predicate returns true. Once predicate returns false then rest of the Stream will be returned.

### Demo Program:

```
1) import java.util.*;
2) import java.util.stream.*;
3) public class Test
4) {
5)    public static void main(String[] args)
6)    {
7)       ArrayList<Integer> l1 = new ArrayList<Integer>();
8)       l1.add(2);
9)       l1.add(4);
10)      l1.add(1);
```



```
11)    l1.add(3);
12)    l1.add(6);
13)    l1.add(5);
14)    l1.add(8);
15)    System.out.println("Initial List:"+l1);
16)    List<Integer> l2=l1.stream().dropWhile(i->i%2==0).collect(Collectors.toList());
17)    System.out.println("After dropWhile:"+l2);
18) }
19) }
```

### Output:

Initial List:[2, 4, 1, 3, 6, 5, 8]  
After dropWhile:[1, 3, 6, 5, 8]

### Eg 2:

```
Stream.of("A","AA","BBB","CCC","CC","C").dropWhile(s->s.length()<=2).forEach(System.out::println);
```

### Output:

BBB  
CCC  
CC  
C

## 3. Stream.iterate():

It is the static method present in Stream interface.

### Form-1: iterate() method with 2 Arguments

This method introduced in Java 8.

```
public static Stream iterate (T initial,UnaryOperator<T> f)
```

It takes an initial value and a function that provides next value.

Eg: Stream.iterate(1,x->x+1).forEach(System.out::println);

### Output:

1  
2  
3  
...infinite times



## How to limit the number of iterations:

For this we can use limit() method.

Eg: Stream.iterate(1,x->x+1).limit(5).forEach(System.out::println);

### Output:

1  
2  
3  
4  
5

## Form-2: iterate() method with 3 arguments

The problem with 2 argument iterate() method is there may be a chance of infinite loop. To avoid, we should use limit method.

To prevent infinite loops, in Java 9, another version of iterate() method introduced, which is nothing but 3-arg iterate() method.

This method is something like for loop

```
for(int i =0;i<10;i++){}
```

```
public static Stream iterate(T initial,Predicate conditionCheck,UnaryOperator<T> f)
```

This method takes an initial value,  
A terminate Predicate  
A function that provides next value.

Eg: Stream.iterate(1,x->x<5,x->x+1).forEach(System.out::println);

### Output:

1  
2  
3  
4

## 4. ofNullable():

```
public static Stream<T> ofNullable(T t)
```

This method will check whether the provided element is null or not. If it is not null, then this method returns the Stream of that element. If it is null then this method returns empty stream.

This method is helpful to deal with null values in the stream



The main advantage of this method is to we can avoid *NullPointerException* and null checks everywhere.

Usually we can use this method in flatMap() to handle null values.

Eg 1:

```
List l=Stream.ofNullable(100).collect(Collectors.toList());
System.out.println(l);
```

Output:[100]

Eg 2:

```
List l=Stream.ofNullable(null).collect(Collectors.toList());
System.out.println(l);
```

Output:[]

## Demo Program:

```
1) import java.util.*;
2) import java.util.stream.*;
3) import java.util.*;
4) import java.util.stream.*;
5) public class Test
6) {
7)     public static void main(String[] args)
8)     {
9)         List<String> l=new ArrayList<String>();
10)        l.add("A");
11)        l.add("B");
12)        l.add(null);
13)        l.add("C");
14)        l.add("D");
15)        l.add(null);
16)        System.out.println(l);
17)
18)        List<String> l2= l.stream().filter(o->o!=null).collect(Collectors.toList());
19)        System.out.println(l2);
20)
21)        List<String> l3= l.stream()
22)                            .flatMap(o->Stream.ofNullable(o)).collect(Collectors.toList());
23)        System.out.println(l3);
24)    }
25) }
```

Output:

[A, B, null, C, D, null]

[A, B, C, D]

[A, B, C, D]



## Demo Program:

```
1) import java.util.*;
2) import java.util.stream.*;
3) public class Test
4) {
5)     public static void main(String[] args)
6)     {
7)         Map<String, String> m=new HashMap<>();
8)         m.put("A","Apple");
9)         m.put("B","Banana");
10)        m.put("C",null);
11)        m.put("D","Dog");
12)        m.put("E",null);
13)        List<String> l=m.entrySet().stream().map(e->e.getKey()).collect(Collectors.toList());
14)        System.out.println(l);
15)
16)        List<String> l2=m.entrySet().stream()
17)                      .flatMap(e->Stream.ofNullable(e.getValue())).collect(Collectors.toList());
18)        System.out.println(l2);
19)
20)    }
21) }
```

## Output:

[A, B, C, D, E]  
[Apple, Banana, Dog]



# The Java Shell (RPEL)

## JShell Agenda

|                                                        |     |
|--------------------------------------------------------|-----|
| 1) Introduction to the JShell .....                    | 686 |
| 2) Getting Started with JShell .....                   | 687 |
| 3) Getting Help from the JShell .....                  | 691 |
| 4) Understanding JShell Snippets .....                 | 700 |
| 5) Editing and Navigating Code Snippets .....          | 705 |
| 6) Working with JShell Variables .....                 | 707 |
| 7) Working with JShell Methods .....                   | 712 |
| 8) Using An External Editor with JShell .....          | 717 |
| 9) Using classes,interfaces and enum with JShell ..... | 719 |
| 10) Loading and Saving Snippets in JShell .....        | 721 |
| 11) Using Jar Files in the JShell .....                | 725 |
| 12) How to customize JShell Startup .....              | 727 |
| 13) Shortcuts and Auto-Completion of Commands .....    | 731 |



# UNIT 1: Introduction to the JShell

Jshell is also known as interactive console.

JShell is Java's own REPL Tool.

REPL means →Read, Evaluate, Print and Loop

By using this tool we can execute Java code snippets and we can get immediate results.

For beginners it is very good to start programming in fun way.

By using this Jshell we can test and execute Java expressions, statements, methods, classes etc. It is useful for testing small code snippets very quickly, which can be plugged into our main coding based on our requirement.

Prior to Java 9 we cannot execute a single statement, expression, methods without full pledged classes. But in Java 9 with JShell we can execute any small piece of code without having complete class structure.

It is not new thing in Java. It is already there in other languages like Python, Swift, Lisp, Scala, Ruby etc..

Python →IDLE

Apple's Swift Programming Language → PlayGround

### Limitations of JShell:

1. JShell is not meant for Main Coding. We can use just to test small coding snippets, which can be used in our Main Coding.
2. JShell is not replacement of Regular Java IDEs like Eclipse, NetBeans etc
3. It is not that much impressed feature. All other languages like Python, LISP, Scala, Ruby, Swift etc are already having this REPL tools



# UNIT-2: Getting Started with JShell

## Starting and Stopping JShell:

Open the jshell from the command prompt in verbose mode

```
jshell -v
```

```
Command Prompt
D:\durga_classes>jshell -v
| Welcome to JShell -- Version 9
| For an introduction type: /help intro

jshell> /exit
| Goodbye

D:\durga_classes>
```

```
D:\durga_classes>jshell -v
| Welcome to JShell -- Version 9
| For an introduction type: /help intro
```

## How to exit jshell:

```
jshell> /exit
| Goodbye
```

Note: Observe the difference b/w with -v and without -v (verbose mode)

```
D:\durga_classes>jshell -v
| Welcome to JShell -- Version 9
| For an introduction type: /help intro
```

Note: If any information displaying on the jshell starts with '|', it is the information to the programmer from the jshell

```
jshell> 10+20
$1 ==> 30
| created scratch variable $1 : int
```

```
jshell> 20-30*6/2
$2 ==> -70
| created scratch variable $2 : int
```



```
jshell> System.out.println("DURGASOFT")
DURGASOFT
```

Here if we observe the output not starts with | because it is not information from the Jshell.

**Note:** Terminating semicolons are automatically added to the end of complete snippet by JShell if not entered. .

```
jshell> Math.sqrt(4)
$4 ==> 2.0
| created scratch variable $4 : double
```

```
jshell> Math.max(10,20)
$5 ==> 20
| created scratch variable $5 : int
```

```
jshell> Math.random()
$6 ==> 0.6956946870985563
| created scratch variable $6 : double
```

```
jshell> Math.random()
$7 ==> 0.3657412865477785
| created scratch variable $7 : double
```

```
jshell> Math.random()
$8 ==> 0.8828801968574324
| created scratch variable $8 : double
```

**Note:** We are not required to import *Java.lang* package, because by default available.

**Can you check whether the following will work or not?**

```
jshell> ArrayList<String> l = new ArrayList<String>();
l ==> []
| created variable l : ArrayList<String>
```

**Note:** The following packages are bydefault available to the Jshell and we are not required to import. We can check with /imports command

```
jshell> /imports
import Java.io.*
import Java.math.*
import Java.net.*
import Java.nio.file.*
import Java.util.*
import Java.util.concurrent.*
import Java.util.function.*
import Java.util.prefs.*
import Java.util.regex.*
import Java.util.stream.*
```



```
jshell> ArrayList<String> l=new ArrayList<String>();  
l ==> []  
  
jshell> l.add("Sunny");l.add("Bunny");l.add("Chinny");  
$2 ==> true  
$3 ==> true  
$4 ==> true  
  
jshell> l  
l ==> [Sunny, Bunny, Chinny]  
  
jshell> l.isEmpty()  
$6 ==> false  
  
jshell> l.get(2)  
$7 ==> "Chinny"  
  
jshell> l.get(10)  
| Java.lang.IndexOutOfBoundsException thrown: Index 10 out-of-bounds for length 3  
  
jshell> l.size()  
$9 ==> 3  
  
jshell> if(l.isEmpty()) System.out.println("Empty");else System.out.println("Not Empty");  
Not Empty  
  
jshell> for(int i =0;i<10;i=i+2)System.out.println(i)  
0  
2  
4  
6  
8
```

**Note:** Internally jshell having Java compiler which is responsible to check syntax. If any violation we will get Compile time error which is exactly same as normal compile time errors.

```
jshell> System.out.println(x+y)  
| Error:  
| cannot find symbol  
|   symbol: variable x  
| System.out.println(x+y)  
|           ^  
  
| Error:  
| cannot find symbol  
|   symbol: variable y  
| System.out.println(x+y)  
jshell> System.out.println("Durga")  
| Error:  
| package System does not exist  
| System.out.println("Durga")
```



| ^-----^

**Note:** In our program if there is any chance of getting checked exceptions compulsory we required to handle either by try-catch or by throws keyword. Otherwise we will get Compile time error.

**Eg:**

```
1) import java.io.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         PrintWriter pw=new PrintWriter("abc.txt");
7)         pw.println("Hello");
8)     }
9) }
```

D:\durga\_classes>javac Test.java

Test.java:6: error: unreported exception FileNotFoundException; must be caught or declared to be thrown

PrintWriter pw=new PrintWriter("abc.txt");

But in the case of Jshell, jshell itself will takes care of these and we are not required to use try-catch or throws. Thanks to Jshell.

```
jshell> PrintWriter pw=new PrintWriter("abc.txt");pw.println("Hello");pw.flush();
pw ==> Java.io.PrintWriter@e25b2fe
```

## Conclusions:

1. From the jshell we can execute any expression, any Java statement.
2. Most of the packages are not required to import to the Jshell because by default already available to the jshell.
3. Internally jshell use Java compiler to check syntaxes
4. If we are not handling any checked exceptions we won't get any compile time errors, because jshell will takes care.



# UNIT - 3: Getting Help from the JShell

If You Cry For Help....JShell will provide everything.

JShell can provide complete information about available commands with full documentation, and how to use each command and what are various options are available etc..

## Agenda:

- 1) To know list of options allowed with jshell:

Type jshell --help from normal command prompt

- 2) To know the version of jshell:

Type jshell --version from normal command prompt

- 3) To know introduction of jshell:

jshell> /help intro

- 4) For List of commands:

type /help from jshell

- 5) To get information about a particular command:

jshell>/help commandname

- 6) To get just names of all commands without any description:

just type / followed by tab

- 7) To know the list of options available for a command

jshell>/command - tab

jshell> /list -  
-all      -history -start



## 1) To know list of options allowed with jshell

Type `jshell --help` from normal command prompt

`D:\durga_classes>jshell --help`

Usage: `jshell <options> <load files>`

where possible options include:

- `--class-path <path>` Specify where to find user class files
- `--module-path <path>` Specify where to find application modules
- `--add-modules <module>(<module>)*`  
Specify modules to resolve, or all modules on the module path if `<module>` is ALL-MODULE-PATHs
- `--startup <file>` One run replacement for the start-up definitions
- `--no-startup` Do not run the start-up definitions
- `--feedback <mode>` Specify the initial feedback mode. The mode may be predefined (silent, concise, normal, or verbose) or previously user-defined
  - `-q` Quiet feedback. Same as: `--feedback concise`
  - `-s` Really quiet feedback. Same as: `--feedback silent`
  - `-v` Verbose feedback. Same as: `--feedback verbose`
- `-J<flag>` Pass `<flag>` directly to the runtime system.  
Use one `-J` for each runtime flag or flag argument
- `-R<flag>` Pass `<flag>` to the remote runtime system.  
Use one `-R` for each remote flag or flag argument
- `-C<flag>` Pass `<flag>` to the compiler.  
Use one `-C` for each compiler flag or flag argument
- `--version` Print version information and exit
- `--show-version` Print version information and continue
- `--help` Print this synopsis of standard options and exit
- `--help-extra, -X` Print help on non-standard options and exit

## 2) To know the version of jshell

Type `jshell --version` from normal command prompt

`D:\durga_classes>jshell --version`

`jshell 9`



### 3) To know introduction of jshell

```
jshell> /help intro
```

```
| intro
```

The jshell tool allows you to execute Java code, getting immediate results. You can enter a Java definition (variable, method, class, etc), like: int x = 8 or a Java expression, like: x + x or a Java statement or import. These little chunks of Java code are called 'snippets'.

There are also jshell commands that allow you to understand and control what you are doing, like: /list

```
| For a list of commands: /help
```

### 4) For List of commands

```
type /help from jshell
```

```
jshell> /help
```

```
| Type a Java language expression, statement, or declaration.
```

```
| Or type one of the following commands:
```

```
/list [<name or id> | -all | -start]
```

```
    list the source you have typed
```

```
/edit <name or id>
```

```
    edit a source entry referenced by name or id
```

```
/drop <name or id>
```

```
    delete a source entry referenced by name or id
```

```
/save [-all | -history | -start] <file>
```

```
    Save snippet source to a file.
```

```
/open <file>
```

```
    open a file as source input
```

```
/vars [<name or id> | -all | -start]
```

```
    list the declared variables and their values
```

```
/methods [<name or id> | -all | -start]
```

```
    list the declared methods and their signatures
```

```
/types [<name or id> | -all | -start]
```

```
    list the declared types
```

```
/imports
```

```
    list the imported items
```

```
/exit
```

```
    exit jshell
```

```
/env [-class-path <path>] [-module-path <path>] [-add-modules <modules>] ...
```

```
    view or change the evaluation context
```



```
/reset [-class-path <path>] [-module-path <path>] [-add-modules <modules>]...
    reset jshell
/reload [-restore] [-quiet] [-class-path <path>] [-module-path <path>]...
    reset and replay relevant history -- current or previous (-restore)
/history
    history of what you have typed
/help [<command> | <subject>]
    get information about jshell
/set editor|start|feedback|mode|prompt|truncation|format ...
    set jshell configuration information
/? [<command> | <subject>]
    get information about jshell
/!
    re-run last snippet
/<id>
    re-run snippet by id
/-<n>
    re-run n-th previous snippet
```

For more information type '/help' followed by the name of a command or a subject.

For example '/help /list' or '/help intro'.

#### Subjects:

intro

    an introduction to the jshell tool

shortcuts

    a description of keystrokes for snippet and command completion,  
    information access, and automatic code generation

context

    the evaluation context options for /env /reload and /reset

## 5) To get information about a particular command

jshell>/help commandname

jshell> /help list

/list

Show the source of snippets, prefaced with the snippet id.

/list

    List the currently active snippets of code that you typed or read with /open

/list -start

    List the automatically evaluated start-up snippets



```
/list -all
List all snippets including failed, overwritten, dropped, and start-up

/list <name>
List snippets with the specified name (preference for active snippets)

/list <id>
List the snippet with the specified snippet id
```

## To get Information about methods command

```
jshell> /help methods

/methods

List the name, parameter types, and return type of jshell methods.

/methods
List the name, parameter types, and return type of the current active jshell methods

/methods <name>
List jshell methods with the specified name (preference for active methods)

/methods <id>
List the jshell method with the specified snippet id

/methods -start
List the automatically added start-up jshell methods

/methods -all
List all snippets including failed, overwritten, dropped, and start-up
```

## 6) To get just names of all commands without any description

just type / followed by tab

```
jshell> /
!      /?      /drop   /edit   /env    /exit   /help
/history /imports /list   /methods /open   /reload /reset
/save    /set    /types  /vars
```

<press tab again to see synopsis>



**If we press tab again then we will get one line synopsis for every command:**

```
jshell> /  
/!  
re-run last snippet  
  
/-<n>  
re-run n-th previous snippet  
  
/<id>  
re-run snippet by id  
  
/?  
get information about jshell  
  
/drop  
delete a source entry referenced by name or id  
  
/edit  
edit a source entry referenced by name or id  
  
/env  
view or change the evaluation context  
  
/exit  
exit jshell  
  
/help  
get information about jshell  
  
/history  
history of what you have typed  
  
/imports  
list the imported items  
  
/list  
list the source you have typed  
  
/methods  
list the declared methods and their signatures  
  
/open  
open a file as source input  
  
/reload  
reset and replay relevant history -- current or previous (-restore)
```



/reset  
reset jshell

/save  
Save snippet source to a file.

/set  
set jshell configuration information

/types  
list the declared types

/vars  
list the declared variables and their values

<press tab again to see full documentation>

**If we press tab again then we can see full documentation of command one by one:**

jshell> /  
/!  
Reevaluate the most recently entered snippet.

<press tab to see next command>

jshell> /  
/-<n>  
Reevaluate the n-th most recently entered snippet.

<press tab to see next command>

jshell> /  
/<id>  
Reevaluate the snippet specified by the id.

<press tab to see next command>

## 7) To know the list of options available for a command

jshell>/command - tab

jshell> /list -  
-all -history -start

<press tab again to see synopsis>

jshell> /list -



## If we press tab again then we will get synopsis:

```
jshell> /list -  
list the source you have typed
```

```
<press tab again to see full documentation>  
jshell> /list -
```

## If we press tab again then we will get documentation:

```
jshell> /list -  
Show the source of snippets, prefaced with the snippet id.
```

```
/list  
List the currently active snippets of code that you typed or read with /open
```

```
/list -start  
List the automatically evaluated start-up snippets
```

```
/list -all  
List all snippets including failed, overwritten, dropped, and start-up
```

```
/list <name>  
List snippets with the specified name (preference for active snippets)
```

```
/list <id>  
List the snippet with the specified snippet id
```



# Conclusions:

## 1) To know list of options allowed with jshell

Type jshell --help from normal command prompt

## 2) To know the version of jshell

Type jshell --version from normal command prompt

## 3) To know introduction of jshell

jshell> /help intro

## 4) For List of commands

type /help from jshell

## 5) To get information about a particular command

jshell>/help commandname

## 6) To get just names of all commands without any description

just type / followed by tab

## 7) To know the list of options available for a command

jshell>/command - tab

jshell> /list -  
-all    -history -start



## UNIT - 4: Understanding JShell Snippets

### What are Coding Snippets?

Everything what allowed in Java is a snippet. It can be Expression, Declaration, Statement, classe, interface, method, variable, import,.... We can use all these as snippets from jshell.

\*\*\*But package declarations are not allowed from the jshell.

```
jshell> System.out.println("Hello")
Hello
```

```
jshell> int x=10
x ==> 10
| created variable x : int
```

```
jshell> 10+20
$3 ==> 30
| created scratch variable $3 : int
```

```
jshell> $3>x
$4 ==> true
| created scratch variable $4 : boolean
```

```
jshell> String s =10
| Error:
| incompatible types: int cannot be converted to Java.lang.String
| String s=10;
|     ^^
```

```
jshell> String s= "Durga"
s ==> "Durga"
| created variable s : String
```

```
jshell> public void m1()
...> {
...>     System.out.println("hello");
...> }
| created method m1()
```

```
jshell> m1()
hello
```



**Note:** We can use /list command to list out all snippets stored in the jshell memory with snippet id.

```
jshell> /list
```

```
1 : System.out.println("Hello")
2 : int x=10;
3 : 10+20
4 : $3>x
5 : String s= "Durga";
6 : public void m1()
{
    System.out.println("hello");
}
7 : m1()
```

The numbers 1,2,3 are snippet id. In the future we can access the snippet with id directly.

**Note:** There are some snippets which will be executed automatically at the time jshell startup, and these are called start-up snippets. We can also add our own snippets as start-up snippets.

We can list out all start-up snippets with command: /list -start

```
jshell> /list -start
```

```
s1 : import Java.io.*;
s2 : import Java.math.*;
s3 : import Java.net.*;
s4 : import Java.nio.file.*;
s5 : import Java.util.*;
s6 : import Java.util.concurrent.*;
s7 : import Java.util.function.*;
s8 : import Java.util.prefs.*;
s9 : import Java.util.regex.*;
s10 : import Java.util.stream.*;
```

All these are default imports to the jshell.

We can list out all snippets by the command: /list -all

```
jshell> /list -all
```

```
s1 : import Java.io.*;
s2 : import Java.math.*;
s3 : import Java.net.*;
s4 : import Java.nio.file.*;
s5 : import Java.util.*;
s6 : import Java.util.concurrent.*;
s7 : import Java.util.function.*;
```



```
s8 : import Java.util.prefs.*;
s9 : import Java.util.regex.*;
s10 : import Java.util.stream.*;
1 : System.out.println("Hello")
2 : int x=10;
3 : 10+20
4 : $3>x
e1 : String s =10;
5 : String s= "Durga";
6 : public void m1()
{
    System.out.println("hello");
}
7 : m1()
```

We can access snippets by using id directly.

```
jshell> /list 1
```

```
1 : System.out.println("Hello")
```

```
jshell> /list 1 2
```

```
1 : System.out.println("Hello")
2 : int x=10;
```

```
jshell> /list 1 5
```

```
1 : System.out.println("Hello")
5 : String s= "Durga";
```

We can also access snippets directly by using name. The name can be either variable name, class name ,method name etc

```
jshell> /list m1
```

```
6 : public void m1()
{
    System.out.println("hello");
}
```

```
jshell> /list x
```

```
2 : int x=10;
```

```
jshell> /list s
```

```
5 : String s= "Durga";
```



---

We can execute snippet directly by using id with the command: /id

```
jshell> /3  
10+20  
$8 ==> 30  
| created scratch variable $8 : int
```

```
jshell> /7  
m10  
hello
```

We can use drop command to drop a snippet(Making it inactive)  
We can drop snippet by name or id.

```
jshell> /list
```

```
1 : System.out.println("Hello")  
2 : int x=10;  
3 : 10+20  
4 : $3>x  
5 : String s= "Durga";  
6 : public void m1()  
{  
    System.out.println("hello");  
}  
7 : m1()  
8 : 10+20  
9 : m1()
```

```
jshell> /drop $3  
| dropped variable $3
```

```
jshell> /list
```

```
1 : System.out.println("Hello")  
2 : int x=10;  
4 : $3>x  
5 : String s= "Durga";  
6 : public void m1()  
{  
    System.out.println("hello");  
}  
7 : m1()  
8 : 10+20  
9 : m1()
```

```
jshell> /4  
$3>x  
| Error:  
| cannot find symbol
```



```
| symbol: variable $3
| $3>x
| ^^
```

## Conclusions:

1. We can use /list command to list out all snippets stored in the jshell memory with snippet id.  
`jshell>/list`

2. In the future we can access the snippet with id directly without retypinng whole snippet.  
`jshell>/list id`

3. There are some snippets which will be executed automatically at the time jshell star-tup, and these are called start-up snippets. We can list out all start-up snippets with command: /list -start

`jshell> /list -start`  
We can also add our own snippets as start-up snippets.

4. The default start-up snippets are default imports to the jshell.

5. We can list out all snippets by the command: /list -all  
`jshell> /list -all`

6. We can access snippets by using id directly.

`jshell> /list 1`

7. We can access snippets directly by using name. The name can be either variable name, class name ,method name etc

`jshell> /list m1`

8. We can execute snippet directly by using id with the command: /id

`jshell> /3`

9. We can use drop command to drop a snippet(Making it inactive)  
We can drop snippet by name or id.

`jshell> /drop $3`  
Once we dropped a snippet,we cannot use otherwise we will get compile time error.



## UNIT – 5: Editing and Navigating Code Snippets

We can list all our active snippets with /list command and we can list total history of our jshell activities with /history command.

```
jshell> /list
```

```
1 : int x=10;
2 : String s="Durga";
3 : System.out.println("Hello");
4 : class Test{}
```

```
jshell> /history
```

```
int x=10;
String s="Durga";
System.out.println("Hello");
class Test{}
/list
/history
```

1. By using down arrow and up arrow we can navigate through history. While navigating we can use left and right arrows to move character by character with in the snippet.
2. We can Ctrl+A to move to the beginning of the line and Ctrl+E to move to the end of the line.
3. We can use Alt+B to move backward by one word and Alt+F to move forward by one word.
4. We can use Delete key to delete the character at the cursor. We can us Backspace to delete character before the cursor.
5. We can use Ctrl+K to delete the text from the cursor to the end of line.
6. We can use Alt+D to delete the text from the cursor to the end of the word.
7. Ctrl+W to delete the text from cursor to the previous white space.
8. Ctrl+Y to paste most recently deleted text into the line.
9. Ctrl+R to Search backward through history
10. Ctrl+S to search forward through histroy



| KEY         | ACTION                                                        |
|-------------|---------------------------------------------------------------|
| Up arrow    | Moves up one line, backward through history                   |
| Down arrow  | Moves down one line, forward through history                  |
| Left arrow  | Moves backward one character                                  |
| Right arrow | Moves forward one character                                   |
| Ctrl+A      | Moves to the beginning of the line                            |
| Ctrl+E      | Moves to the end of the line                                  |
| Alt+B       | Moves backward one word                                       |
| Alt+F       | Moves forward one word                                        |
| Delete      | Deletes the character at the cursor                           |
| Backspace   | Deletes the character before the cursor                       |
| Ctrl+K      | Deletes the text from the cursor to the end of the line       |
| Alt+D       | Deletes the text from the cursor to the end of the word       |
| Ctrl+W      | Deletes the text from the cursor to the previous white space. |
| Ctrl+Y      | Pastes the most recently deleted text into the line.          |
| Ctrl+R      | Searches backward through history                             |
| Ctrl+S      | Searches forwards through history                             |



# UNIT – 6: Working with JShell Variables

After completing this JShell Variables session, we can answer the following:

1. What are various types of variables possible in jshell?
2. Is it possible to use scratch variable in our code?
3. Is it possible 2 variables with the same name in JShell?
4. If we are trying to declare a variable with the same name which is already available in JShell then what will happen?
5. How to list out all active variables of jshell?
6. How to list out all active& in-active variables of jshell?
7. How to drop variables in the JShell?
8. What is the difference between print() and printf() methods?

In JShell, there are 2 types of variables

1. Explicit variables
2. Implicit variables or Scratch variables

### Explicit variables:

These variables created by programmer explicitly based on our programming requirement.

Eg:

```
jshell> int x =10
x ==> 10
| created variable x : int
```

```
jshell> String s="Durga"
s ==> "Durga"
| created variable s : String
```

The variables x and s provided explicitly by the programmer and hence these are explicit variables.

### Implicit Variables:

Sometimes JShell itself creates variables implicitly to hold temporary values, such type of variables are called Implicit variables.

Eg:

```
jshell> 10+20
$3 ==> 30
| created scratch variable $3 : int
jshell> 10<20
$4 ==> true
```



```
| created scratch variable $4 : boolean
```

The variables \$3 and \$4 are created by JShell and hence these are implicit variables.

Based on requirement we can use these scratch variables also.

```
jshell> $3+40
$5 ==> 70
| created scratch variable $5 : int
```

If we are trying to declare a variable with the same name which is already available then old variable will be replaced with new variable.i.e in JShell, variable overriding is possible.

In JShell at a time only one variable is possible with the same name.i.e 2 variables with the same name is not allowed.

```
jshell> String x="DURGASOFT"
x ==> "DURGASOFT"
| replaced variable x : String
| update overwrote variable x : int
```

In the above case,int variable x is replaced with String variable x.

While declaring variables compulsory the types must be matched,otherwise we will get compile time error.

```
jshell> String s1=true
| Error:
| incompatible types: boolean cannot be converted to Java.lang.String
| String s1=true;
|     ^--^
```

```
jshell> String s1="Hello"
s1 ==> "Hello"
| created variable s1 : String
```

Note: By using /vars command we can list out type,name and value of all variables which are created in JShell.

Instead of /vars we can also use /var,/va,/v

```
jshell> /help vars
|
| /vars
|
| List the type, name, and value of jshell variables.
|
| /vars
|   List the type, name, and value of the current active jshell variables
|
| /vars <name>
|   List jshell variables with the specified name (preference for active var
```



iables)

```
/vars <id>
  List the jshell variable with the specified snippet id

/vars -start
  List the automatically added start-up jshell variables

/vars -all
  List all jshell variables including failed, overwritten, dropped, and st
art-up
```

## To List out All Active variables of JShell:

```
jshell> /vars
| String s = "Durga"
| int $3 = 30
| boolean $4 = true
| String x = "DURGASOFT"
| String s1 = "Hello"
```

## To List out All Variables(both active and not-active):

```
jshell> /vars -all
| int x = (not-active)
| String s = "Durga"
| int $3 = 30
| boolean $4 = true
| String x = "DURGASOFT"
| String s1 = (not-active)
| String s1 = "Hello"
```

We can drop a variable by using /drop command

```
jshell> /vars
| String s = "Durga"
| int $3 = 30
| boolean $4 = true
| String x = "DURGASOFT"
| String s1 = "Hello"
```

```
jshell> /drop $3
| dropped variable $3
jshell> /vars
| String s = "Durga"
| boolean $4 = true
| String x = "DURGASOFT"
| String s1 = "Hello"
```



We can create complex variables also

```
jshell>List<String> heroes=List.of("Ameer","Sharukh","Salman");
heroes ==> [Ameer, Sharukh, Salman]
| created variable heroes : List<String>

jshell>List<String> heroines=List.of("Katrina","Kareena","Deepika");
heroines ==> [Katrina, Kareena, Deepika]
| created variable heroines : List<String>

jshell> List<List<String>> l=List.of(heroes,heroines);
l ==> [[Ameer, Sharukh, Salman], [Katrina, Kareena, Deepika]]
| created variable l : List<List<String>>
```

```
jshell> /vars
| String s = "Durga"
| boolean $4 = true
| String x = "DURGASOFT"
| String s1 = "Hello"
| List<String> heroes = [Ameer, Sharukh, Salman]
| List<String> heroines = [Katrina, Kareena, Deepika]
| List<List<String>> l = [[Ameer, Sharukh, Salman], [Katrina, Kareena, Deepika]]
```

## System.out.println() vs System.out.printf() methods:

```
public class PrintStream
{
    public void print(boolean);
    public void print(char);
    public void println(boolean);
    public void println(char);
    public PrintStream printf(String,Object...);
    ....
}
```

System.out.println() method return type is void.  
But System.out.printf() method return type is PrintStream object. On that PrintStream object we can call printf() method again.

```
jshell> System.out.println("Hello");
Hello

jshell> System.out.printf("Hello:%s\n","Durga")
Hello:Durga
$11 ==> Java.io.PrintStream@10bdf5e5
| created scratch variable $11 : PrintStream

jshell> $11.printf("Hello")
Hello$12 ==> Java.io.PrintStream@10bdf5e5
| created scratch variable $12 : PrintStream
```



```
jshell> /vars
| String s = "Durga"
| boolean $4 = true
| String x = "DURGASOFT"
| String s1 = "Hello"
| List<String> heroes = [Ameer, Sharukh, Salman]
| List<String> heroines = [Katrina, Kareena, Deepika]
| List<List<String>> l = [[Ameer, Sharukh, Salman],
| [Katrina, Kareena, Deepika]]
| PrintStream $11 = Java.io.PrintStream@10bdf5e5
| PrintStream $12 = Java.io.PrintStream@10bdf5e5
```

## FAQs:

1. What are various types of variables possible in jshell?
2. Is it possible to use scratch variable in our code?
3. Is it possible 2 variables with the same name in JShell?
4. If we are trying to declare a variable with the same name which is already available in JShell then what will happen?
5. How to list out all active variables of jshell?
6. How to list out all active& in-active variables of jshell?
7. How to drop variables in the JShell?
8. What is the difference between print() and printf() methods?



# UNIT – 7: Working with JShell Methods

In the JShell we can create our own methods and we can invoke these methods multiple times based on our requirement.

Eg:

```
jshell> public void m1()
...> {
...>   System.out.println("Hello");
...> }
| created method m1()
```

```
jshell> m1()
Hello
```

```
jshell> public void m2()
...> {
...>   System.out.println("New Method");
...> }
| created method m2()
```

```
jshell> m2()
New Method
```

In the JShell there may be a chance of multiple methods with the same name but different argument types, and such type of methods are called overloaded methods. Hence we can declare overloaded methods in the JShell.

```
jshell> public void m1(){}
| created method m1()
```

```
jshell> public void m1(int i){}
| created method m1(int)
```

```
jshell> /methods
| void m1()
| void m1(int)
```

We can list out all methods information by using /methods command.

```
jshell> /help methods
|
| /methods
|
| List the name, parameter types, and return type of jshell methods.
|
| /methods
```



List the name, parameter types, and return type of the current active jshell methods

/methods <name>  
List jshell methods with the specified name (preference for active methods)

/methods <id>  
List the jshell method with the specified snippet id

/methods -start  
List the automatically added start-up jshell methods

/methods -all  
List all snippets including failed, overwritten, dropped, and start-up

jshell> /methods

```
| void m10  
| void m20  
| void m1(int)
```

If we are trying to declare a method with same signature of already existing method in JShell, then old method will be overridden with new method (even though return types are different).  
i.e in JShell at a time only one method with same signature is possible.

jshell> public void m1(int i){}  
| created method m1(int)

jshell> public int m1(int i){return 10;}  
| replaced method m1(int)  
| update overwrote method m1(int)

jshell> /methods  
| int m1(int)

jshell> /methods -all  
| void m1(int)  
| int m1(int)

In the JShell we can create more complex methods also.

**Eg1:** To print the number of occurrences of specified character in the given String

```
1) 5 : public void charCount(String s,char ch)
2) {
3)     int count=0;
4)     for(int i =0; i <s.length(); i++)
5)     {
6)         if(s.charAt(i)==ch)
7)         {
8)             count++;
9)         }
```



```
10)    }
11)    System.out.println("The number of occurrences:"+count);
12) }
```

```
jshell> charCount("Hello DurgaSoft",'o')
The number of occurrences:2
```

```
jshell> charCount("Jajaja",'j')
The number of occurrences:2
```

## Eg 2: To print the sum of given integers

```
1) 8 : public void sum(int... x)
2) {
3)     int total=0;
4)     for(int x1: x)
5)     {
6)         total=total+x1;
7)     }
8)     System.out.println("The Sum:"+total);
9) }
```

```
jshell> sum(10,20)
The Sum:30
```

```
jshell> sum(10,20,30,40)
The Sum:100
```

In JShell, inside method body we can use undeclared variables and methods. But until declaring all dependent variables and methods, we cannot invoke that method.

## Eg1: Usage of undeclared variable inside method body

```
jshell> public void m10
...> {
...>     System.out.println(x);
...> }
| created method m10, however, it cannot be invoked until variable x is declared
```

```
jshell> m10
| attempted to call method m10 which cannot be invoked until variable x is declared
```

```
jshell> int x=10
x ==> 10
| created variable x : int
| update modified method m10
```



```
jshell> m1()
10
```

## Eg 2: Usage of undeclared method inside method body

```
jshell> public void m1()
...> {
...>   m2();
...> }
```

| created method m1(), however, it cannot be invoked until method m2() is declared

```
jshell> m1()
| attempted to call method m1() which cannot be invoked until method m2() is declared
```

```
jshell> public void m2()
...> {
...>   System.out.println("Hello DURGASOFT");
...> }
```

| created method m2()
| update modified method m1()

```
jshell> m1()
Hello DURGASOFT
```

```
jshell> m2()
Hello DURGASOFT
```

We can drop methods by name with /drop command. If multiple methods with the same name then we should drop by snippet id.

```
jshell> public void m1(){}
| created method m1()
```

```
jshell> public void m1(int i){}
| created method m1(int)
```

```
jshell> public void m2(){}
| created method m2()
```

```
jshell> public void m3(){}
| created method m3()
```

```
jshell> /methods
| void m1()
| void m1(int)
| void m2()
| void m3()
```



```
jshell> /drop m3
| dropped method m3()

jshell> /methods
| void m1()
| void m1(int)
| void m2()

jshell> /drop m1
| The argument references more than one import, variable, method, or class.
| Use one of:
| /drop 1  : public void m1(){}
| /drop 2  : public void m1(int i){}

jshell> /methods
| void m1()
| void m1(int)
| void m2()

jshell> /list

1 : public void m1(){}
2 : public void m1(int i){}
3 : public void m2()

jshell> /drop 2
| dropped method m1(int)

jshell> /methods
| void m1()
| void m2()
```

## FAQs:

1. Is it possible to declare methods in the JShell?
2. Is it possible to declare multiple methods with the same name in JShell?
3. Is it possible to declare multiple methods with the same signature in JShell?
4. If we are trying to declare a method with the same name which is already there in the JShell, but with different argument types then what will happen?
5. If we are trying to declare a method with the same signature which is already there in the JShell, then what will happen?
6. Inside a method if we are trying to use a variable or method which is not yet declared then what will happen?
7. How to drop methods in JShell?
8. If multiple methods with the same name then how to drop these methods?



# UNIT – 8: Using An External Editor with JShell

It is very difficult to type lengthy code from JShell. To overcome this problem, JShell provide in-built editor.

We can open inbuilt editor with the command: /edit

```
jshell> /edit
```

diagram(image) of inbuilt-editor

If we are not satisfied with JShell in-built editor, then we can set our own editor to the JShell. For this we have to use /set editor command.

Eg:

```
jshell> /set editor "C:\\WINDOWS\\system32\\notepad.exe"  
jshell> /set editor "C:\\Program Files\\EditPlus\\editplus.exe"
```

## How to Set Notepad as editor to JShell:

```
jshell> /set editor "C:\\WINDOWS\\system32\\notepad.exe"  
| Editor set to: C:\\WINDOWS\\system32\\notepad.exe
```

If we type /edit automatically Notepad will be openend.

```
jshell> /edit
```

But this way of setting editor is temporary and it is applicable only for current session.

If we want to set current editor as permanent, then we have to use the command

```
jshell> /set editor -retain  
| Editor setting retained: C:\\WINDOWS\\system32\\notepad.exe
```

## How to set EditPlus as editor to JShell:

```
jshell> /set editor "C:\\Program Files\\EditPlus\\editplus.exe"  
| Editor set to: C:\\Program Files\\EditPlus\\editplus.exe
```

If we type /edit automatically EditPlus editor will be opened.



## How to set default editor once again:

We have to type the following command from the jshell

```
jshell> /set editor -default  
| Editor set to: -default
```

## To make default editor as permanent:

```
jshell> /set editor -retain  
| Editor setting retained: -default
```

**Note:** It is not recommended to set IntelliJ, Eclipse, NetBeans as JShell editors, because it increases startup time and shutdown time of jshell.

## FAQs:

1. How to open default editor of JShell?
2. How to configure our own editor to the JShell?
3. How to configure Notepad as editor to the JShell?
4. How to make our customized editor as permanent editor in the JShell?



## UNIT – 9: Using classes, interfaces and enum with JShell

In the JShell we can declare classes,interfaces,enums also.

We can use /types command to list out our created types like classes,interfaces and enums.

```
jshell> class Student{}  
| created class Student  
  
jshell> interface Interf{}  
| created interface Interf  
  
jshell> enum Colors{}  
| created enum Colors  
  
jshell> /types  
| class Student  
| interface Interf  
| enum Colors
```

But recommended to use editor to type lengthy classes,interfaces and enums.

```
1) 1 : public class Student  
2) {  
3)   private String name;  
4)   private int rollno;  
5)   Student(String name,int rollno)  
6) {  
7)     this.name=name;  
8)     this.rollno=rollno;  
9)   }  
10)  public String getName()  
11) {  
12)   return name;  
13) }  
14)  public int getRollno()  
15) {  
16)   return rollno;  
17) }  
18) }
```

```
jshell> /edit  
| created class Student
```

```
jshell> /types  
| class Student
```

```
jshell> Student s=new Student("Durga",101);  
s ==> Student@754ba872  
| created variable s : Student
```



```
jshell> s.getName()
$3 ==> "Durga"
| created scratch variable $3 : String
```

```
jshell> s.getRollno()
$4 ==> 101
| created scratch variable $4 : int
```

```
1) public interface Interf
2) {
3)     public static void m1()
4)     {
5)         System.out.println("interface static method");
6)     }
7) }
8) enum Beer
9) {
10) KF("Sour"),KO("Bitter"),RC("Salty");
11) String taste;
12) Beer(String taste)
13) {
14)     this.taste=taste;
15) }
16) public String getTaste()
17) {
18)     return taste;
19) }
20) }
```

```
jshell> /edit
| created interface Interf
| created enum Beer
```

```
jshell> Interf.m1()
interface static method
```

```
jshell> Beer.KF.getTaste()
$8 ==> "Sour"
| created scratch variable $8 : String
```



# UNIT – 10: Loading and Saving Snippets in JShell

We can load and save snippets from the file.

Assume all our required snippets are available in mysnippets.jsh. This file can be with any extension like .txt, But recommended to use .jsh.

### mysnippets.jsh:

```
String s="Durga";
public void m1()
{
    System.out.println("method defined in the file");
}
int x=10;
```

We can load all snippets of this file from the JShell with /open command as follows.

```
jshell> /list
jshell> /open mysnippets.jsh
jshell> /list

1 : String s="Durga";
2 : public void m1()
{
    System.out.println("method defined in the file");
}
3 : int x=10;
```

Once we loaded snippets, we can use these loaded snippets based on our requirement.

```
jshell> m1()
method defined in the file

jshell> s
s ==> "Durga"
| value of s : String

jshell> x
x ==> 10
| value of x : int
```



# Saving JShell snippets to the file:

We can save JShell snippets to the file with /save command.

```
jshell> /help save
| /save <file>
|   Save the source of current active snippets to the file.

| /save -all <file>
|   Save the source of all snippets to the file.
|   Includes source including overwritten, failed, and start-up code.

| /save -history <file>
|   Save the sequential history of all commands and snippets entered since jshell was launched.

| /save -start <file>
|   Save the current start-up definitions to the file.
```

**Note:** If the specified file is not available then this save command itself will create that file.

```
jshell> /save active.jsh
jshell> /save -all all.jsh
jshell> /save -start start.jsh
jshell> /save -history history.jsh
jshell> /ex
| Goodbye
```

```
D:\>type active.jsh
int x=10;
x
System.out.println("Hello");
public void m1()
```

```
D:\>type start.jsh
import Java.io.*;
import Java.math.*;
import Java.net.*;
import Java.nio.file.*;
import Java.util.*;
import Java.util.concurrent.*;
import Java.util.function.*;
import Java.util.prefs.*;
import Java.util.regex.*;
```



```
import Java.util.stream.*;
```

**Note:** By default, all files will be created in current working directory. If we want in some other location then we have to use absolute path(Full Path).

```
jshell> /save D:\\durga_classes\\active.jsh
```

## How to Reload Previous state (session) of JShell:

We can reload previous session with /reload command so that all snippets of previous session will be available in the current session.

```
jshell> /reload -restore
```

**Eg:**

```
jshell> int x=10
x ==> 10
| created variable x : int
```

```
jshell> 10+20
$2 ==> 30
| created scratch variable $2 : int
```

```
jshell> System.out.println("Hello");
Hello
```

```
jshell> /list
```

```
1 : int x=10;
2 : 10+20
3 : System.out.println("Hello");
```

```
jshell> /exit
| Goodbye
```

```
D:\\>jshell -v
| Welcome to JShell -- Version 9
| For an introduction type: /help intro
```

```
jshell> /reload -restore
| Restarting and restoring from previous state.
-: int x=10;
-: 10+20
-: System.out.println("Hello");
Hello
```



```
jshell> /list
1 : int x=10;
2 : 10+20
3 : System.out.println("Hello");
```

## How to reset JShell State:

We can reset JShell state by using /reset command.

```
jshell> /help reset
|
| /reset
|
| Reset the jshell tool code and execution state:
|   * All entered code is lost.
|   * Start-up code is re-executed.
|   * The execution state is restarted.
|   Tool settings are maintained, as set with: /set ...
|
| Save any work before using this command.
```

Eg:

```
jshell> /list
1 : int x=10;
2 : 10+20
3 : System.out.println("Hello");
```

```
jshell> /reset
| Resetting state.
```

```
jshell> /list
```



# UNIT – 11: Using Jar Files in the JShell

It is very easy to use external jar files in the jshell. We can add Jar files to the JShell in two ways.

1. From the Command Prompt
2. From the JShell Itself

## 1. Adding Jar File to the JShell from Command Prompt:

We have to open jshell with --class-path option.

```
D:\>jshell -v --class-path C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6.jar
```

### Demo Program to get all employees information from oracle database:

#### mysnippets.jsh:

```
1) import Java.sql.*;
2) public void getEmplInfo() throws Exception
3) {
4)     Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521:XE",
5)           "scott", "tiger");
6)     Statement st=con.createStatement();
7)     ResultSet rs=st.executeQuery("select * from employees");
8)     while(rs.next())
9)     {
10)         System.out.println(rs.getInt(1)+".." +rs.getString(2)+".." +rs.getDouble(3)+".." +rs.getString(4));
11)     }
12)     con.close();
```

#### DaTabase info:

```
1) create Table employees(eno number,ename varchar2(10),esal number(10,2),eaddr varchar
2(10));
2) insert into employees values(100,'Sunny',1000,'Mumbai');
3) insert into employees values(200,'Bunny',2000,'Hyd');
4) insert into employees values(300,'Chinny',3000,'Hyd');
5) insert into employees values(400,'Vinny',4000,'Delhi');
```

```
D:\>jshell -v --class-path C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6.jar
```

```
jshell> /open mysnippets.jsh
jshell> getEmplInfo()
100..Sunny..1000.0..Mumbai
```



200..Bunny..2000.0..Hyd  
300..Chinny..3000.0..Hyd  
400..Vinny..4000.0..Delhi

## 2.Adding Jar File to the JShell from JShell itself:

We can add External Jars to the jshell from the Jshell itself with /env command.

```
jshell> /env --class-path C:\oraclexe\app\oracle\product\11.2.0\server\jdbc\lib\ojdbc6.jar  
| Setting new options and restoring state.
```

```
jshell> /open mysnippets.jsh
```

```
jshell> getEmplInfo()  
100..Sunny..1000.0..Mumbai  
200..Bunny..2000.0..Hyd  
300..Chinny..3000.0..Hyd  
400..Vinny..4000.0..Delhi
```

**Note:** Internally JShell will use environment variable CLASSPATH if we are not setting CLASSPATH explicitly.



## UNIT – 12: How to customize JShell Startup

By default the following snippets will be executed at the time of JShell Startup.

```
jshell> /list -start
```

```
s1 : import Java.io.*;
s2 : import Java.math.*;
s3 : import Java.net.*;
s4 : import Java.nio.file.*;
s5 : import Java.util.*;
s6 : import Java.util.concurrent.*;
s7 : import Java.util.function.*;
s8 : import Java.util.prefs.*;
s9 : import Java.util.regex.*;
s10 : import Java.util.stream.*;
```

We can customize these start-up snippets based on our requirement.  
Assume our required start-up snippets are available in myStartup.jsh.

### mystartup.jsh:

```
int x =10;
String s="DURGA";
System.out.println("Hello Durga Welcome to JShell");
```

To provide these snippets as startup snippets we have to open JShell as follows

```
D:\>jshell -v --startup mystartup.jsh
Hello Durga Welcome to JShell
| Welcome to JShell -- Version 9
| For an introduction type: /help intro
```

```
jshell> /list -start
```

```
s1 : int x =10;
s2 : String s="DURGA";
s3 : System.out.println("Hello Durga Welcome to JShell");
```

Note: if we want DEFAULT import start-up snippets also then we have to open JShell as follows.

```
D:\>jshell -v --startup DEFAULT mystartup.jsh
Hello Durga Welcome to JShell
| Welcome to JShell -- Version 9
| For an introduction type: /help intro
```



```
jshell> /list
1 : int x =10;
2 : String s="DURGA";
3 : System.out.println("Hello Durga Welcome to JShell");
```

```
jshell> /list -start
```

```
s1 : import Java.io.*;
s2 : import Java.math.*;
s3 : import Java.net.*;
s4 : import Java.nio.file.*;
s5 : import Java.util.*;
s6 : import Java.util.concurrent.*;
s7 : import Java.util.function.*;
s8 : import Java.util.prefs.*;
s9 : import Java.util.regex.*;
s10 : import Java.util.stream.*;
```

**Note:** To import all JAVASE packages (almost around 173 packages) at the time of startup we have to open JShell as follows.

```
D:\>jshell -v --startup JAVASE
```

```
jshell> /list
```

```
jshell> /list -start
```

```
s1 : import Java.applet.*;
s2 : import Java.awt.*;
s3 : import Java.awt.color.*;
..
s172 : import org.xml.sax.ext.*;
s173 : import org.xml.sax.helpers.*;
```

**Note:** In addition to JAVASE, to provide our own snippets we have to open JShell as follows

```
D:\>jshell -v --startup JAVASE mystartup.jsh
Hello Durga Welcome to JShell
| Welcome to JShell -- Version 9
| For an introduction type: /help intro
```

```
jshell> /list
```

```
1 : int x =10;
2 : String s="DURGA";
3 : System.out.println("Hello Durga Welcome to JShell");
```

```
jshell> /list -start
```



```
s1 : import Java.applet.*;
s2 : import Java.awt.*;
s3 : import Java.awt.color.*;
..
s172 : import org.xml.sax.ext.*;
s173 : import org.xml.sax.helpers.*;
```

## Q. What is the difference between the following?

1. jshell -v
2. jshell -v --startup mystartup.jsh
3. jshell -v --startup DEFAULT mystartup.jsh
4. jshell -v --startup JAVASE
5. jshell -v --startup JAVASE mystartup.jsh

## Need of PRINTING Option at the startup:

Usually we can use `System.out.print()` or `System.out.println()` methods to print some statements to the console. If we use PRINTING Option then several overloaded `print()` and `println()` methods will be provided at the time of startup and these internally call `System.out.print()` and `System.out.println()` methods.

Hence to print statements to the console just we can use `print()` or `println()` methods directly instead of using `System.out.print()` or `System.out.println()` methods.

D:\>jshell -v --startup PRINTING

jshell> /list -start

```
s1 : void print(boolean b) { System.out.print(b); }
s2 : void print(char c) { System.out.print(c); }
s3 : void print(int i) { System.out.print(i); }
s4 : void print(long l) { System.out.print(l); }
s5 : void print(float f) { System.out.print(f); }
s6 : void print(double d) { System.out.print(d); }
s7 : void print(char s[]) { System.out.print(s); }
s8 : void print(String s) { System.out.print(s); }
s9 : void print(Object obj) { System.out.print(obj); }
s10 : void println() { System.out.println(); }
s11 : void println(boolean b) { System.out.println(b); }
s12 : void println(char c) { System.out.println(c); }
s13 : void println(int i) { System.out.println(i); }
s14 : void println(long l) { System.out.println(l); }
s15 : void println(float f) { System.out.println(f); }
s16 : void println(double d) { System.out.println(d); }
s17 : void println(char s[]) { System.out.println(s); }
s18 : void println(String s) { System.out.println(s); }
s19 : void println(Object obj) { System.out.println(obj); }
```



```
s20 : void printf(Java.util.Locale l, String format, Object... args) { System.out.printf(l, format, args);  
}  
s21 : void printf(String format, Object... args) { System.out.printf(format, args); }
```

Now onwards,to print some statements to the console directly we can use print() and println() methods.

```
jshell> print("Hello");  
Hello  
jshell> print(10.5)  
10.5
```

**Note:**

1. Total 21 overloaded print(),println() and printf() methods provided because of PRINTING shortcut.
2. Whenever we are using PRINTING shortcut,then DEFAULT imports won't come. Hence,to get DEFAULT imports and PRINTING shortcut simultaneously,we have to open JShell as follows.

```
D:\>jshell -v --startup DEFAULT PRINTING
```

**Note:**

Various allowed options with --startup are :

1. DEFAULT
2. JAVASE
3. PRINTING



## UNIT – 13: Shortcuts and Auto-Completion of Commands

### Shortcut for Creating Variables:

Just type the value on the JShell and then "Shift+Tab followed by v" then complete variable declaration code will be generated we have to provide only name of the variable.

```
jshell> "Durga" // just press "Shift+Tab followed by v"  
jshell> String s= "Durga"  
We have to provide only name s
```

```
jshell> 10.5 // just press "Shift+Tab followed by v"  
jshell> double d = 10.5
```

### Shortcut for auto-import:

just type class or interface name on the JShell and press "Shift+Tab followed by i". Then we will get options for import.

```
jshell> Connection // press "Shift+Tab followed by i"  
0: Do nothing  
1: import: com.sun.jdi.connect.spi.Connection  
2: import: Java.sql.Connection  
Choice: //enter 2  
Imported: Java.sql.Connection
```

```
jshell> /imports  
| import Java.io.*  
| import Java.math.*  
| import Java.net.*  
| import Java.nio.file.*  
| import Java.util.*  
| import Java.util.concurrent.*  
| import Java.util.function.*  
| import Java.util.prefs.*  
| import Java.util.regex.*  
| import Java.util.stream.*  
| import Java.sql.Connection
```



## Auto Completion commands :

### 1. To get all static members of the class:

```
jshell>classname.<Tab>
```

Eq:

```
jshell> String.<Tab>
CASE_INSENSITIVE_ORDER  class      copyValueOf(
format(      join(      valueOf(
```

### 2. To get all instance members of class:

```
jshell>objectreference.<Tab>
```

```
jshell> String s="Durga";
jshell> s.<Tab>
charAt(      chars()      codePointAt(      codePointBefore(
codePointCount(  codePoints()      compareTo(      compareToIgnoreCase(
concat(      contains(      contentEquals(      endsWith(
equals(      equalsIgnoreCase(  getBytes(      getChars(
getClass()    hashCode()      indexOf(      intern()
isEmpty()     lastIndexOf(      length()      matches(
notify()      notifyAll()      offsetByCodePoints(  regionMatches(
replace(      replaceAll(      replaceFirst(      split(
startsWith(   subSequence(      substring(      toCharArray(
toLowerCase(  toString()      toUpperCase(      trim()
```

### 3. To get signature and documentation of a method:

```
jshell> classname.methodname(<Tab>
jshell> objectreference.methodname(<Tab>
```

```
jshell> s.sub<Tab>
subSequence(  substring(
```

```
jshell> s.substring(
substring(
```

```
jshell> s.substring(<Tab>
Signatures:
String String.substring(int beginIndex)
String String.substring(int beginIndex, int endIndex)
```

```
<press Tab again to see documentation>
jshell> s.substring(<Tab>
String String.substring(int beginIndex)
```



---

Returns a string that is a substring of this string. The substring begins with the character at the specified index and extends to the end of this string.

**Examples:**

"unhappy".substring(2) returns "happy"  
"Harbison".substring(3) returns "bison"  
"emptiness".substring(9) returns "" (an empty string)

**Parameters:**

`beginIndex` - the beginning index, inclusive.

**Returns:**

the specified substring.

**Note:** Even this <Tab> short cut applicable for our own classes and methods also.

```
jshell> public void m1(int...x){}
| created method m1(int...)
```

```
jshell> m1(<Tab>
m1(
```

```
jshell> m1(<Tab>
Signatures:
void m1(int... x)
```



# The Java Platform Module System (JPMS)

## Introduction:

Modularity concept introduced in Java 9 as the part of Jigsaw project. It is the main important concept in java 9.

The development of modularity concept started in 2005.

The First JEP(JDK Enhancement Proposal) for Modularity released in 2005. Java people tried to release Modularity concept in Java 7(2011) & Java 8(2014). But they failed. Finally after several postponements this concept introduced in Java 9.

Until Java 1.8 version we can develop applications by writing several classes, interfaces and enums. We can place these components inside packages and we can convert these packages into jar files. By placing these jar files in the classpath, we can run our applications. An enterprise application can contain 1000s of jar files also.

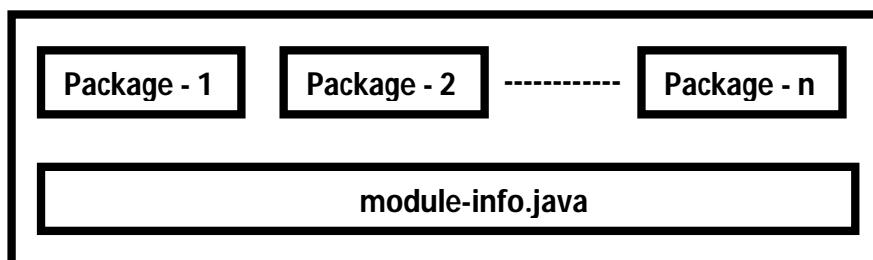
Hence jar file is nothing but a group of packages and each package contains several .class files.

But in Java 9, a new construct got introduced which is nothing but 'Module'. From java 9 version onwards we can develop applications by using module concept.

Module is nothing but a group of packages similar to jar file. But the specialty of module when compared with jar file is, module can contain configuration information also.

Hence module is more powerful than jar file. The configuration information of module should be specified in a special file named with module-info.java

Every module should compulsorily contain module-info.java, otherwise JVM won't consider that as a module of Java 9 platform.



In Java 9, JDK itself modularized. All classes of Java 9 are grouped into several modules (around 98) like

java.base  
java.logging  
java.sql  
java.desktop(AWT/Swing)  
java.rmi etc

java.base module acts as base for all java 9 modules.



We can find module of a class by using `getModule()` method.

Eg: `System.out.println(String.class.getModule()); //module java.base`

## What is the need of JPMS?

Application development by using jar file concept has several serious problems.

### Problem-1: Unexpected `NoClassDefFoundError` in middle of program execution

There is no way to specify jar file dependencies until java 1.8V. At runtime, if any dependent jar file is missing then in the middle of execution of our program, we will get `NoClassDefFoundError`, which is not at all recommended.

### Demo Program to demonstrate `NoClassDefFoundError`:

```
durgajava9  
| -A1.java  
| -A2.java  
| -A3.java  
| -Test.java
```

#### A1.java:

```
1) package pack1;  
2) public class A1  
3) {  
4)     public void m1()  
5)     {  
6)         System.out.println("pack1.A");  
7)     }  
8) }
```

#### A2.java

```
1) package pack2;  
2) import pack1.A1;  
3) public class A2  
4) {  
5)     public void m2()  
6)     {  
7)         System.out.println("pack2.A2 method");  
8)         A1 a = new A1();  
9)         a.m1();  
10)    }  
11) }
```



## A3.java:

```
1) import pack2.A2;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         System.out.println("Test class main");
7)         A2 a= new A2();
8)         a.m2();
9)     }
10) }
```

```
D:\durgajava9>javac -d . A1.java
D:\durgajava9>javac -d . A2.java
D:\durgajava9>javac -d . A3.java
D:\durgajava9>javac Test.java
```

```
durgajava9
|-Test.class
|-pack1
  |-A1.class
|-pack2
  |-A2.class
```

At runtime, by mistake if pack1 is not available then after executing some part of the code in the middle, we will get *NoClassDefFoundError*.

```
D:\durgajava9>java Test
Test class main
pack2.A2 method
Exception in thread "main" java.lang.NoClassDefFoundError: pack1/A1
```

But in Java9, there is a way to specify all dependent modules information in `module-info.java`. If any module is missing then at the beginning only, JVM will identify and won't start its execution. Hence there is no chance of raising *NoClassDefFoundError* in the middle of execution.

## Problem 2: Version Conflicts or Shadowing Problems

If JVM required any .class file, then it always searches in the classpath from left to right until required match found.

**classpath = jar1;jar2;jar3;jar4**

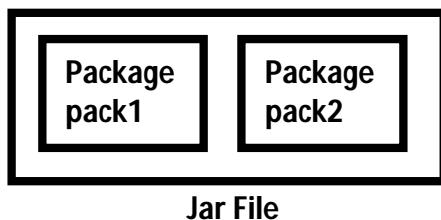
If jar4 requires Test.class file of jar3. But Different versions of Test.class is available in jar1, jar2 and jar3. In this case jar1 Test.class file will be considered, because JVM will always search from Left to Right in the classpath. It will create version conflicts and causes abnormal behavior of program.



But in java9 module system, there is a way to specify dependent modules information for every module separately. JVM will always consider only required module and there is no order importance. Hence version conflicts won't be raised in Java 9.

## Problem 3: Security problem

There is no mechanism to hide packages of jar file.



Assume pack1 can be used by other jar files, but pack2 is just for internal purpose only. Until Java 8 there is no way to specify this information. Everything in jar file is public and available to everyone. Hence there may be a chance of Security problems.

public is too much public in jar files.

But in Java 9 Module system, we can export particular package of a module. Only this exported package can be used by other modules. The remaining packages of that module are not visible to outside. Hence Strong encapsulation is available in Java 9 and there is no chance of security problems.

Even though class is public, if module won't export the corresponding package, then it cannot be accessed by other modules. Hence public is not really that much public in Java 9 Module System.

Module can offer Strong Encapsulation than Jar File.

## Problem 4: JDK/JRE having Monolithic Structure and Very Large Size

The number of classes in Java is increasing very rapidly from version to version.

JDK 1.0V having 250+ classes

JDK 1.1V having 500+ classes

...

JDK 1.8V having 4000+ classes

And all these classes are available in rt.jar.

Hence the size of rt.jar is increasing from version to version.

The size of rt.jar in Java 1.8Version is around 60 MB.

To run small program also, total rt.jar should be loaded, which makes our application heavy weight and not suitable for IOT applications and micro services which are targeted for portable devices.

It will create memory and performance problems also.



(This is something like inviting a Big Elephant in our Small House: Installing a Heavy Weight Java application in a small portable device).

But in java 9, rt.jar removed. Instead of rt.jar all classes are maintained in the form of modules. Hence from Java 9 onwards JDK itself modularized. Whenever we are executing a program only required modules will be loaded instead of loading all modules, which makes our application light weighted.

Now we can use java applications for small devices also. From Java 9 version onwards, by using JLINK , we can create our own very small custom JREs with only required modules.

## Q. Explain differences between jar file and Java 9 module

| Jar                                                                                                                                                                                                                                                                                | Module                                                                                                                                                                                                                                                 |
|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 1) Jar is a group of packages and each package contains several classes                                                                                                                                                                                                            | 1) Module is also a group of packages and each package contains several classes. Module can also contain one special file module-info.java to hold module specific dependencies and configuration information.                                         |
| 2) In jar file, there is no way to specify dependent jar files information                                                                                                                                                                                                         | 2) For every module we have to maintain a special file module-info.java to specify module dependencies                                                                                                                                                 |
| 3) There is no way to check all jar file dependencies at the beginning only. Hence in the middle of the program execution there may be a chance of NoClassDefFoundError.                                                                                                           | 3) JVM will check all module dependencies at the beginning only with the help of module-info.java. If any dependent module is missing then JVM won't start its execution. Hence there is no chance of NoClassDefFoundError in the middle of execution. |
| 4) In the classpath the order of jar files important and JVM will always consider from left to right for the required .class files. If multiple jars contain the same .class file then there may be a chance of Version conflicts and results abnormal behavior of our application | 4) In the module-path order is not important. JVM will always check from the dependent module only for the required .class files. Hence there is no chance of version conflicts and abnormal behavior of the application.                              |
| 5) In jar file there is no mechanism to control access to the packages. Everything present in the jar file is public to everyone. Any person is allowed to access any component from the jar file. Hence there may be a chance of security problems                                | 5) In module there is a mechanism to control access to the packages. Only exported packages are visible to other modules. Hence there is no chance of security problems                                                                                |
| 6) Jars follows monolithic structure and applications will become heavy weight and not suitable for small devices.                                                                                                                                                                 | 6) Modules follow distributed structure and applications will become light weighted and suitable for small devices.                                                                                                                                    |
| 7) Jar files approach cannot be used for IOT devices and micro services.                                                                                                                                                                                                           | 7) Modules based approach can be used for IOT devices and micro services                                                                                                                                                                               |



## Q. What is Jar Hell or Classpath Hell?

The problems with use of jar files are:

1. *NoClassDefFoundError* in the middle of program execution
2. Version Conflicts and Abnormal behavior of program
3. Lack of Security
4. Bigger Size

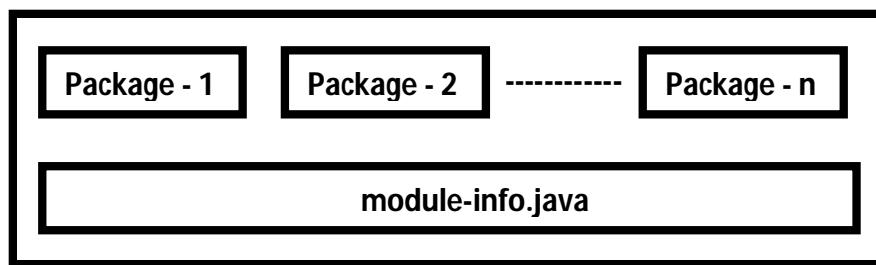
This set of Problems is called Jar Hell OR Classpath Hell. To overcome this, we should go for JPMS.

## Q. What are various Goals/Benefits of JPMS?

1. Reliable Configuration
  2. Strong Encapsulation & Security
  3. Scalable Java Platform
  4. Performance and Memory Improvements
- etc

## What is a Module:

Module is nothing but collection of packages. Each module should compulsory contains a special configuration file: `module-info.java`.

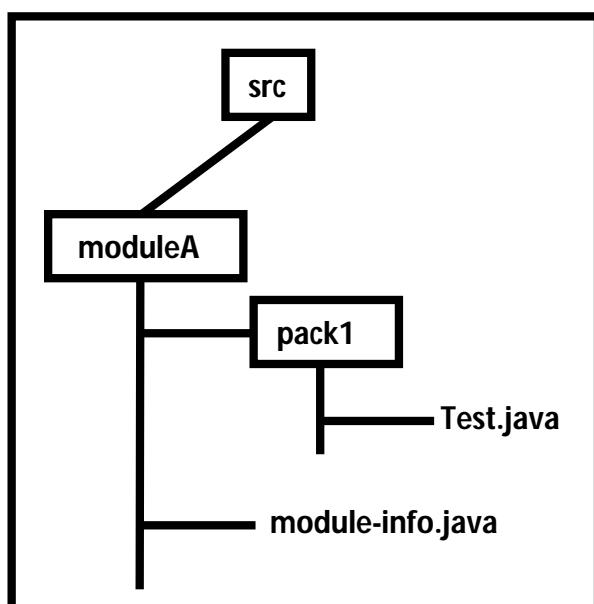


We can define module dependencies inside `module-info.java` file.

```
module moduleName
{
    Here we have to define module dependencies which represents
    1. What other modules required by this module?
    2. What packages exported by this module for other modules?
    etc
}
```



## Steps to Develop First Module Based Application:



### Step-1: Create a package with our required classes

```
1) package pack1;
2) public class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         System.out.println("First Module in JPMS");
7)     }
8) }
```

### Step-2: Writing module-info.java

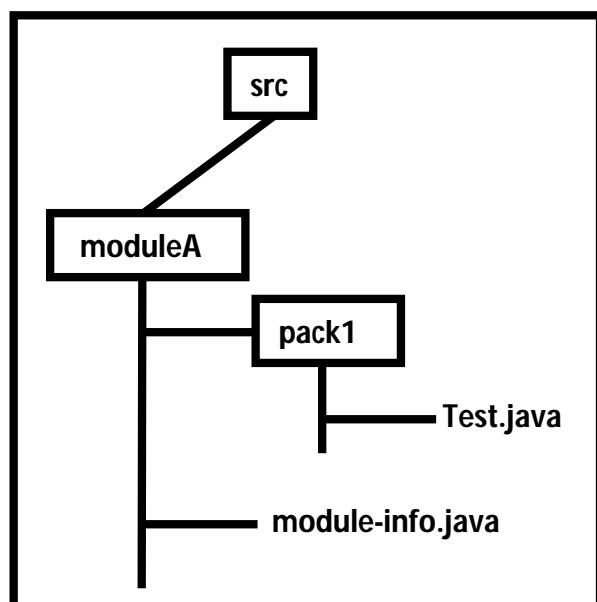
For every module we should to write a special file named with **module-info.java**.  
In this file we have to define dependencies of module.

```
module moduleA
{
}
```



## Step-3: Arrange all files in the required package structure

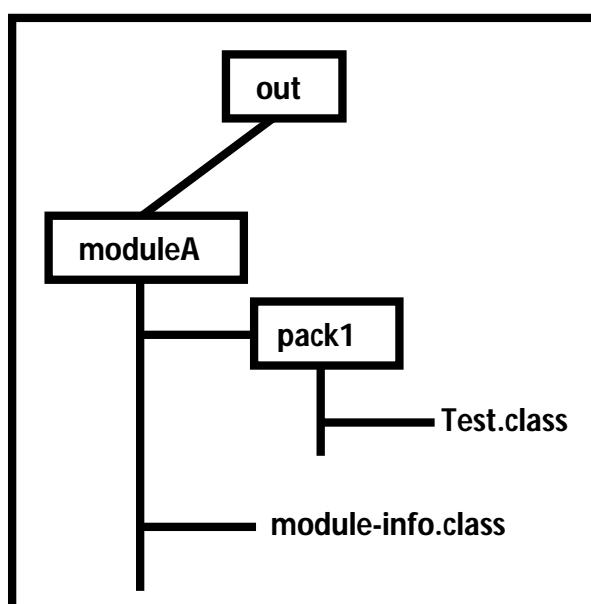
Arrange all the files according to required folder structure



## Step-4: Compile module with --module-source-path option

```
javac --module-source-path src -d out -m moduleA
```

The generated class file structure is:





## Step-5: Run the class with --module-path option

```
java --module-path out -m moduleA/pack1.Test
```

Output: First Module in JPMS

### Case-1:

If module-info.java is not available then the code won't compile and we will get error. Hence module-info.java is mandatory for every module.

```
javac --module-source-path src -d out -m moduleA  
error: module moduleA not found in module source path
```

### Case-2:

Every class inside module should be part of some package, otherwise we will get compile time error saying : unnamed package is not allowed in named modules  
In the above application inside Test.java if we comment package statement

```
//package pack1;  
  
1) public class Test  
2) {  
3)     public static void main(String[] args)  
4)     {  
5)         System.out.println("First Module in JPMS");  
6)     }  
7) }
```

error: unnamed package is not allowed in named modules

### Case-3:

The module name should not ends with digit (like module1, module2 etc), otherwise we will get warning at compile time.

```
javac --module-source-path src -d out -m module1  
warning: [module] module name component module1 should avoid terminal digits
```



## Various Possible Ways to Compile a Module:

```
javac --module-source-path src -d out -m moduleA  
javac --module-source-path src -d out --module moduleA  
javac --module-source-path src -d out  
    src/moduleA/module-info.java src/moduleA/pack1/Test.java  
javac --module-source-path src -d out  
    C:/Users/Durga/Desktop/src/moduleA/module-info.java  
C:/Users/Durga/Desktop/src/moduleA/pack1/Test.java
```

## Various Possible Ways to Run a Module:

```
java --module-path out --add-modules moduleA pack1.Test  
java --module-path out -m moduleA/pack1.Test  
java --module-path out --module moduleA/pack1.Test
```

## Inter Module Dependencies:

Within the application we can create any number of modules and one module can use other modules.

We can define module dependencies inside module-info.java file.

```
module moduleName  
{  
    Here we have to define module dependencies which represents  
    1. What other modules required by this module?  
    2. What packages exported by this module for other modules?  
    etc  
}
```

Mainly we can use the following 2 types of directives

### 1. requires directive:

It can be used to specify the modules which are required by current module.

Eg:

```
1) module moduleA  
2) {  
3)     requires moduleB;  
4) }
```

It indicates that moduleA requires members of moduleB.



## Note:

1. We cannot use same requires directive for multiple modules. For every module we have to use separate requires directive.

requires moduleA,moduleB; ➔ invalid

2. We can use requires directive only for modules but not for packages and classes.

## 2. exports directive:

It can be used to specify what packages exported by current module to the other modules.

### Eg:

```
1) module moduleA  
2) {  
3)   exports pack1;  
4) }
```

It indicates that moduleA exporting pack1 package so that this package can be used by other modules.

Note: We cannot use same exports directive for exporting multiple packages. For every package a separate exports directive must be required.

exports pack1,pack2; ➔ invalid

Note: Be careful about syntax requires directive always expecting module name where as exports directive expecting package name.

```
1) module modulename  
2) {  
3)   requires modulename;  
4)   exports packagename;  
5) }
```

## Note:

By default all packages present in a module are private to that module. If module exports any package only that particular package is accessible by other modules. Non exporting packages cannot be accessed by other modules.

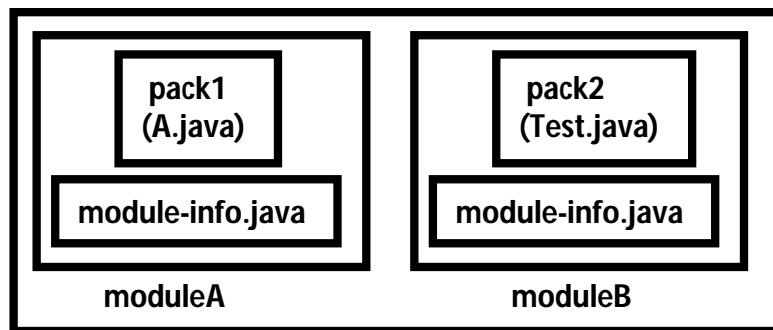
Eg: Assume moduleA contains 2 packages pack1 and pack2. If moduleA exports only pack1 then other modules can use only pack1. pack2 is just for its internal purpose and cannot be accessed by other modules.

```
1) module moduleA  
2) {  
3)   exports pack1;  
4) }
```



## Demo program for inter module dependencies:

Rectangle diagram which represents total application



## moduleA components:

### A.java:

```
1) package pack1;
2) public class A
3) {
4)     public void m1()
5)     {
6)         System.out.println("Method of moduleA");
7)     }
8) }
```

### module-info.java:

```
1) module moduleA
2) {
3)     exports pack1;
4) }
```



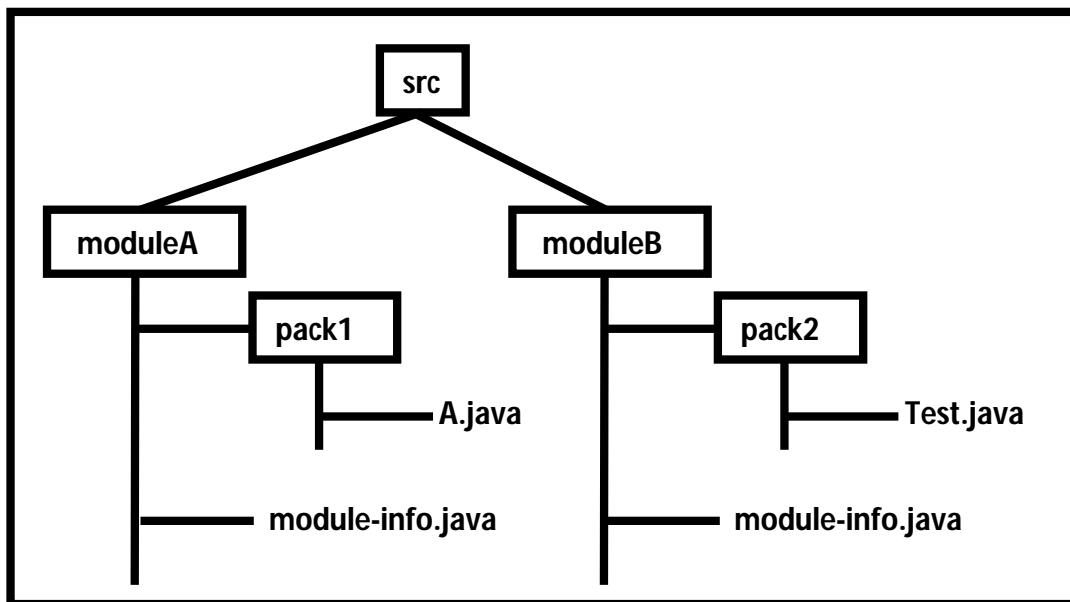
## moduleB components:

### Test.java:

```
1) package pack2;
2) import pack1.A;
3) public class Test
4) {
5)     public static void main(String[] args)
6)     {
7)         System.out.println("moduleB accessing members of moduleA");
8)         A a = new A();
9)         a.m1();
10)    }
11) }
```

### module-info.java:

```
1) module moduleB
2) {
3)     requires moduleA;
4) }
```

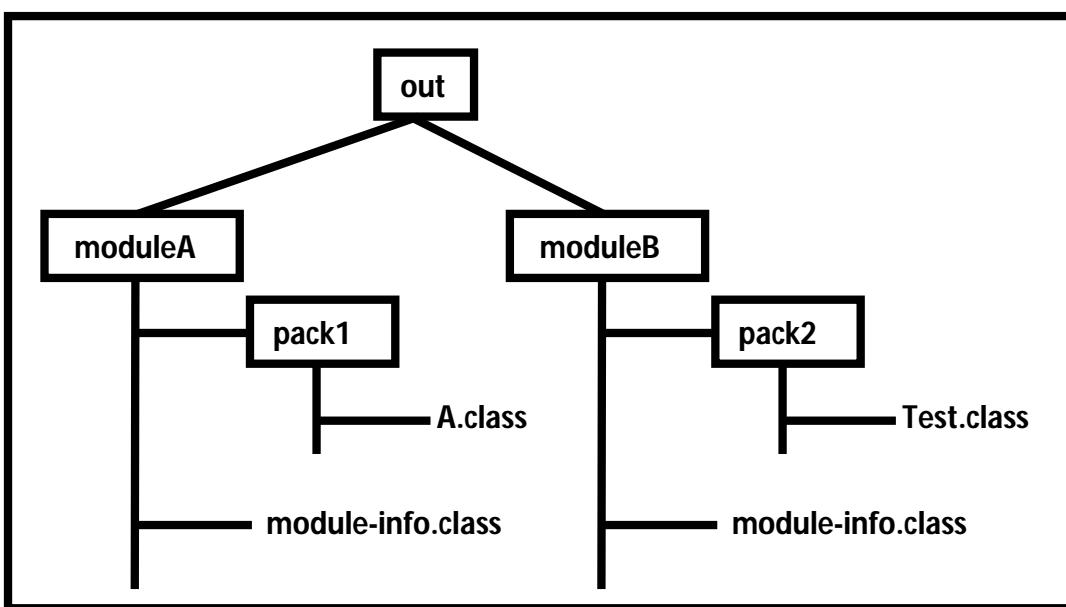


## Compilation:

```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m moduleA,moduleB
```

Note: space is not allowed between the module names otherwise we will get error.

```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m moduleA, moduleB
error: Class names, 'moduleB', are only accepted if annotation processing is explicitly requested
```



## Execution:

```
C:\Users\Durga\Desktop>java --module-path out -m moduleB/pack2.Test
```

## Output:

moduleB accessing members of moduleA  
Method of moduleA

## Case-1:

Even though class A is public, if moduleA won't export pack1, then moduleB cannot access A class.

### Eg:

```
1) module moduleA
2) {
3)     //exports pack1;
4) }
```

```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m moduleA,moduleB
src\moduleB\pack2\Test.java:2: error: package pack1 is not visible
import pack1.A;
```

(package pack1 is declared in module moduleA, which does not export it)  
1 error

## Case-2:

We have to export only packages. If we are trying to export modules or classes then we will get compile time error.



## Eg-1: exporting module instead of package

```
1) module moduleA  
2) {  
3)     exports moduleA;  
4) }
```

```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m moduleA, moduleB  
src\moduleB\pack2\Test.java:2: error: package pack1 is not visible  
import pack1.A;  
^  
(package pack1 is declared in module moduleA, which does not export it)  
src\moduleA\module-info.java:3: error: package is empty or does not exist: moduleA  
    exports moduleA;
```

In this case compiler considers moduleA as package and it is trying to search for that package.

## Eg-2: exporting class instead of package:

```
1) module moduleA  
2) {  
3)     exports pack1.A;  
4) }
```

```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m moduleA, moduleB  
src\moduleB\pack2\Test.java:2: error: package pack1 is not visible  
import pack1.A;  
^  
(package pack1 is declared in module moduleA, which does not export it)  
src\moduleA\module-info.java:3: error: package is empty or does not exist: pack1.A  
    exports pack1.A;  
^
```

2 errors

In this case compiler considers pack1.A as package and it is trying to search for that package.

## Case-3:

If moduleB won't use "requires moduleA" directive then moduleB is not allowed to use members of moduleA, even though moduleA exports.

```
1) module moduleB  
2) {  
3)     //requires moduleA;  
4) }
```

```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m moduleA, moduleB  
src\moduleB\pack2\Test.java:2: error: package pack1 is not visible
```

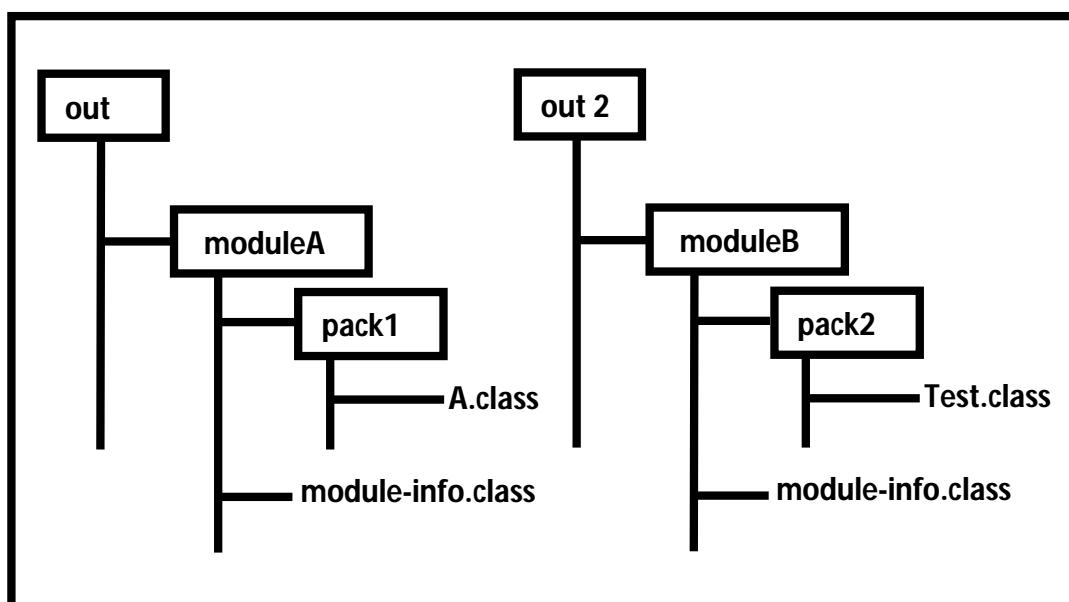


```
import pack1.A;  
^  
(package pack1 is declared in module moduleA, but module moduleB does not read it)  
1 error
```

## Case-4:

If compiled codes are available in different packages then how to run?  
We have to use special option: --upgrade-module-path

If compiled codes of moduleA is available in out and compiled codes of moduleB available in out2



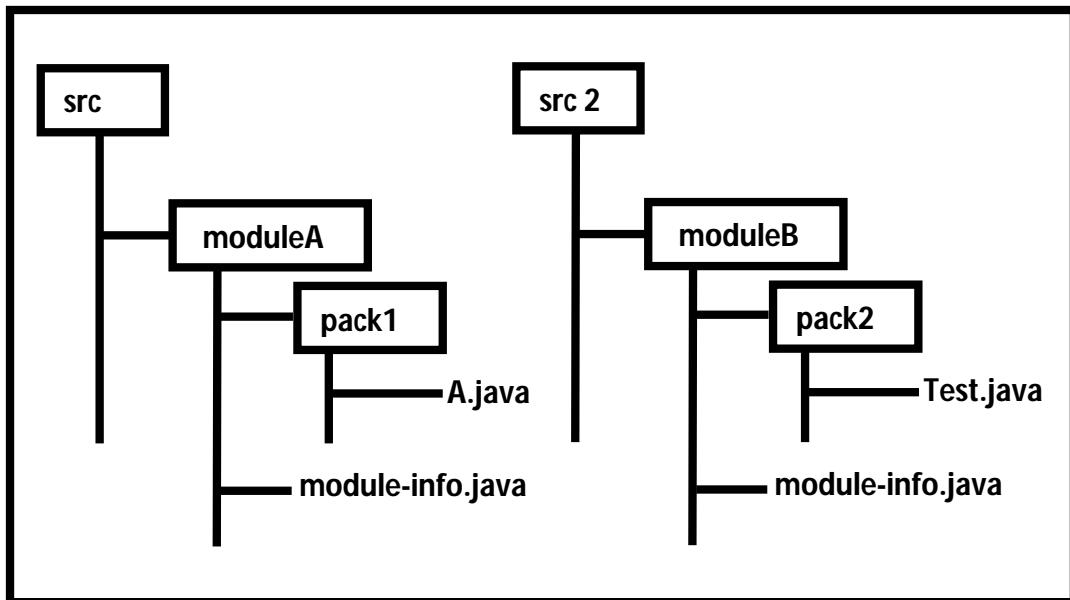
```
C:\Users\Durga\Desktop>java --upgrade-module-path out;out1 -m moduleB/pack2.Test  
moduleB accessing members of moduleA  
Method of moduleA
```



## Case-5:

**If source codes of two modules are in different directories then how to compile?**

Assume moduleA source code is available in src directory and moduleB source code is available in src2



C:\Users\Durga\Desktop>javac --module-source-path src;src2 -d out -m moduleA, moduleB

## Q. Which of the following are meaningful?

- 1) module moduleName
- 2) {
- 3)   1. requires modulename;
- 4)   2. requires modulename.packagename;
- 5)   3. requires modulename.packagename.classname;
- 6)   4. exports modulename;
- 7)   5. exports packagename;
- 8)   6. exports packagename.classname;
- 9) }

Answer: 1 & 5 are Valid

**Note:** We can use exports directive only for packages but not modules and classes, and we can use requires directive only for modules but not for packages and classes.

**Note:** To access members of one module in other module, compulsory we have to take care the following 3 things.

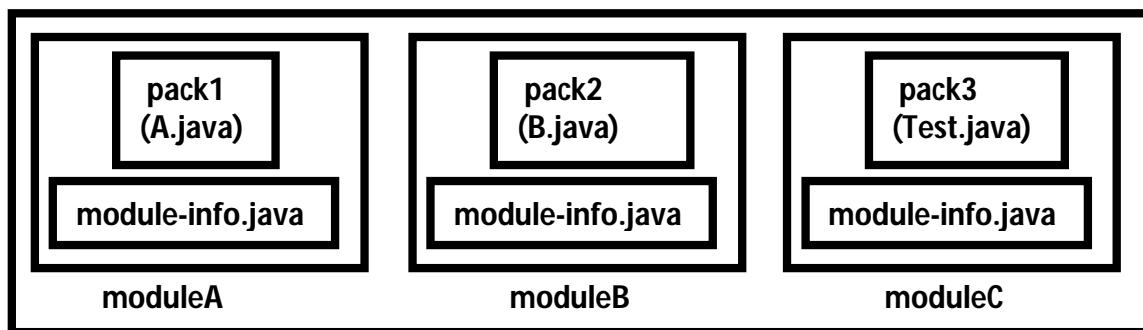
1. The module which is accessing must have requires dependency
2. The module which is providing functionality must have exports dependency
3. The member must be public.



## JPMS vs NoClassDefFoundError:

In Java 9 Platform Modular System, JVM will check all dependencies at the beginning only. If any dependent module is missing then JVM won't start its execution. Hence there is no chance of NoClassDefFoundError in the middle of Program execution.

### Demo Program:



## Components of moduleA:

### A.java:

```
1) package pack1;
2) public class A
3) {
4)     public void m1()
5)     {
6)         System.out.println("Method of moduleA");
7)     }
8) }
```

### module-info.java:

```
1) module moduleA
2) {
3)     exports pack1;
4) }
```



## Components of moduleB:

### B.java:

```
1) package pack2;
2) import pack1.A;
3) public class B
4) {
5)     public void m2()
6)     {
7)         System.out.println("Method of moduleB");
8)         A a = new A();
9)         a.m1();
10)    }
11)}
```

### module-info.java:

```
1) module moduleB
2) {
3)     requires moduleA;
4)     exports pack2;
5) }
```

## Components of moduleC:

### Test.java:

```
1) package pack3;
2) import pack2.B;
3) public class Test
4) {
5)     public static void main(String[] args)
6)     {
7)         System.out.println("Test class main method");
8)         B b = new B();
9)         b.m2();
10)    }
11)}
```

### module-info.java:

```
1) module moduleC
2) {
3)     requires moduleB;
4) }
```



## Compilation and Execution:

```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m moduleA, moduleB, moduleC
```

```
C:\Users\Durga\Desktop>java --module-path out -m moduleC/pack3.Test  
Test class main method
```

Method of moduleB  
Method of moduleA

If we delete compiled code of module (inside out folder), then JVM will raise error at the beginning only and JVM won't start program execution.

```
C:\Users\Durga\Desktop>java --module-path out -m moduleC/pack3.Test  
Error occurred during initialization of boot layer  
java.lang.module.FindException: Module moduleA not found, required by moduleB
```

But in Non Modular programming, JVM will start execution and in the middle, it will raise NoClassDefFoundError.

Hence in Java Platform Module System, there is no chance of getting NoClassDefFoundError in the middle of program execution.

## Transitive Dependencies (requires with transitive Keyword):

A→B, B→C ==> A→C

This property in mathematics is called Transitive Property.

Student1 requires Material, only for himself, if any other person asking he won't share.

```
1) module student1  
2) {  
3)     requires material;  
4) }
```

"Student1 requires material not only for himself, if any other person asking him, he will share it"

```
1) module Student1  
2) {  
3)     requires transitive material;  
4) }
```

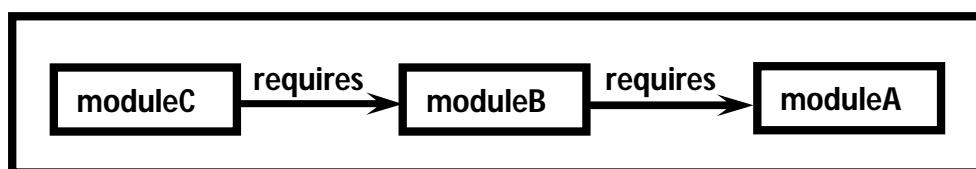
Sometimes module requires the components of some other module not only for itself and for the modules that require that module also. For this requirement we can use transitive keyword.

The transitive keyword says that "Whatever I have will be given to a module that asks me."



## Case-1:

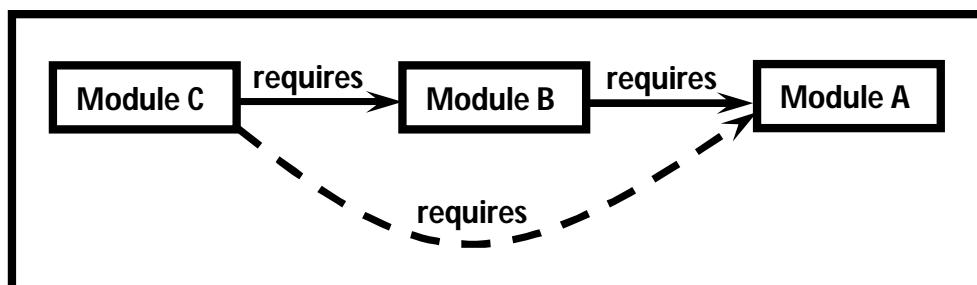
```
1) module moduleA
2) {
3)   exports pack1;
4) }
5) module moduleB
6) {
7)   requires moduleA;
8) }
9) module moduleC
10){
11)   requires moduleB;
12})
```



In this case only moduleB is available to moduleC and moduleA is not available. Hence moduleC cannot use the members of moduleA directly.

## Case-2:

```
1) module moduleA
2) {
3)   exports pack1;
4) }
5) module moduleB
6) {
7)   requires transitive moduleA;
8) }
9) module moduleC
10){
11)   requires moduleB;
12})
```

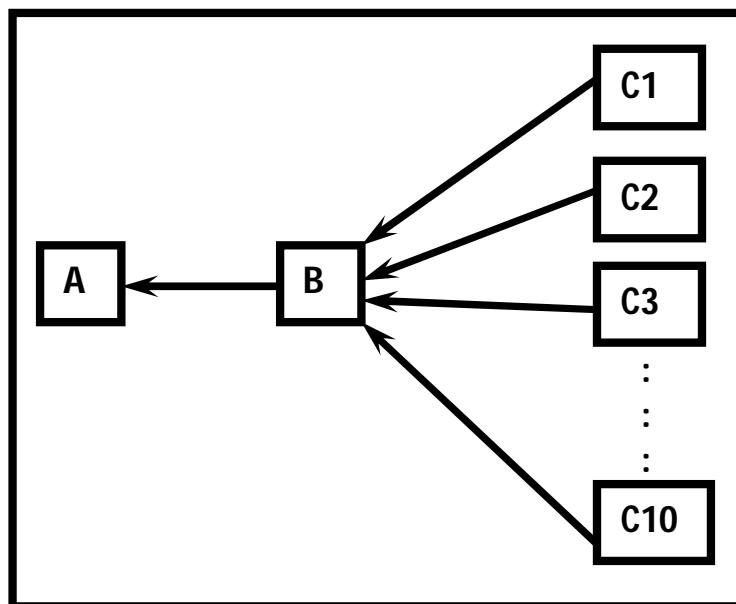


In this both moduleB and moduleA are available to moduleC. Now moduleC can use members of both modules directly.



## Case Study:

Assume Modules C1, C2,...C10 requires Module B and Module B requires A.



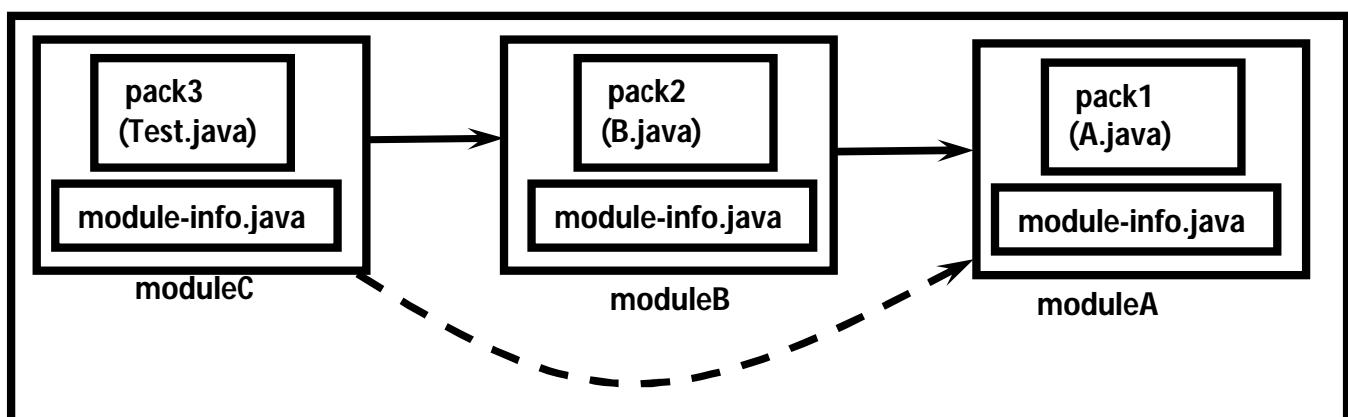
If we write "requires transitive A" inside module B

```
1) module B  
2) {  
3)     requires transitive A;  
4) }
```

Then module A is by default available to C1, C2,.., C10 automatically. Inside every module of C, we are not required to use "requires A" explicitly. Hence transitive keyword promotes code reusability.

Note: Transitive means implied readability i.e., Readability will be continues to the next level.

## Demo Program for transitive keyword:





## A.java:

```
1) package pack1;
2) public class A
3) {
4)     public void m1()
5)     {
6)         System.out.println("moduleA method");
7)     }
8) }
```

## module-info.java:

```
1) module moduleA
2) {
3)     exports pack1;
4) }
```

## moduleB components:

### B.java:

```
1) package pack2;
2) import pack1.A;
3) public class B
4) {
5)     public A m2()
6)     {
7)         System.out.println("moduleB method");
8)         A a = new A();
9)         return a;
10)    }
11) }
```

### module-info.java:

```
1) module moduleB
2) {
3)     requires transitive moduleA;
4)     exports pack2;
5) }
```



## moduleC components:

### Test.java:

```
1) package pack3;
2) import pack2.B;
3) public class Test
4) {
5)     public static void main(String[] args)
6)     {
7)         System.out.println("Test class main method");
8)         B b = new B();
9)         b.m2().m1();
10)    }
11)}
```

### module-info.java:

```
1) module moduleC
2) {
3)     requires moduleB;
4) }
```

```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m moduleA, moduleB, moduleC
```

```
C:\Users\Durga\Desktop>java --module-path out -m moduleC/pack3.Test
Test class main method
moduleB method
moduleA method
```

In the above program if we are not using transitive keyword then we will get compile time error because moduleA is not available to moduleC.

```
javac --module-source-path src -d out -m moduleA,moduleB,moduleC
src\moduleC\pack3\Test.java:9: error: A.m1() in package pack1 is not accessible
    b.m2().m1();
           ^
(package pack1 is declared in module moduleA, but module moduleC does not read it)
```

## Optional Dependencies (Requires Directive with static keyword):

If Dependent Module should be available at compile time but optional at runtime, then such type of dependency is called Otonal Dependency. We can specify optional dependency by using static keyword.

Syntax: requires static <modulename>

The static keyword is used to say that, "This dependency check is mandatory at compile time and optional at runtime."



## Eg1:

```
1) module moduleB  
2) {  
3)     requires moduleA;  
4) }
```

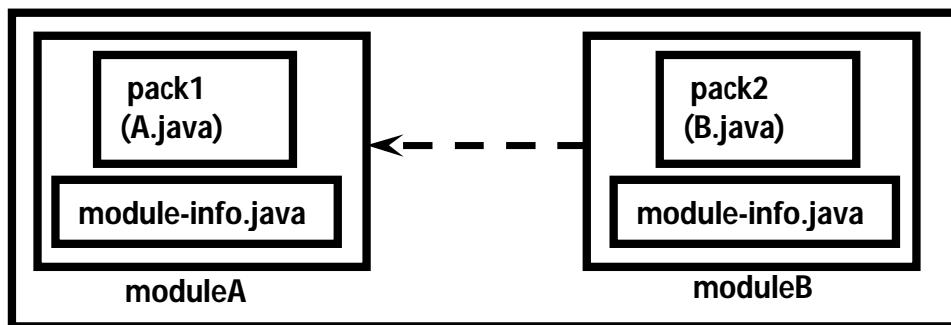
moduleA should be available at the time of compilation and runtime. It is not optional dependency.

## Eg2:

```
1) module moduleB  
2) {  
3)     requires static moduleA;  
4) }
```

At the time of compilation moduleA should be available, but at runtime it is optional. i.e., at runtime even moduleA is not available JVM will execute code.

## Demo Program for Optional Dependency:



## moduleA components:

### A.java:

```
1) package pack1;  
2) public class A  
3) {  
4)     public void m1()  
5)     {  
6)         System.out.println("moduleA method");  
7)     }  
8) }
```

### module-info.java:

```
1) module moduleA {  
2)     exports pack1;  
3) }
```



## moduleB components:

### Test.java:

```
1) package pack2;
2) public class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         System.out.println("Optional Dependencies Demo!!!");
7)
8)     }
9) }
```

### module-info.java:

```
1) module moduleB
2) {
3)     requires static moduleA;
4) }
```

At the time of compilation both modules should be available. But at runtime, we can run moduleB Test class, even moduleA compiled classes are not available i.e., moduleB having optional dependency with moduleA.

```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m moduleA,moduleB
```

```
C:\Users\Durga\Desktop>java --module-path out -m moduleB/pack2.Test
Optional Dependencies Demo!!!
```

If we remove static keyword and at runtime if we delete compiled classes of moduleA, then we will get error.

```
C:\Users\Durga\Desktop>java --module-path out -m moduleB/pack2.Test
Error occurred during initialization of boot layer
java.lang.module.FindException: Module moduleA not found, required by moduleB
```

## Use cases of Optional Dependencies:

Usage of optional dependencies is very common in Programming world.

Sometimes we can develop library with optional dependencies.

Eg 1: If apache http Client is available use it, otherwise use HttpURLConnection.

Eg 2: If oracle module is available use it, otherwise use mysql module.



Why we should do this? For various reasons –

1. When distributing a library and we may not want to force a big dependency to the client.
2. On the other hand, a more advanced library may have performance benefits, so whatever module client needs, he can use.
3. We may want to allow easily pluggable implementations of some functionality. We may provide implementations using all of these, and pick the one whose dependency is found.

## Q. What is the difference between the following?

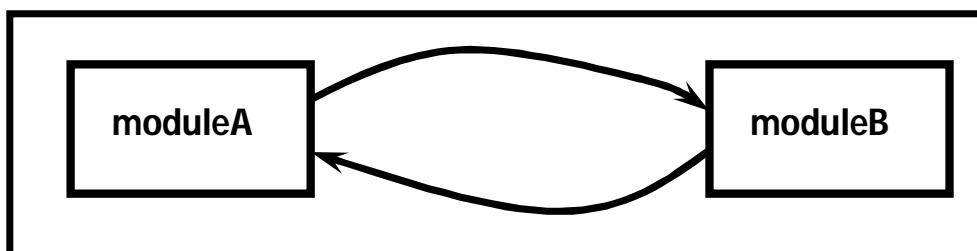
```
1) module moduleB
2) {
3)   1. requires moduleA;
4)   2. requires transitive moduleA
5)   3. requires static moduleA
6) }
```

## Cyclic Dependencies:

If moduleA depends on moduleB and moduleB depends on moduleA, such type of dependency is called cyclic dependency.

Cyclic Dependencies between the modules are not allowed in java 9.

### Demo Program:



### moduleA components:

#### module-info.java

```
1) module moduleA
2) {
3)   requires moduleB;
4) }
```



## moduleB components:

### module-info.java

```
1) module moduleB
2) {
3)     requires moduleA;
4) }
```

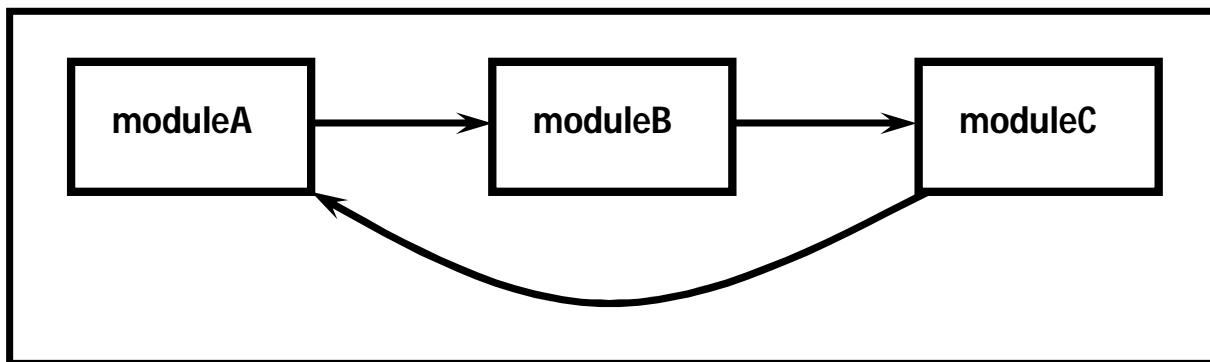
```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m moduleA, moduleB
src\moduleB\module-info.java:3: error: cyclic dependence involving moduleA
    requires moduleA;
```

There may be a chance of cyclic dependency between more than 2 modules also.

moduleA requires moduleB

moduleB requires moduleC

moduleC requires moduleA

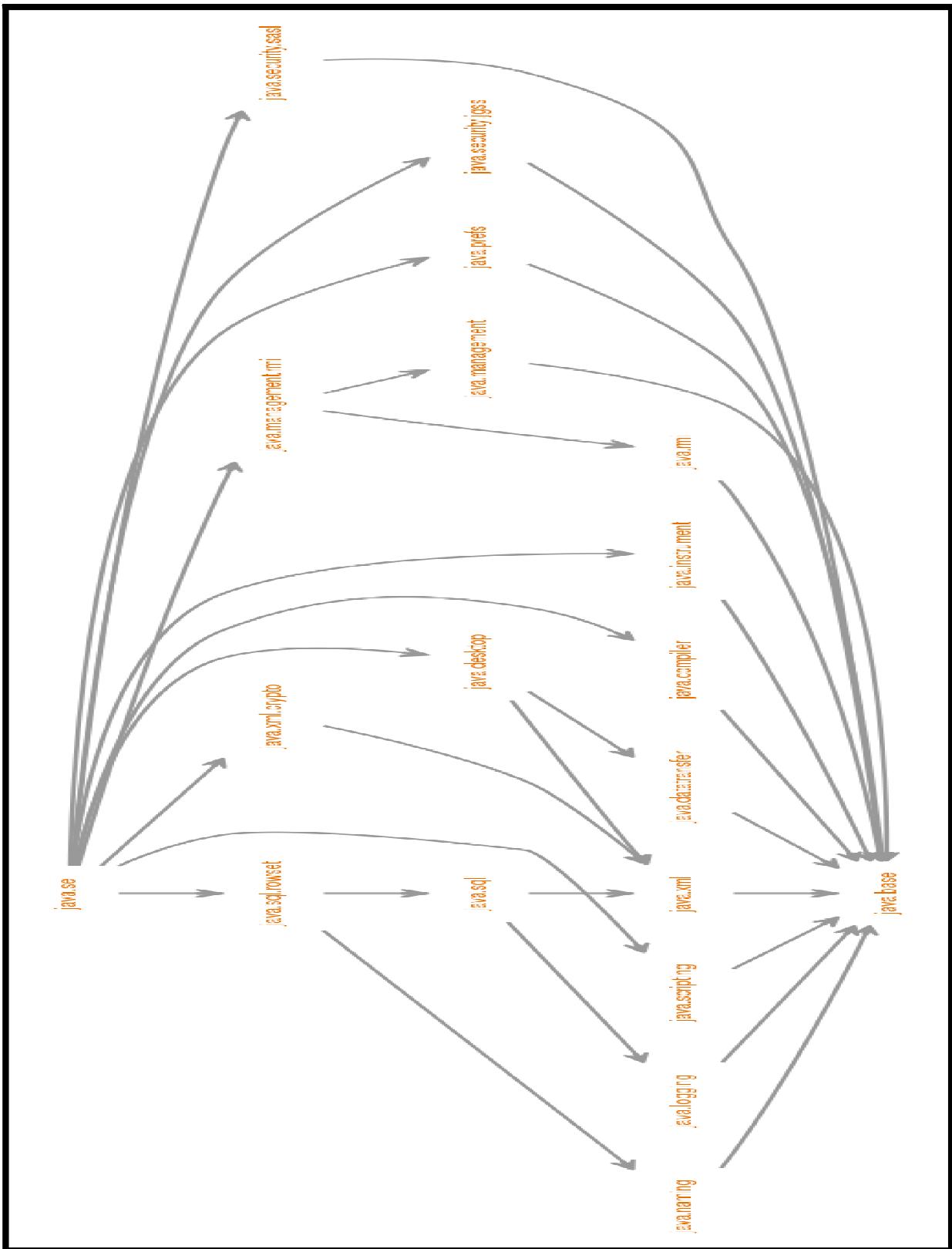


**Note:** In all predefined modules also, there is no chance of cyclic dependency



# Core Java

**DURGA**  
SOFTWARE SOLUTIONS®





## Qualified Exports:

Sometimes a module can export its package to specific module instead of every module. Then the specified module only can access. Such type of exports are called Qualified Exports.

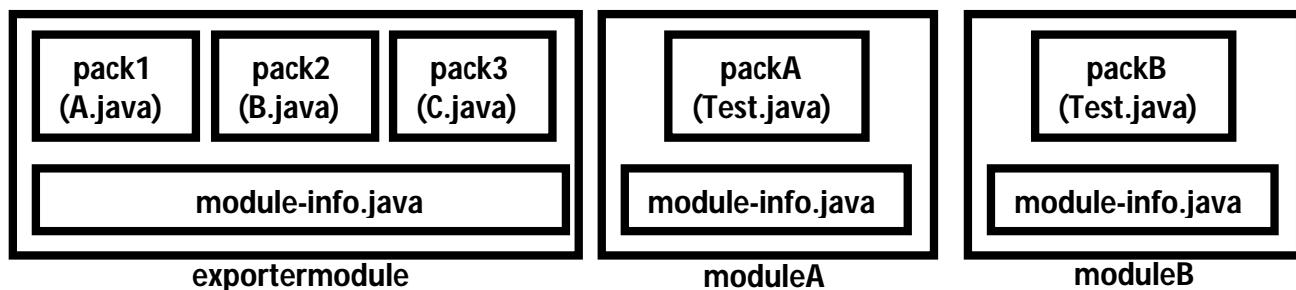
### Syntax:

exports <pack1> to <module1>,<module2>,...

### Eg:

```
1) module moduleA
2) {
3)     exports pack1; //to export pack1 to all modules
4)     exports pack1 to moduleA; // to export pack1 only for moduleA
5)     exports pack1 to moduleA,moduleB; // to export pack1 for both moduleA,mo
       duleB
6) }
```

## Demo Program for Qualified Exports:



## Components of exportermodule:

### A.java:

```
1) package pack1;
2) public class A
3) {
4) }
```

### B.java:

```
1) package pack2;
2) public class B
3) {
4) }
```



## C.java:

```
1) package pack3;  
2) public class C  
3) {  
4) }
```

## module-info.java:

```
1) module exportermodule  
2) {  
3)     exports pack1;  
4)     exports pack2 to moduleA;  
5)     exports pack3 to moduleA,moduleB;  
6) }
```

|       | moduleA | moduleB |
|-------|---------|---------|
| pack1 | ✓       | ✓       |
| pack2 | ✓       | ✗       |
| pack3 | ✓       | ✓       |

## Components of moduleA:

### Test.java:

```
1) package packA;  
2) import pack1.A;  
3) import pack2.B;  
4) import pack3.C;  
5) public class Test  
6) {  
7)     public static void main(String[] args)  
8)     {  
9)         System.out.println("Qualified Exports Demo");  
10)    }  
11)}
```

## module-info.java:

```
1) module moduleA  
2) {  
3)     requires exportermodule;  
4) }
```

## Explanation:

For moduleA, all 3 packages are available. Hence we can compile and run moduleA successfully.

C:\Users\Durga\Desktop>javac --module-source-path src -d out -m exportermodule, moduleA

C:\Users\Durga\Desktop>java --module-path out -m moduleA/packA.Test

Qualified Exports Demo



## Components of moduleB:

### Test.java:

```
1) package packB;
2) import pack1.A;
3) import pack2.B;
4) import pack3.C;
5) public class Test
6) {
7)     public static void main(String[] args)
8)     {
9)         System.out.println("Qualified Exports Demo");
10)    }
11)}
```

### module-info.java:

```
1) module moduleB
2) {
3)     requires exportermodule;
4) }
```

### Explanation:

For moduleB, only pack1 and pack3 are available. pack2 is not available. But in moduleB we are trying to access pack2 and hence we will get compile time error.

```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m exportermodule, moduleB
src\moduleB\packB\Test.java:3: error: package pack2 is not visible
import pack2.B;
^
(package pack2 is declared in module exportermodule, which does not export it to module
moduleB)
1 error
```

### Q. Which of the following directives are valid inside module-info.java:

1. requires moduleA;
2. requires moduleA,moduleB;
3. requires moduleA.pack1;
4. requires moduleA.pack1.A;
5. requires static moduleA;
6. requires transitive moduleA;
7. exports pack1;
8. exports pack1,pack2;
9. exports moduleA;
10. exports moduleA.pack1.A;



11. exports pack1 to moduleA;
12. exports pack1 to moduleA,moduleB;

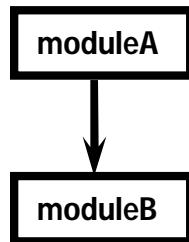
Answers: 1,5,6,7,11,12

## Module Graph:

The dependencies between the modules can be represented by using a special graph, which is nothing but Module Graph.

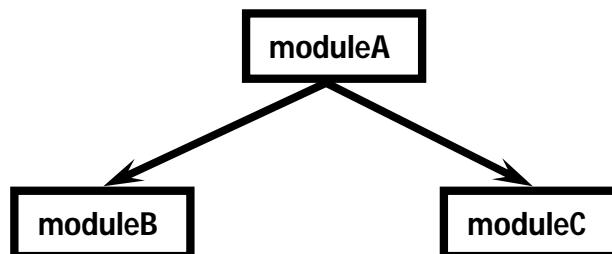
Eg1: If moduleA requires moduleB then the corresponding module graph is :

```
1) module moduleA  
2) {  
3)     requires moduleB;  
4) }
```



Eg 2: If moduleA requires moduleB and moduleC then the corresponding module graph is:

```
1) module moduleA  
2) {  
3)     requires moduleB;  
4)     requires moduleC;  
5) }
```

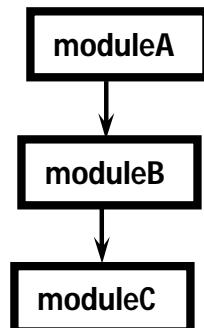


Eg 3: If moduleA requires moduleB and moduleB requires moduleC then the corresponding module graph is:

```
1) module moduleA  
2) {  
3)     requires moduleB;  
4) }  
5) module moduleB  
6) {  
7)     requires moduleC;
```

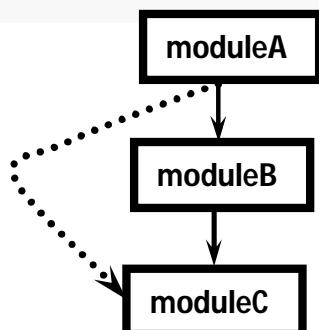


8) }



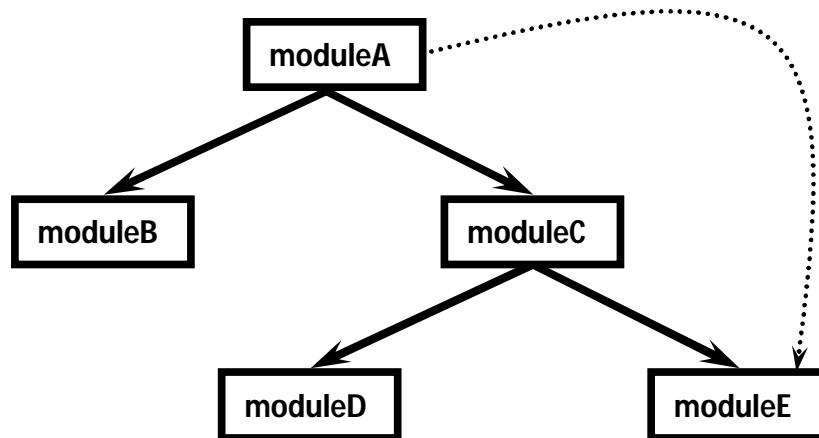
Eg 4: If moduleA requires moduleB and moduleB requires transitive moduleC then the corresponding module graph is:

```
1) module moduleA  
2) {  
3)     requires moduleB;  
4) }  
5) module moduleB  
6) {  
7)     requires transitive moduleC;  
8) }
```



Eg 5: If moduleA requires moduleB and moduleC, moduleC requires moduleD and transitive moduleE then the corresponding Modular Graph is:

```
1) module moduleA  
2) {  
3)     requires moduleB;  
4)     requires moduleC;  
5) }  
6) module moduleC  
7) {  
8)     requires moduleD;  
9)     requires transitive moduleE;  
10) }
```

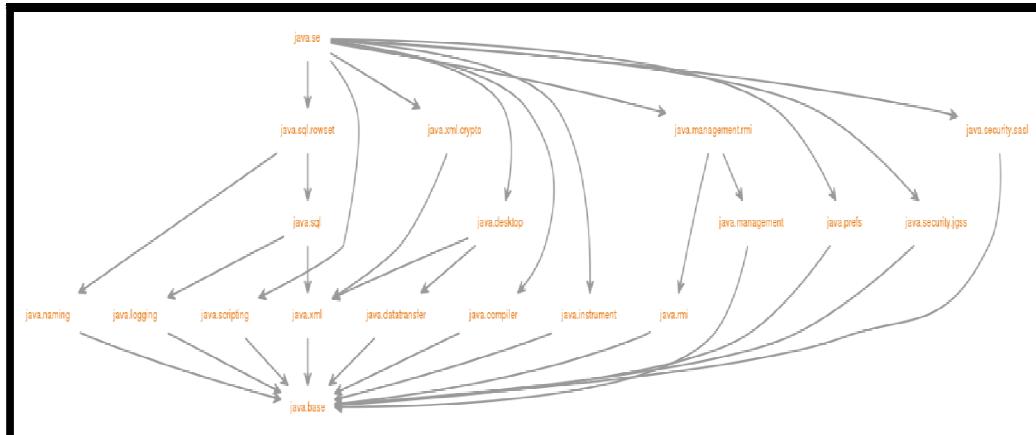


Java 9 JDK itself modularized. All classes of Java SE are divided into several modules.

Eg:

java.base  
java.sql  
java.xml  
java.rmi  
etc...

The module graph of JDK is



In the above diagram all modules require java.base module either directly or indirectly. Hence this module acts as BASE module for all java modules.

Observe modular graphs carefully:java.se and java.sql modules etc

## Rules of Module Graph:

1. Two modules with the same name is not allowed.
2. Cyclic Dependency is not allowed between the modules and hence Module Graph should not contain cycles.



## Observable Modules:

The modules which are observed by JVM at runtime are called Observable modules.

The modules we are specifying with --module-path option with java command are observed by JVM and hence these are observable modules.

```
java --module-path out -m moduleA/pack1.Test
```

The modules present in module-path out are observable modules

JDK itself contains several modules (like java.base, java.sql, java.rmi etc). These modules can be observed automatically by JVM at runtime and we are not required to use --module-path. Hence these are observable modules.

Observable Modules = All Predefined JDK Modules + The modules specified with --module-path option

We can list out all Observable Modules by using --list-modules option with java command.

**Eg1:** To print all readymade compiled modules (pre defined modules) present in Java 9

```
C:\Users\Durga\Desktop>java --list-modules
java.activation@9
java.base@9
java.compiler@9
....
```

**Eg2:** Assume our own created compiled modules are available in out folder. To list out these modules including readymade java modules

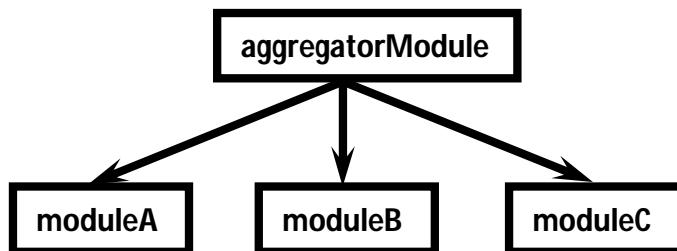
```
C:\Users\Durga\Desktop>java --module-path out --list-modules
java.activation@9
java.base@9
...
exportermodule file:///C:/Users/Durga/Desktop/out/exportermodule/
moduleA file:///C:/Users/Durga/Desktop/out/moduleA/
```



## Aggregator Module:

Sometimes a group of modules can be reused by multiple other modules. Then it is not recommended to read each module individually. We can group those common modules into a single module, and we can read that module directly. This module which aggregates functionality of several modules into a single module is called Aggregator module. If any module reads aggregator module then automatically all its modules are by default available to that module.

Aggregator module won't provide any functionality by its own, just it gathers and bundles together a bunch of other modules.



```
1) module aggregatorModule  
2) {  
3)   requires transitive moduleA;  
4)   requires transitive moduleB;  
5)   requires transitive moduleC;  
6) }
```

Aggregator Module not required to contain a single java class. Just it "requires transitive" of all common modules.

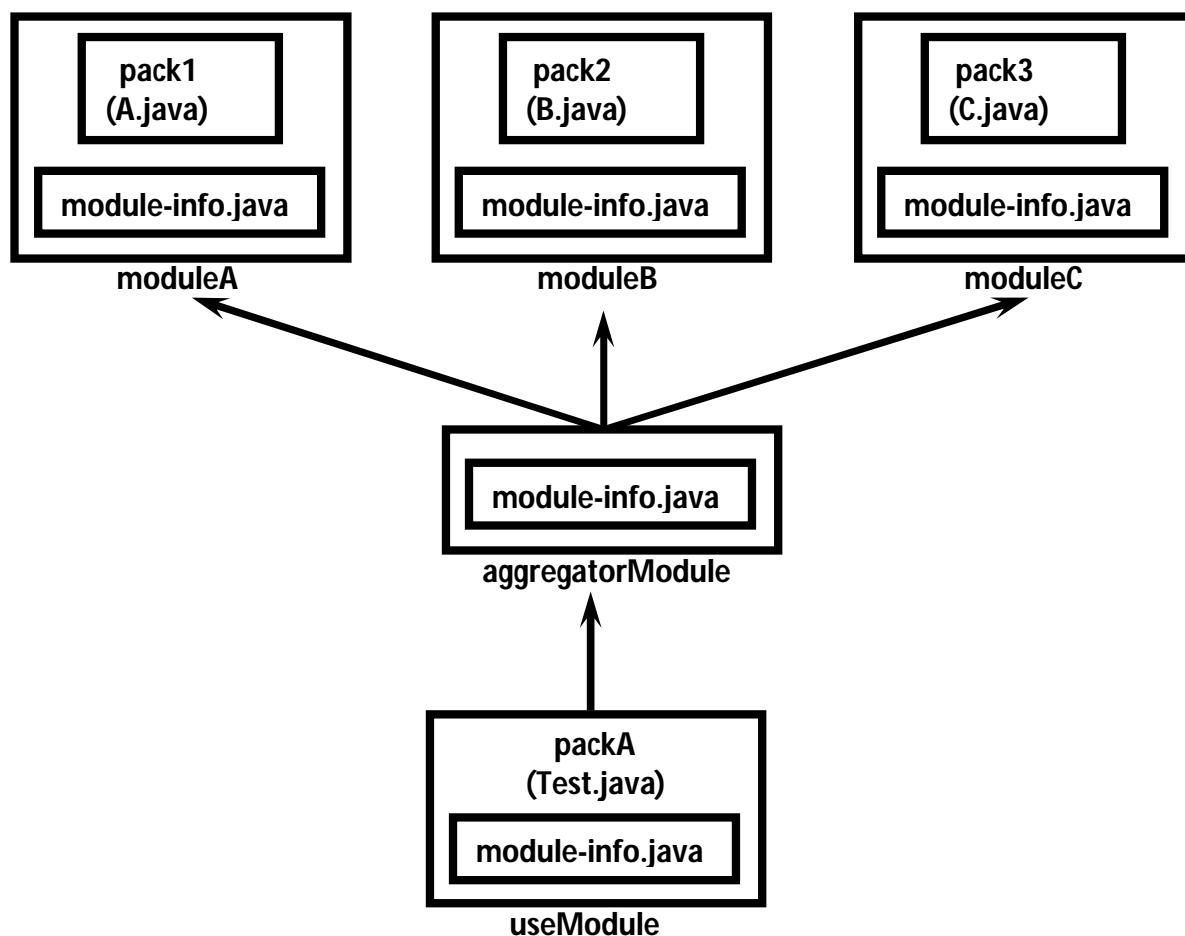
If any module reads aggregatorModule automatically all 3 modules are by default available to that module also.

```
1) module useModule  
2) {  
3)   requires aggregatorModule;  
4) }
```

Now useModule can use functionality of all 3 modules moduleA, moduleB and moduleC.



## Demo Program for Aggregator Module:



## moduleA components:

### A.java:

```
1) package pack1;
2) public class A
3) {
4)     public void m1()
5)     {
6)         System.out.println("moduleA method");
7)     }
8) }
```

### module-info.java:

```
1) module moduleA
2) {
3)     exports pack1;
4) }
```



## moduleB components:

### B.java:

```
1) package pack2;
2) public class B
3) {
4)     public void m1()
5)     {
6)         System.out.println("moduleB method");
7)     }
8) }
```

### module-info.java:

```
1) module moduleB
2) {
3)     exports pack2;
4) }
```

## moduleC components:

### C.java:

```
1) package pack3;
2) public class C
3) {
4)     public void m1()
5)     {
6)         System.out.println("moduleC method");
7)     }
8) }
```

### module-info.java:

```
1) module moduleC
2) {
3)     exports pack3;
4) }
```



## aggregatorModule components:

### module-info.java:

```
1) module aggregatorModule
2) {
3)     requires transitive moduleA;
4)     requires transitive moduleB;
5)     requires transitive moduleC;
6) }
```

## useModule components:

### Test.java:

```
1) package packA;
2) import pack1.A;
3) import pack2.B;
4) import pack3.C;
5) public class Test
6) {
7)     public static void main(String[] args)
8)     {
9)         System.out.println("Aggregator Module Demo");
10)        A a = new A();
11)        a.m1();
12)
13)        B b = new B();
14)        b.m1();
15)
16)        C c = new C();
17)        c.m1();
18)    }
19) }
```

### module-info.java:

Here we are not required to use requires directive for every module, just we have to use requires only for aggregatorModule.

```
1) module useModule
2) {
3)     //requires moduleA;
4)     //requires moduleB;
5)     //requires moduleC;
6)     requires aggregatorModule;
7) }
```



```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m  
moduleA,moduleB,moduleC,aggregatorModule,useModule
```

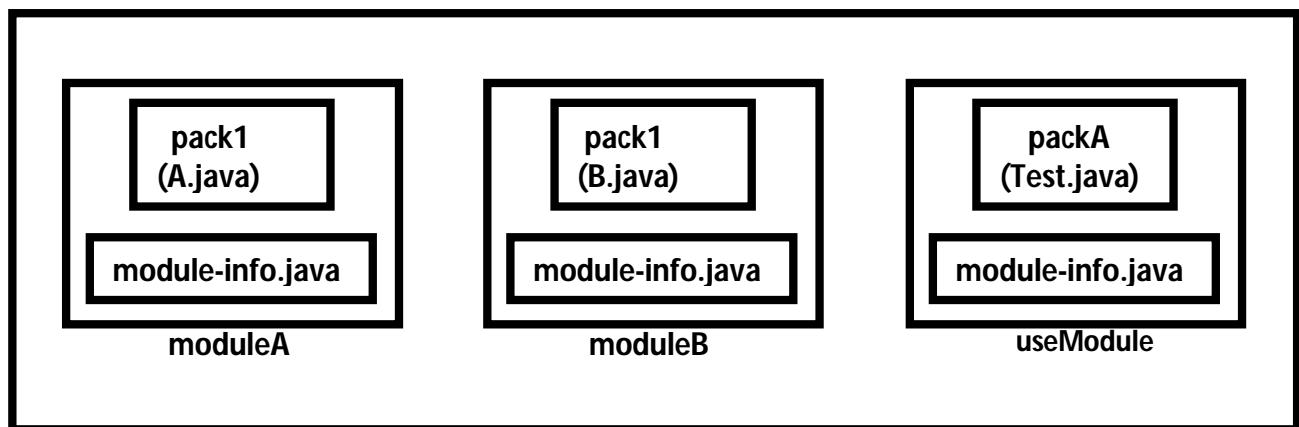
```
C:\Users\Durga\Desktop>java --module-path out -m useModule/packA.Test  
Aggregator Module Demo  
moduleA method  
moduleB method  
moduleC method
```

## Package Naming Conflicts:

Two jar files can contain a package with same name, which may creates version conflicts and abnormal behavior of the program at runtime.

But in Java 9 module System, two modules cannot contain a package with same name; otherwise we will get compile time error. Hence in module system, there is no chance of version conflicts and abnormal behavior of the program.

## Demo Program:



## moduleA components:

### A.java:

```
1) package pack1;  
2) public class A  
3) {  
4)     public void m1()  
5)     {  
6)         System.out.println("moduleA method");  
7)     }  
8) }
```



## module-info.java:

```
1) module moduleA
2) {
3)     exports pack1;
4) }
```

## moduleB components:

### B.java:

```
1) package pack1;
2) public class B
3) {
4)     public void m1()
5)     {
6)         System.out.println("moduleB method");
7)     }
8) }
```

## module-info.java:

```
1) module moduleB
2) {
3)     exports pack1;
4) }
```

## useModule components:

### Test.java:

```
1) package packA;
2) public class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         System.out.println("Package Naming Conflicts");
7)     }
8) }
```

## module-info.java:

```
1) module useModule {
2)     requires moduleA;
3)     requires moduleB;
4) }
```



```
javac --module-source-path src -d out -m moduleA,moduleB,useModule  
error: module useModule reads package pack1 from both moduleA and moduleB
```

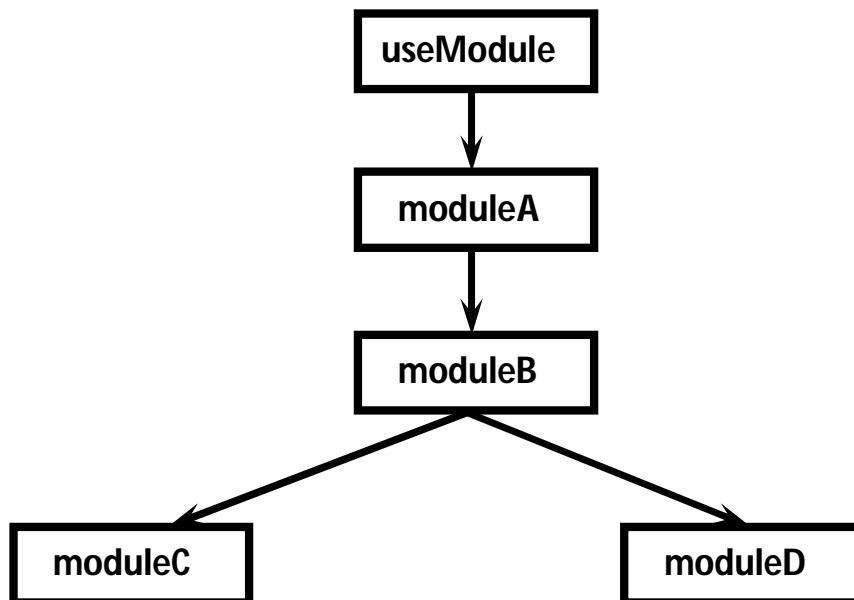
Two modules cannot contain a package with same name.

## Module Resolution Process (MRP):

In the case of traditional classpath, JVM won't check the required .class files at the beginning. While executing program if JVM required any .class file, then only JVM will search in the classpath for the required .class file. If it is available then it will be loaded and used and if it is not available then at runtime we will get NoClassDefFoundError, which is not at all recommended.

But in module programming, JVM will search for the required modules in the module-path before it starts execution. If any module is missing at the beginning only JVM will identify and won't start its execution. Hence in modular programming, there is no chance of getting NoClassDefFoundError in the middle of program execution.

### Demo Program:



### Components of useModule:

#### Test.java:

```
1) package packA;  
2) public class Test  
3) {  
4)     public static void main(String[] args)  
5)     {  
6)         System.out.println("Module Resolution Process(MRP) Demo");  
7)     }  
8) }
```



## module-info.java:

```
1) module useModule  
2) {  
3)     requires moduleA;  
4) }
```

## Components of moduleA:

### module-info.java:

```
1) module moduleA  
2) {  
3)     requires moduleB;  
4) }
```

## Components of moduleB:

### module-info.java:

```
1) module moduleB  
2) {  
3)     requires moduleC;  
4)     requires moduleD;  
5) }
```

## Components of moduleC:

### module-info.java:

```
1) module moduleC  
2) {  
3)  
4) }
```

## Components of moduleD:

### module-info.java:

```
1. module moduleD  
2. {  
3.  
4. }
```

```
javac --module-source-path src -d out -m moduleA,moduleB,moduleC,moduleD,useModule
```

```
java --module-path out --show-module-resolution -m useModule/packA.Test  
The module what we are trying to execute will become root module.
```



---

Root module should contain the class with main method.

The main advantages of Module Resolution Process at beginning are:

1. We will get error if any dependent module is not available.
2. We will get error if multiple modules with the same name
3. We will get error if any cyclic dependency
4. We will get error if two modules contain packages with the same name.

**Note:**

The following are restricted keywords in java 9:

module, requires, transitive, exports

In normal Java program no restrictions and we can use for identifier purpose also.



# JLINK (Java Linker)

Until 1.8 version to run a small Java program (like Hello World program) also, we should use a bigger JRE which contains all java's inbuilt 4300+ classes. It increases the size of Java Runtime environment and Java applications. Due to this Java is not suitable for IOT devices and Micro Services. (No one invite a bigger Elephant into their small house).

To overcome this problem, Java people introduced Compact Profiles in Java 8. But they didn't succeed that much. In Java 9, they introduced a permanent solution to reduce the size of Java Runtime Environment, which is nothing but JLINK.

JLINK is Java's new command line tool (which is available in `JDK_HOME\bin`) which allows us to link sets of only required modules (and their dependencies) to create a runtime image (our own JRE).

Now, our Custom JRE contains only required modules and classes instead of having all 4300+ classes.

It reduces the size of Java Runtime Environment, which makes java best suitable for IOT and micro services.

Hence, Jlink's main intention is to avoid shipping everything and, also, to run on very small devices with little memory. By using Jlink, we can get our own very small JRE.

Jlink also has a list of plugins (like compress) that will help optimize our solutions.

## How to use JLINK: Demo Program

```
src
|-demoModule
  |-module-info.java
  |-packA
    |-Test.java
```

### module-info.java:

```
1) module demoModule
2) {
3) }
```



## Test.java:

```
1) package packA;
2) public class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         System.out.println("JLINK Demo To create our own customized & small JRE");
7)     }
8) }
```

## Compilation:

```
C:\Users\Durga\Desktop>javac --module-source-path src -d out -m demoModule
```

## Run with default JRE:

```
C:\Users\Durga\Desktop>java --module-path out -m demoModule/packA.Test
```

o/p: JLINK Demo To create our own customized & small JRE

## Creation of our own JRE only with required modules:

demoModule requires java.base module. Hence add java.base module to out directory  
(copy java.base.jmod from jdk-9\jmods to out folder)

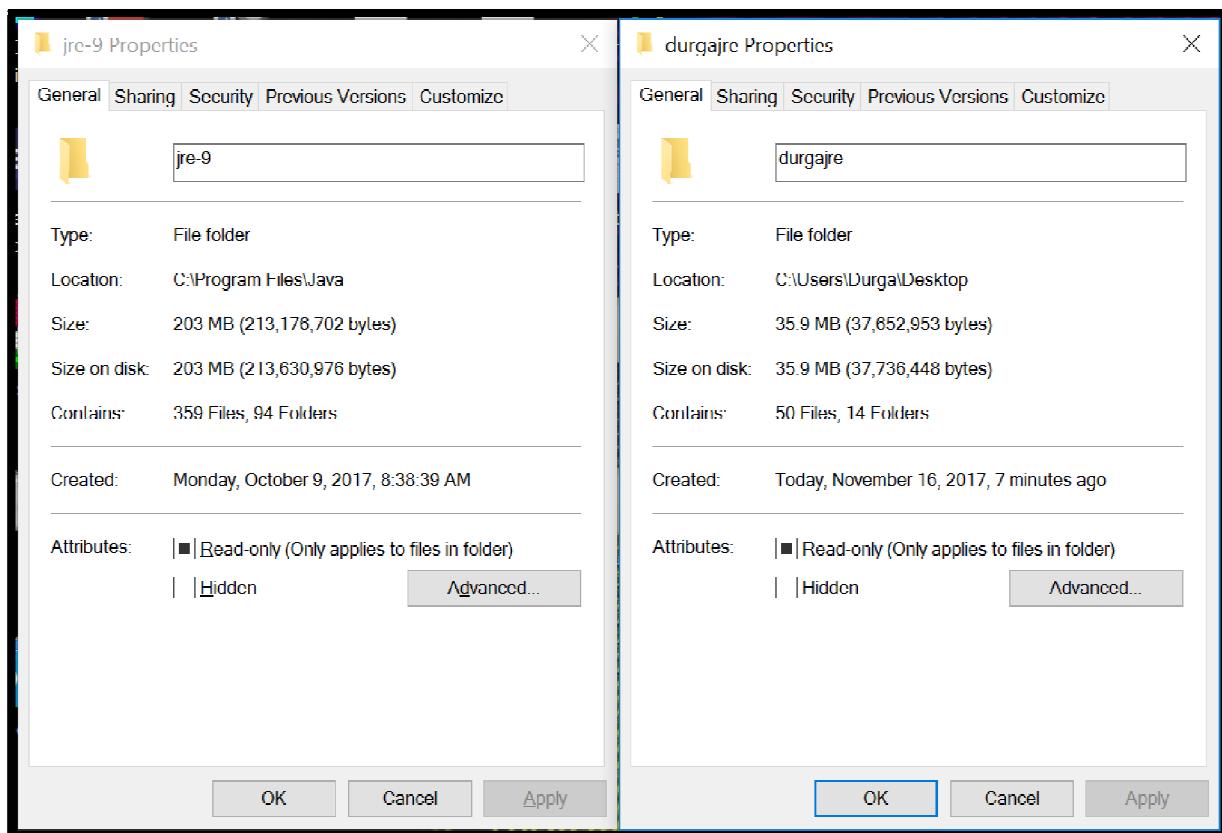
```
out
|-java.base.jmod
|-demoModule
| |-module-info.class
| |-packA
| |-Test.class
```

Now we can create our own JRE with JLINK command

```
C:\Users\Durga\Desktop>jlink --module-path out --add-modules demoModule,java.base --output durgajre
```



Now observe the size of durgajre is just 35.9MB which is very small when compared with default JRE size 203MB.

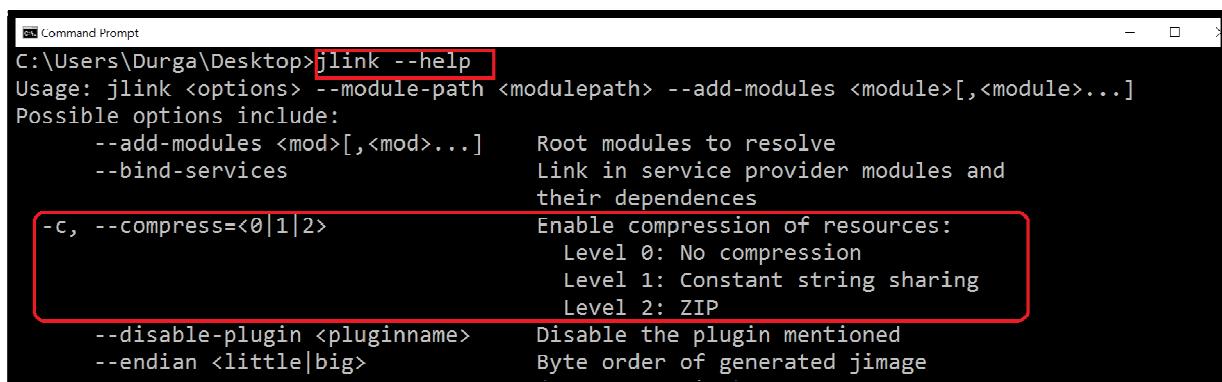


We can run our application with our own custom jre (durgajre) as follows

C:\Users\Durga\Desktop\durgajre\bin>java -m demoModule/packA.Test  
o/p: JLINK Demo To create our own customized & small JRE

## Compressing the size of JRE with compress plugin:

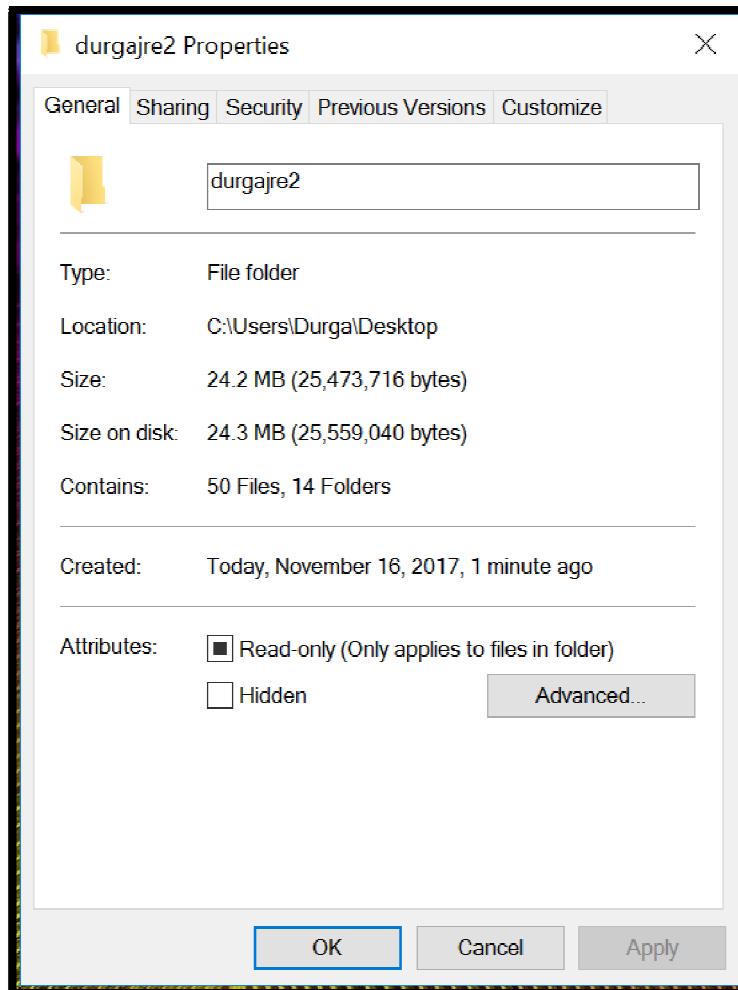
Still we can compress the size of JRE with compress plugin.





```
C:\Users\Durga\Desktop>jlink --module-path out --add-modules demoModule,java.base --compress 2 --output durgajre2
```

```
C:\Users\Durga\Desktop\durgajre2\bin>java -m demoModule/packA.Test  
o/p: JLINK Demo To create our own customized & small JRE
```



### Providing our own name to the application with launcher plugin:

```
C:\Users\Durga\Desktop>jlink --module-path out --add-modules demoModule,java.base --launcher demoapp=demoModule/packA.Test --compress 2 --output durgajre3
```

Now we can run our application only with the name demoapp

```
C:\Users\Durga\Desktop\durgajre3\bin>demoapp  
JLINK Demo To create our own customized & small JRE
```

If we set the path **PATH = C:\Users\Durga\Desktop\durgajre3\bin**

Then we can run our application from anywhere.

D:\>demoapp  
E:\>demoapp



# Process API Updates (JEP-102)

Until java 8, communicating with processor/os/machine is very difficult. We required to write very complex native code and we have to use 3rd party jar files.

The way of communication with processor is varied from system to system (i.e. os to os). For example, in windows one way, but in Mac other way. Being a programmer we have to write code based on operating system, which makes programming very complex.

To resolve this complexity, JDK 9 engineers introduced several enhancements to Process API. By using this Updated API, we can write java code to communicate with any processor very easily. According to worldwide Java Developers, Process API Updates is the number 1 feature in Java 9.

With this Enhanced API, we can perform the following activities very easily.

1. Get the Process ID (PID) of running process.
2. Create a new process
3. Destroy already running process
4. Get the process handles for processes
5. Get the parent and child processes of running process
6. Get the process information like owner, children,...  
etc...

### What's New in Java 9 Process API:

1. Added several new methods (like pid(),info() etc) to Process class.
2. Added several new methods (like startPipeline()) to ProcessBuilder class. We can use ProcessBuilder class to create operating system processes.
3. Introduced a new powerful interface ProcessHandle. With this interface, we can access current running process, we can access parent and child processes of a particular process etc
4. Introduced a new interface ProcessHandle.Info, by using this we can get complete information of a particular process.

Note: All these classes and interfaces are part of java.lang package and hence we are not required to use any import statement.



## How to get ProcessHandle object:

It is the most powerful and useful interface introduced in java 9.

We can get ProcessHandle object as follows

1. `ProcessHandle handle=ProcessHandle.current();`  
Returns the ProcessHandle of current running Process

2. `ProcessHandle handle=p.toHandle();`  
Returns the ProcessHandle of specified Process object.

3. `Optional<ProcessHandle> handle=ProcessHandle.of(PID);`  
Returns the ProcessHandle of process with the specified pid.  
Here, the return type is Optional, because PID may exist or may not exist.

### Use Case-1: To get the process ID (PID) of current process

```
1) public class Test
2) {
3)     public static void main(String[] args) throws Exception
4)     {
5)         ProcessHandle p=ProcessHandle.current();
6)         long pid=p.pid();
7)         System.out.println("The PID of current running JVM instance :"+pid);
8)         Thread.sleep(100000);
9)     }
10) }
```

We can see this process id in Task Manager (alt+ctrl+delete in windows)

## ProcessHandle.Info:

We can get complete information of a particular process by using ProcessHandle.Info object.  
We can get this Info object as follows.

```
ProcessHandle p = ProcessHandle.current();
ProcessHandle.Info info = p.info();
```

Once we got Info object, we can call the following methods on that object.

### 1. user():

Return the user of the process.  
`public Optional<String> user()`

### 2. command():

Returns the command,that can be used to start the process.  
`public Optional<String> command()`



### 3. startInstant():

Returns the start time of the process.

```
public Optional<String> startInstant()
```

### 4. totalCpuDuration():

Returns the total cputime accumulated of the process.

```
public Optional<String> totalCpuDuration()
```

## Use Case-2: To get snapshot of the current running process info

```
1) public class Test
2) {
3)     public static void main(String[] args) throws Exception
4)     {
5)         ProcessHandle p=ProcessHandle.current();
6)         ProcessHandle.Info info=p.info();
7)         System.out.println("Complete Process Inforamtion:\n"+info);
8)         System.out.println("User: "+info.user().get());
9)         System.out.println("Command: "+info.command().get());
10)        System.out.println("Start Time: "+info.startInstant().get());
11)        System.out.println("Total CPU Time Acquired: "+info.totalCpuDuration().get());
12)    }
13) }
```

## Use Case-3: To get snapshot of the Particular Process Based on id

```
1) import java.util.*;
2) public class Test
3) {
4)     public static void main(String[] args) throws Exception
5)     {
6)         Optional<ProcessHandle> opt=ProcessHandle.of(7532);
7)         ProcessHandle p=opt.get();
8)         ProcessHandle.Info info=p.info();
9)         System.out.println("Complete Process Inforamtion:\n"+info);
10)        System.out.println("User: "+info.user().get());
11)        System.out.println("Command: "+info.command().get());
12)        System.out.println("Start Time: "+info.startInstant().get());
13)        System.out.println("Total CPU Time Acquired: "+info.totalCpuDuration().get());
14)    }
15) }
```



## ProcessBuilder:

We can use ProcessBuilder to create processes.

We can create ProcessBuilder object by using the following constructor.

```
public ProcessBuilder(String... command)
```

The argument should be valid command to invoke the process.

Eg:

```
ProcessBuilder pb = new ProcessBuilder("javac","Test.java");
ProcessBuilder pb = new ProcessBuilder("java","Test");
ProcessBuilder pb = new ProcessBuilder("notepad.exe","D:\\names.txt");
```

Once we create a ProcessBuilder object, we can start the process by using start() method.

```
pb.start();
```

## Use Case-4: To create and Start a process by using ProcessBuilder

### FrameDemo.java:

```
1) import java.awt.*;
2) import java.awt.event.*;
3) public class FrameDemo
4) {
5)     public static void main(String[] args)
6)     {
7)         Frame f = new Frame();
8)         f.addWindowListener(new WindowAdapter()
9)         {
10)             public void windowClosing(WindowEvent e)
11)             {
12)                 System.exit(0);
13)             }
14)         });
15)         f.add(new Label("This Process Started from Java by using ProcessBuilder !!!"));
16)         f.setSize(500,500);
17)         f.setVisible(true);
18)     }
19) }
```



## Test.java

```
1) public class Test
2) {
3)     public static void main(String[] args) throws Exception
4)     {
5)         ProcessBuilder pb=new ProcessBuilder("java","FrameDemo");
6)         pb.start();
7)     }
8) }
```

## Use Case-5: To open a file with notepad from java by using ProcessBuilder

```
1) public class Test
2) {
3)     public static void main(String[] args) throws Exception
4)     {
5)         new ProcessBuilder("notepad.exe","FrameDemo.java").start();
6)     }
7) }
```

## Use Case-6: To start and destroy a process from java by using ProcessBuilder

```
1) class Test
2) {
3)     public static void main(String[] args) throws Exception
4)     {
5)         ProcessBuilder pb=new ProcessBuilder("java","FrameDemo");
6)         Process p=pb.start();
7)         System.out.println("Process Started with id:"+p.pid());
8)         Thread.sleep(10000);
9)         System.out.println("Destroying the process with id:"+p.pid());
10)        p.destroy();
11)    }
12) }
```

## Use Case-7: To destroy a process which is not created from Java

```
1) import java.util.*;
2) class Test
3) {
4)     public static void main(String[] args) throws Exception {
5)         Optional<ProcessHandle> optph=ProcessHandle.of(5232);
6)         ProcessHandle ph=optph.get();
7)         ph.destroy();
8)     }
9) }
```



## Use Case-8: To display all running process information

```
1) import java.util.*;
2) import java.util.stream.*;
3) import java.time.*;
4) class Test
5) {
6)     public static void dumpProcessInfo(ProcessHandle p)
7)     {
8)         ProcessHandle.Info info=p.info();
9)         System.out.println("Process Id:"+p.pid());
10)        System.out.println("User: "+info.user().orElse(""));
11)        System.out.println("Command: "+info.command().orElse(""));
12)        System.out.println("Start Time: "+info.startInstant().orElse(Instant.now()).toString());
13)        System.out.println("Total CPU Time Acquired: "+info.totalCpuDuration()
14)                               .orElse(Duration.ofMillis(0)).toMillis());
15)        System.out.println();
16)    }
17)    public static void main(String[] args) throws Exception
18)    {
19)        Stream<ProcessHandle> allp=ProcessHandle.allProcesses();
20)        allp.limit(100).forEach(ph->dumpProcessInfo(ph));
21)    }
22} }
```

## Use Case-9: To display all child process information

```
1) import java.util.stream.*;
2) import java.time.*;
3) class Test
4) {
5)     public static void dumpProcessInfo(ProcessHandle p) {
6)         ProcessHandle.Info info=p.info();
7)         System.out.println("Process Id:"+p.pid());
8)         System.out.println("User: "+info.user().orElse(""));
9)         System.out.println("Command: "+info.command().orElse(""));
10)        System.out.println("Start Time: "+info.startInstant().orElse(Instant.now()).toString());
11)        System.out.println("Total CPU Time Acquired: "+info.totalCpuDuration()
12)                               .orElse(Duration.ofMillis(0)).toMillis());
13)        System.out.println();
14)    }
15)    public static void main(String[] args) throws Exception {
16)        ProcessHandle handle=ProcessHandle.current();
17)        Stream<ProcessHandle> childp=handle.children();
18)        childp.forEach(ph->dumpProcessInfo(ph));
19)    }
20} }
```



**Note:** If Current Process not having any child processes then we won't get any output

## Use Case-10: To perform a task at the time of Process Termination

```
1) import java.util.concurrent.*;
2) public class Test
3) {
4)     public static void main(String[] args) throws Exception
5)     {
6)         ProcessBuilder pb=new ProcessBuilder("java","FrameDemo");
7)         Process p=pb.start();
8)         System.out.println("Process Started with id:"+p.pid());
9)         CompletableFuture<Process> future=p.onExit();
10)        future.thenAccept(p1->System.out.println("Process Terminated with Id:"+p1.pid()));
11)        System.out.println(future.get());
12)    }
13) }
```

### Output [In Normal Termination]:

```
D:\durga_classes>java Test
Process Started with id:4828
Process[pid=4828, exitValue=0]
Process Terminated with Id:4828
```

### Output [In Abnormal Termination alt+ctrl+delete]:

```
D:\durga_classes>java Test
Process Started with id:12512
Process[pid=12512, exitValue=1]
Process Terminated with Id:12512
```

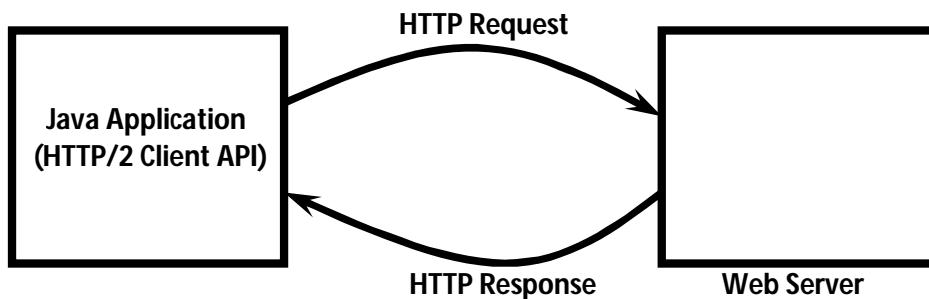
Observe exit value: 0 for normal termination and non-zero for abnormal termination



# HTTP/2 Client

## What is the purpose of HTTP/2 Client:

HTTP/2 Client is one of the most exciting features, for which developers are waiting for long time. By using this new HTTP/2 Client, from Java application, we can send HTTP Request and we can process HTTP Response.



Prior to Java 9, we are using `HttpURLConnection` class to send HTTP Request and to Process HTTP Response. It is the legacy class which was introduced as the part of JDK 1.1 (1997). There are several problems with this `HttpURLConnection` class.

## Problems with Traditional `HttpURLConnection` class:

1. It is very difficult to use.
2. It supports only HTTP/1.1 protocol but not HTTP/2(2015) where
  - A. We can send only one request at a time per TCP Connection, which creates network traffic problems and performance problems.
  - B. It supports only Text data but not binary data
3. It works only in Blocking Mode (Synchronous Mode), which creates performance problems.

Because of these problems, slowly developers started using 3<sup>rd</sup> party Http Clients like Apache Http client and Google Http client etc.

JDK 9 Engineers addresses these issues and introduced a brand new HTTP/2 Client in Java 9.



## Advantages of Java 9 HTTP/2 Client:

1. It is Lightweight and very easy to use.
  2. It supports both HTTP/1.1 and HTTP/2.
  3. It supports both Text data and Binary Data (Streams)
  4. It can work in both Blocking and Non-Blocking Modes (Synchronous Communication and Asynchronous Communication)
  5. It provides better performance and Scalability when compared with traditional HttpURLConnection.
- etc...

## Important Components of Java 9 HTTP/2 Client:

In Java 9, HTTP/2 Client provided as incubator module.

Module: jdk.incubator.httpclient

Package: jdk.incubator.http

Mainly 3 important classes are available:

1. HttpClient
2. HttpRequest
3. HttpResponse

### Note:

Incubator module is by default not available to our java application. Hence compulsory we should read explicitly by using requires directive.

```
1) module demoModule
2) {
3)   requires jdk.incubator.httpclient;
4) }
```

## Steps to send Http Request and process Http Response from Java Application:

1. Create HttpClient Object
2. Create HttpRequest object
3. Send HttpRequest by using HttpClient and Get the HttpResponse
4. Process HttpResponse

### 1. Creation of HttpClient object:

We can use HttpClient object to send HttpRequest to the web server. We can create HttpClient object by using factory method: newHttpClient()

```
HttpClient client = HttpClient.newHttpClient();
```



## 2. Creation of HttpRequest object:

We can create HttpRequest object as follows:

```
String url="http://www.durgasoft.com";
HttpRequest req=HttpRequest.newBuilder(new URI(url)).GET().build();
```

**Note:**

newBuilder() method returns Builder object.  
GET() method sets the request method of this builder to GET.  
build() method builds and returns a HttpRequest.

```
public static HttpRequest.Builder newBuilder(URI uri)
public static HttpRequest.Builder GET()
public abstract HttpRequest build()
```

## 3. Send HttpRequest by using HttpClient and Get the HttpResponse:

HttpClient contains the following methods:

1. send() to send synchronous request(blocking mode)
2. sendAsync() to send Asynchronous Request(Non Blocking Mode)

**Eg:**

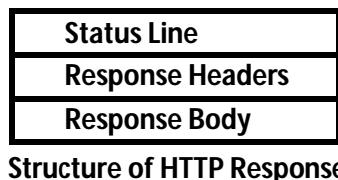
```
HttpResponse resp=client.send(req,HttpResponse.BodyHandler.asString());
HttpResponse resp=client.send(req,HttpResponse.BodyHandler.asFile(Paths.get("abc.txt")));
```

**Note:**

BodyHandler is a functional interface present inside HttpResponse. It can be used to handle body of HttpResponse.

## 4. Process HttpResponse:

HttpResponse contains the status code, response headers and body.



HttpResponse class contains the following methods retrieve data from the response

### 1. statusCode()

Returns status code of the response  
It may be (1XX,2XX,3XX,4XX,5XX)

### 2. body()

Returns body of the response



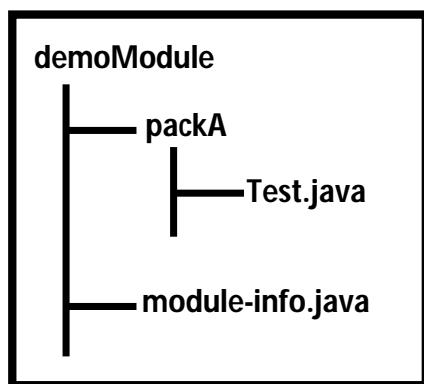
### 3. headers()

Returns header information of the response

Eg:

```
System.out.println("Status Code:"+resp.statusCode());
System.out.println("Body:"+resp.body());
System.out.println("Response Headers Info");
HttpHeaders header=resp.headers();
Map<String,List<String>> map=header.map();
map.forEach((k,v)->System.out.println("\t"+k+":"+v));
```

### Demo Program to send GET Request in Blocking Mode:



#### module-info.java:

```
1) module demoModule
2) {
3)     requires jdk.incubator.httpclient;
4) }
```

#### Test.java:

```
1) package packA;
2) import jdk.incubator.http.HttpClient;
3) import jdk.incubator.http.HttpRequest;
4) import jdk.incubator.http.HttpResponse;
5) import jdk.incubator.http.HttpHeaders;
6) import java.net.URI;
7) import java.util.Map;
8) import java.util.List;
9) public class Test
10) {
11)     public static void main(String[] args) throws Exception
12)     {
13)         String url="https://www.redbus.in/info/aboutus";
14)         sendGetSyncRequest(url);
15)     }
16)     public static void sendGetSyncRequest(String url) throws Exception
```



```
17) {
18)     HttpClient client=HttpClient.newHttpClient();
19)     HttpRequest req=HttpRequest.newBuilder(new URI(url)).GET().build();
20)     HttpResponse resp=client.send(req,HttpResponse.BodyHandler.asString());
21)     processResponse(resp);
22) }
23) public static void processResponse(HttpResponse resp)
24) {
25)     System.out.println("Status Code:" +resp.statusCode());
26)     System.out.println("Response Body:" +resp.body());
27)     HttpHeaders header=resp.headers();
28)     Map<String,List<String>> map=header.map();
29)     System.out.println("Response Headers");
30)     map.forEach((k,v)->System.out.println("\t"+k+ ":" +v));
31) }
32} }
```

**Note:**

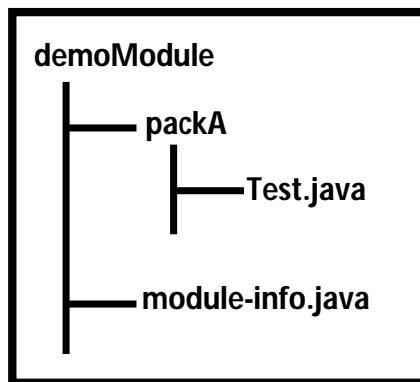
**Writing Http Response body to file abc.html:**

```
HttpResponse resp = client.send(req,HttpResponse.BodyHandler.asFile(Paths.get("abc.html")));
```

Paths is a class present in java.nio.file package and hence we should write import as  
`import java.nio.file.Paths;`

In this case, abc.html file will be created in the current working directory which contains total response body.

**Demo Program:**



**module-info.java:**

```
1) module demoModule
2) {
3)     requires jdk.incubator.httpclient;
4) }
```



## Test.java:

```
1) package packA;
2) import jdk.incubator.http.HttpClient;
3) import jdk.incubator.http.HttpRequest;
4) import jdk.incubator.http.HttpResponse;
5) import jdk.incubator.http.HttpHeaders;
6) import java.net.URI;
7) import java.util.Map;
8) import java.util.List;
9) import java.nio.file.Paths;
10) public class Test
11) {
12)     public static void main(String[] args) throws Exception
13)     {
14)         String url="https://www.redbus.in/info/aboutus";
15)         sendGetSyncRequest(url);
16)     }
17)     public static void sendGetSyncRequest(String url) throws Exception
18)     {
19)         HttpClient client=HttpClient.newHttpClient();
20)         HttpRequest req=HttpRequest.newBuilder(new URI(url)).GET().build();
21)         HttpResponse resp=client.send(req,HttpResponse.BodyHandler.asFile(Paths.get("abc.
html")));
22)         processResponse(resp);
23)     }
24)     public static void processResponse(HttpResponse resp)
25)     {
26)         System.out.println("Status Code:"+resp.statusCode());
27)         //System.out.println("Response Body:"+resp.body());
28)         HttpHeaders header=resp.headers();
29)         Map<String,List<String>> map=header.map();
30)         System.out.println("Response Headers");
31)         map.forEach((k,v)->System.out.println("\t"+k+":"+v));
32)     }
33) }
```

abc.html will be created in the current working directory. Open that file to see body of response.

## Asynchronous Communication:

In Blocking Mode (Synchronous Mode), Once we send Http Request, we should wait until getting response. It creates performance problems.

But in Non-Blocking Mode (Asynchronous Mode), we are not required to wait until getting the response. We can continue our execution and later point of time we can use that HttpResponse once it is ready, so that performance of the system will be improved.



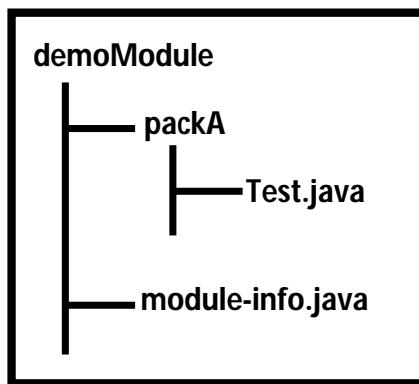
HttpClient class contains sendAsync() method to send asynchronous request.

```
CompletableFuture<HttpResponse<String>> cf =  
client.sendAsync(req,HttpResponse.BodyHandler.asString());
```

CompletableFuture Object can be used to hold HttpResponse in asynchronous communication. This class present in java.util.concurrent package. This class contains isDone() method to check whether processing completed or not.

```
public boolean isDone()
```

## Demo Program For Asynchronous Communication:



### module-info.java:

```
1) module demoModule  
2) {  
3)   requires jdk.incubator.httpclient;  
4) }
```

### Test.java:

```
1) package packA;  
2) import jdk.incubator.http.HttpClient;  
3) import jdk.incubator.http.HttpRequest;  
4) import jdk.incubator.http.HttpResponse;  
5) import jdk.incubator.http.HttpHeaders;  
6) import java.net.URI;  
7) import java.util.Map;  
8) import java.util.List;  
9) import java.util.concurrent.CompletableFuture;  
10) public class Test  
11) {  
12)   public static void main(String[] args) throws Exception  
13)   {  
14)     String url="https://www.redbus.in/info/aboutus";  
15)     sendGetAsycnRequest(url);
```



```
16) }
17) public static void sendGetAsyncRequest(String url) throws Exception
18) {
19)     HttpClient client=HttpClient.newHttpClient();
20)     HttpRequest req=HttpRequest.newBuilder(new URI(url)).GET().build();
21)     System.out.println("Sending Asynchronous Request...");
22)     CompletableFuture<HttpResponse<String>> cf = client.sendAsync(req,HttpResponse.BodyHandler.asString());
23)     int count=0;
24)     while(!cf.isDone())
25)     {
26)         System.out.println("Processing not done and doing other activity:"+ ++count);
27)     }
28)     processResponse(cf.get());
29) }
30) public static void processResponse(HttpResponse resp)
31) {
32)     System.out.println("Status Code:"+resp.statusCode());
33)     //System.out.println("Response Body:"+resp.body());
34)     HttpHeaders header=resp.headers();
35)     Map<String,List<String>> map=header.map();
36)     System.out.println("Response Headers");
37)     map.forEach((k,v)->System.out.println("\t"+k+":"+v));
38) }
39) }
```



# JAVA 10 FEATURES



# Java 10 Updations

- 1) Local-Variable Type Inference [286]
- 2) Consolidate the JDK Forest into a Single Repository [296]
- 3) Garbage-Collector Interface [304]
- 4) Parallel Full GC for G1 [307]
- 5) Application Class-Data Sharing [310]
- 6) Thread-Local Handshakes [312]
- 7) Remove the Native-Header Generation Tool (javah) [313]
- 8) Additional Unicode Language-Tag Extensions [314]
- 9) Heap Allocation on Alternative Memory Devices [316]
- 10) Experimental Java-Based JIT Compiler [317]
- 11) Root Certificates [319]
- 12) Time-Based Release Versioning [322]

### **1) Local-Variable Type Inference:**

Upto JAVA9 version, if we want to declare and initialize a variable then we have to use the following syntax.

```
Data_Type varName = Value;
```

#### **EX:**

```
int i = 10;  
String str = "abc";  
ArrayList<String> al = new ArrayList<String>();  
---  
---
```

Java10 version has given an option to declare variables without specifying its type and calculating variable type on the basis of its value , this feature is called as "Type Inference".

In Java10 version, if we want to declare variables with type inference we must use "var" just before variable.

#### **Syntax:**

```
var varName = Value;
```



EX:

```
1) package java10features;
2) public class Test {
3)
4)     public static void main(String[] args) {
5)         //Upto Java9 version
6)         int i = 10;
7)         String str = "Durga";
8)         System.out.println(i);
9)         System.out.println(str);
10)
11)        // Java 10 version
12)        var j = 10;
13)        System.out.println(j);
14)        var string = "Durga";
15)        System.out.println(string);
16)    }
17) }
```

OUTPUT

```
10
Durga
10
Durga
```

We can use Type Inference to the variables inside the loops also.

EX:

```
1) package java10features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)         //Up to Java9 version
7)         for(int i = 0; i < 10; i++) {
8)             System.out.println(i);
9)         }
10)
11)        // Java10 version
12)        System.out.println();
13)        for(var i = 0; i < 10; i++) {
14)            System.out.println(i);
15)        }
```



```
16) }
17) }
```

## OUTPUT

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

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

If we want to declare arrays with Type Inference then we must use the following syntax.  
`var varName = new DataTpe[]{values};`

## EX:

```
1) package java10features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)         // Up to Java9 version
7)         int[] a = {1,2,3,4,5};
8)         for(int i = 0; i < a.length; i++) {
9)             System.out.println(a[i]);
10)        }
11)        System.out.println();
12)        String[] str = {"AAA", "BBB", "CCC", "DDD", "EEE"};
13)        for(String s: str) {
14)            System.out.println(s);
15)        }
```



```
16) System.out.println();
17)
18) //Java10 version
19) //var x = {1,2,3,4,5}; --> Error
20) //var string = {"AAA","BBB","CCC","DDD","EEE"}; --> Error
21) var x = new int[] {1,2,3,4,5};
22) for(var i = 0; i < x.length; i++) {
23)     System.out.println(x[i]);
24) }
25) System.out.println();
26) var string = new String[] {"AAA", "BBB", "CCC", "DDD", "EEE"};
27) for(var element: string ) {
28)     System.out.println(element);
29) }
30}
31}
```

## OUTPUT

1  
2  
3  
4  
5

AAA  
BBB  
CCC  
DDD  
EEE

1  
2  
3  
4  
5

AAA  
BBB  
CCC  
DDD  
EEE



We can use Type Inference for Collection variables declarations.

EX:

```
1) package java10features;
2)
3) import java.util.ArrayList;
4) import java.util.List;
5)
6) public class Test {
7)
8)     public static void main(String[] args) {
9)         // Up to Java9 version
10)        List<String> list1 = new ArrayList<String>();
11)        list1.add("AAA");
12)        list1.add("BBB");
13)        list1.add("CCC");
14)        list1.add("DDD");
15)        System.out.println(list1);
16)
17)        // Java 10 version
18)        var list2 = new ArrayList<String>();
19)        list2.add("AAA");
20)        list2.add("BBB");
21)        list2.add("CCC");
22)        list2.add("DDD");
23)        System.out.println(list2);
24)    }
25} }
```

OUTPUT

```
[AAA, BBB, CCC, DDD]
[AAA, BBB, CCC, DDD]
```

Note: In Collection variables declaration we can use diamond operator along with Type Inference, in this context, Compiler will assign Object type to the respective variable , we can add any type of element to the respective Collection.

EX:

```
1) package java10features;
2)
3) import java.util.ArrayList;
4) import java.util.List;
5)
6) public class Test {
```



```
7)
8) public static void main(String[] args) {
9)     var list = new ArrayList<>();
10)    list.add(new Integer(10));
11)    list.add("ABC");
12)    list.add(new StringBuffer("XYZ"));
13)    System.out.println(list);
14)
15} }
```

## Limitations of Type Inference:

1. Type Inference is applicable for only local variables only, not applicable for class level variables.

EX:

```
1) class A{
2)     int i = 10; --> Valid
3)     var str = "Durga"; ---> Invalid
4)     void m1() {
5)         var j = 20;-----> Valid
6)         System.out.println(j);
7)     }
8) }
```

2. Type Inference is not possible for Parameters in methods, constructors,....

EX:

```
1) class A{
2)     A(var i){ ---> Error
3)     }
4)     void m1(var i) { -----> Error
5)     }
6) }
```



3. Type inference is not possible for variables declaration with out initialization and with null value initialization.

EX:

```
1) class A{  
2)     void m1() {  
3)         var i; -----> Error  
4)         i = 10;  
5)  
6)         var date = null; -----> Error  
7)     }  
8) }
```

4. Type inference is not possible for Lambda Expression variable declaration, but, we can apply Type Inference for variables inside Lambda expressions.

EX:

```
1) package java10features;  
2)  
3) import java.util.function.Predicate;  
4)  
5) public class Test {  
6)     public static void main(String[] args) {  
7)         // Up to Java9 version  
8)         Predicate<Integer> p = n -> n < 10;  
9)         System.out.println(p.test(5));  
10)        System.out.println(p.test(15));  
11)        // Java 10 version  
12)        var p1 = n -> n < 10; -----> Error  
13)    }  
14) }
```

EX:

```
1) package java10features;  
2)  
3) import java.util.function.Function;  
4) import java.util.function.Predicate;  
5)  
6) public class Test {  
7)     public static void main(String[] args) {  
8)         Function<Integer, Integer> f = n -> {  
9)             var result = n * n;  
10)            return result;
```



```
11)    };
12)    System.out.println(f.apply(10));
13)    System.out.println(f.apply(20));
14) }
15} }
```

## OUTPUT

100  
400

**Note:** In Java 10 version, we can use 'var' as variable name , it will not give any error.

## EX:

```
1) public class Test {
2)   public static void main(String[] args) {
3)     var var = 10;
4)     System.out.println(var);
5)   }
6) }
```

## OUTPUT

10

## API Changes in List, Set and Map:

In Java 10 version, copyOf() function is added in List, Set and Map inorder to get Unmodifiable List, Set and Map.

### EX1:

```
1) package java10features;
2)
3) import java.util.ArrayList;
4) import java.util.List;
5)
6) public class Test {
7)   public static void main(String[] args) {
8)     List<Integer> list = new ArrayList<Integer>();
9)     list.add(10);
10)    list.add(20);
11)    list.add(30);
12)    list.add(40);
13)    System.out.println(list);
14)    var newList = List.copyOf(list);
15)    System.out.println(newList);
```



```
16) newList.add(60);---> java.lang.UnsupportedOperationException  
17) }  
18) }
```

EX:

```
1) package java10features;  
2) import java.util.HashSet;  
3) import java.util.Set;  
4) public class Test {  
5)     public static void main(String[] args) {  
6)         Set<Integer> set = new HashSet<Integer>();  
7)         set.add(10);  
8)         set.add(20);  
9)         set.add(30);  
10)        set.add(40);  
11)        System.out.println(set);  
12)        var newSet = Set.copyOf(set);  
13)        System.out.println(newSet);  
14)        newSet.add(60); ---> java.lang.UnsupportedOperationException  
15)    }  
16) }
```

## API Changes in Collectors Class in Stream API:

Upto JAVA9 version, Collectors class has `toList()`, `toSet()`, `toMap()` methods to get list or set or Map of elements from Streams.

JAVA10 has provided the following methods to Collectors class to provide Unmodifiable or immutable List, Set and Map.

- 1) `public static List unmodifiableList()`
- 2) `public static Set unmodifiableSet()`
- 3) `public static Map unmodifiableMap()`

EX:

```
1) package java10features;  
2)  
3) import java.util.ArrayList;  
4) import java.util.List;  
5) import java.util.stream.Collectors;  
6) import java.util.stream.Stream;  
7)  
8) public class Test {  
9)     public static void main(String[] args) {  
10)        List<Integer> list = new ArrayList<Integer>();
```



```
11)    list.add(5);
12)    list.add(10);
13)    list.add(15);
14)    list.add(20);
15)    list.add(25);
16)    list.add(30);
17)    System.out.println(list);
18)    Stream<Integer> stream = list.stream();
19)    List<Integer> list1 = stream.filter(n-> n%2==0).
      collect(Collectors.toUnmodifiableList());
20)    System.out.println(list1);
21)    list1.add(40); --> java.lang.UnsupportedOperationException
22)  }
23} }
```

EX:

```
1) package java10features;
2)
3) import java.util.Set;
4) import java.util.stream.Collectors;
5) import java.util.stream.Stream;
6)
7) public class Test {
8)   public static void main(String[] args) {
9)     Set<Integer> set = Set.of(5,10,15,20,25,30);
10)    Stream<Integer> stream = set.stream();
11)    Set<Integer> s = stream.filter(n->n%2!=0).
      collect(Collectors.toUnmodifiableSet());
12)    System.out.println(s);
13)    s.add(35); --> java.lang.UnsupportedOperationException
14)  }
15} }
```

## 2) Consolidate the JDK Forest into a Single Repository:

Upto Java 9 version, the complete java has devided into repositories like root corba, hotspot, jdk, langtools,jaxp, jaxw, nashorn but, in JAVA10 version all these repositories are provided into single Repository under src inorder to simplify and streamline Development.



### **3) Garbage-Collector Interface**

Upto Java9 version, Different Garbage Collectors are existed in Java like G1, CMS,... which are available in different modules in java, they are not providing cleaner implementation, but, JAVA10 version has introduced GC interface for cleaner implementation of Garbage Collectors.

### **4) Parallel Full GC for G1**

In Java9 version, G1 Garbage is Default Garbage Collector, it will provide effective performance on Single Threading, when it comes for Concurrent collection its performance is down, to overcome this problem, JAVA 10 has provided Multi Threaded model with G1 Garbage Collector in the form of Parallel Full GC.

### **5) Application Class-Data Sharing:**

The main intention of Class-Data Sharing is to share loaded class to multiple JVMs through shared memory when we are working large scale applications, Class-Data Sharing was introduced in JSD1.5 version and it was limited to System classes and Serial GC, but, JAVA10 version, it was extended to allow application classes along with System classes and Garbage Collectors.

### **6) Thread-Local Handshakes:**

- When we Lock an object by a thread explicitly or when we apply synchronization on objects then other threads will come to blocked state safely whose state is described by other threads in JVM, here threads which are in safepoint. In Some Situation JVM will keep all the Threads in Blocked state called as "Stop The Word", before Stop the Word, all threads will come to safepoint, this safepoint is called as Global Safepoint.
- A handshake operation is a callback that is executed for each Java Thread while that thread is in a safepoint state. The callback is executed either by the thread itself or by the VM thread while keeping the thread in a blocked state.
- Thread-Local Handshakes is JVM internal feature to improve performance, This feature provides a way to execute a callback on threads without performing a global VM safepoint.

### **7) Remove the Native-Header Generation Tool (javah):**

- javah is a C-Header file generator from Java classes, it will be used in Java Native Interfaces which accessing native Methods.
- JAVA10 version has removed javah from Java software, because, javah capabilities are already embedded with javac command in JAVA8 version.



### 8) Additional Unicode Language-Tag Extensions

Up to Java9 version, `java.lang.Locale` has the UNICODE support for "ca" [Calender] and "nu" [Numeric], but, JAVA10 version has given UNICODE support for "cu" [Currency], "fw" [First Day Of Week], "rg" [Region Override], "tz" [Time Zone].

### 9) Heap Allocation on Alternative Memory Devices

- This featur allows the HotSpot VM to allocate the Java object heap on an alternative memory device, specified by the user.
- It is very much required in Multi JVM environment, in BIG Data applications, we need to execute some low priority threads like Daemon threads, Services threads and some other High priority threads,
- This new feature would make it possible in a multi-JVM environment to assign lower priority processes to use the NV-DIMM memory, and instead only allocate the higher priority processes to the DRAM.
- NV-DIMM → Non Volatile Dual In Memory Modules, it is low cost and having larger capacity.
- DRAM → Dynamic Random Access Memory, High Cost and More Potential Memory.

### 10) Experimental Java-Based JIT Compiler:

- In JVM, JIT Compiler will increase perofrmance of JVM while converting Byte code to Native, but, it was written in C++ .
- JAVA10 has provided an experimental JIT Compiler written in Java completely and it was provided in support of Linux/64 bit OS.

### 11) Root Certificates

It is Security feature in Java, It is Providing root CA [Certificate Authority] certificates makes "OpenJDK builds more attractive to developers" and "reduces [sic] the differences between those builds and Oracle JDK builds".

### 12) Time-Based Release Versioning

- Starting from Java 10, Oracle has adapted time based version-string scheme [JEP 322]. The new time-based model has replaced the feature-based, multi-year release model of the past. Unlike the old releases, the new time-based releases will not be delayed and features will be released every six months, with no constraints on what features can go out in the releases.
- The updates releases will occur every quarter (Jan, April, July, Oct). Update releases will be strictly limited to fixes of security issues, regressions, and bugs in



---

newer features. Going by schedule planning, we can say that each feature release will receive two updates before the next feature release.

- The new version format of Java is **\$FEATURE.\$INTERIM.\$UPDATE.\$PATCH**
- **\$FEATURE** : It will be incremented every 6 months and based on feature release versions e.g: JDK 10, JDK 11.
- **\$INTERIM** : It will be incremented for non-feature releases that contain compatible bug fixes and enhancements.
- **\$UPDATE** : It will be incremented for compatible update releases that fix security issues, regressions, and bugs in newer features.
- **\$PATCH** : It will be incremented only when it's necessary to produce an emergency release to fix a critical issue.



# JAVA 11 FEATURES



# Java 11 Version Updations

- 1) Running Java File with single command
- 2) New utility methods in String class
- 3) Local-Variable Syntax for Lambda Parameters
- 4) Nested Based Access Control
- 5) HTTP Client
- 6) Reading/Writing Strings to and from the Files
- 7) Flight Recorder

### **1) Running Java File with Single Command:**

- Upto Java 10 version, if we want to run Java file, first we have to compile Java file with "javac" command then we have to execute Java application by using "java" command.
- Java11 version has provided an environment to execute Java file with out compiling java file explicitly, when we execute Java file internally Java file will be compiled , it will

EX: D:\java11practice\Test.java

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         System.out.println("Welcome To Java 11 Version");
6)     }
7) }
```

On Command Prompt:

```
D:\java11practice>java Test.java
Welcome To Java 11 Version
```

If we provide more than one class with main() method in single java file and if we execute that Java file then JVM will search for the main() method class in the order in which we have provided from starting point of the file to ending point, in which class JVM identifies main() method first time then JVM will treat that class as the actual main class and JVM will execute that main class.



EX: abc.java

```
1) class A{  
2)   public static void main(String[] args){  
3)     System.out.println("A-main()");  
4)   }  
5) }  
6) class B{  
7)   public static void main(String[] args){  
8)     System.out.println("B-main()");  
9)   }  
10) }  
11) class C{  
12)   public static void main(String[] args){  
13)     System.out.println("C-main()");  
14)   }  
15) }
```

D:\java7\java11Features>java abc.java  
OUTPUT A-main()

EX: abc.java

```
1) class B {  
2)   public static void main(String[] args){  
3)     System.out.println("B-main()");  
4)   }  
5) }  
6) class C{  
7)   public static void main(String[] args){  
8)     System.out.println("C-main()");  
9)   }  
10) }  
11) class A{  
12)   public static void main(String[] args){  
13)     System.out.println("A-main()");  
14)   }  
15) }
```

D:\java7\java11Features>java abc.java  
OUTPUT B-main()



## abc.java

```
1) class C{
2)   public static void main(String[] args){
3)     System.out.println("C-main()");
4)   }
5) }
6) class A{
7)   public static void main(String[] args){
8)     System.out.println("A-main()");
9)   }
10}
11 class B{
12)   public static void main(String[] args){
13)     System.out.println("B-main()");
14)   }
15}
```

D:\java7\java11Features>java abc.java  
OUTPUT C-main()

**Note:** Java11 Version is providing backward Compatibility, that is, in Java11 version, we can follow the old convention like compilation of Java file with javac and execution of the Java program with java command.

EX:

```
1) public class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     System.out.println("Welcome To Java11 Programming");
6)   }
7) }
```

D:\java7\java11Features>javac Test.java  
D:\java7\java11Features>java Test  
OUTPUT Welcome To Java11 Programming



## 2) New Utility Methods in String Class

In Java11 version, String class has provided the following new methods.

- 1) public String repeat(int count)
- 2) public boolean isBlank()
- 3) public String strip()
- 4) public String stripLeading()
- 5) public String stripTrailing()
- 6) public Stream lines()

### 1) public String repeat(int count)

- It will repeat the String object content upto the given no of times.
- If we provide -ve value then repeat() method will raise an Exception like "java.lang.IllegalArgumentException".
- If the count value is 0 then it will return an empty string.

EX:

```
String str = "Durga";
String str1 = str.repeat(5);
System.out.println(str1);
OUTPUT DurgaDurgaDurgaDurgaDurga
```

EX:

```
String str = "Durga";
String str1 = str.repeat(0);
System.out.println(str1);
OUTPUT Empty String, No Output
```

EX:

```
String str = "Durga";
String str1 = str.repeat(-1);
System.out.println(str1);
OUTPUT java.lang.IllegalArgumentException
```

### 1) public boolean isBlank()

It will check whether the String is Blank[Or White Spaces] or not, if the String is blank then it will return true value otherwise false value.

EX:

```
String str = "";
System.out.println(str.isBlank());
OUTPUT true
```



**EX:**

```
String str = " ";
System.out.println(str.isBlank());
OUTPUT true
```

**EX:**

```
String str = "abc";
System.out.println(str.isBlank());
OUTPUT false
```

## **2) public String strip()**

It is same as trim() method, it will remove all leading spaces and trailing spaces of a String.

**EX:**

```
String str = new String(" Durga Software Solutions ");
String str1 = str.strip();
System.out.println(str1);
OUTPUT Durga Software Solutions
```

## **3) public String stripLeading()**

It will remove all leading spaces[Pre Spaces] to a String, it will not remove Trailing spaces.

**EX:**

```
String str = new String(" Durga Software Solutions ");
String str1 = str.stripLeading();
System.out.println(str1);
OUTPUT Durga Software Solutions [Here Trailing spaces are existed as it is].
```

## **4) public String stripTrailing()**

It will remove all trailing Spaces to a String, it will not remove leading spaces.

**EX:**

```
String str = new String(" Durga Software Solutions ");
String str1 = str.stripTrailing();
System.out.println(str1);
```



## 5) public Stream lines()

This method will split a String into no of Tokens in the form of Stream object on the basis of '\n' literals.

EX:

```
1) import java.util.stream.*;
2) import java.util.*;
3) class Test
4) {
5)     public static void main(String[] args) throws Exception
6)     {
7)         String str = new String("Durga\nSoftware\nSolutions");
8)         Stream<String> s = str.lines();
9)         List<String> l = s.collect(Collectors.toList());
10)        System.out.println(l);
11)    }
12) }
```

## 3) Local-Variable Syntax for Lambda Parameters:

In Java10 version, Local variables type inference was introduced, with this, we can declare variables without providing data type explicitly and just by using "var", here type will be calculated on the basis of the provided variable values.

EX:

```
var str = "Durga";
var i = 10;
```

In Java10 version, "var" is not allowed for the parameter variables.

EX:

```
1) interface I
2) {
3)     int m1(int i, int j);
4) }
5) class Test
6) {
7)     public static void main(String[] args)
8)     {
9)         I l = (var i,var j) -> i+j;
10)        System.out.println(l.m1(10,20));
11)    }
12) }
```



## On Command Prompt:

```
D:\java10>javac Test.java
Test.java:9: error: 'var' is not allowed here
    l1 = (var i,var j) -> i+j;
           ^
Test.java:9: error: 'var' is not allowed here
    l1 = (var i,var j) -> i+j;
```

JAVA 11 version has provided flexibility to use "var" for the parameter variables in Lambda Expressions.

EX:

```
1) package java11features;
2)
3) interface Calculator{
4)     int add(int i, int j);
5) }
6) public class Test {
7)     public static void main(String[] args) {
8)         Calculator cal = (var i, var j) -> i+j;
9)         System.out.println(cal.add(10, 20));
10)        System.out.println(cal.add(30, 40));
11)
12)    }
13) }
```

## OUTPUT

```
30
70
```

## Limitations:

1. We must provide var for all Lambda parameters, not for few lambda parameters.

EX:

```
interface Calculator{
    int add(int i, int j);
}
Calculator cal = (var i, j) -> i+j → Invalid
Calculator cal = (var i, int j) -> i+j; → Invalid
```



2. If we have simple Parameter in Lambda and if we want to use "var" then () are mandatory.

EX:

```
1) package java11features;
2)
3) interface Calculator{
4)     int sqr(int i);
5) }
6) public class Test {
7)     public static void main(String[] args) {
8)         //Calculator cal = var i -> i*i; --> invalid
9)         Calculator cal = (var i) -> i*i;
10)        System.out.println(cal.sqr(10));
11)        System.out.println(cal.sqr(30));
12)
13)    }
14) }
```

## OUTPUT

100  
900

## 4) Nested Based Access Control:

Upto Java10 version, in inner classes, if we access private member of any inner class in the respective outer class by using reflection API then we are able to get an exception like `java.lang.IllegalAccessException`, but, if we use Java11 version then we will get any exception , because, in JAVA11 version new methods are introduced in `java.lang.Class` class like

- 1) `public Class getNestHost()` : It able to get the enclosed outer class `java.lang.Class` object.
- 2) `public Class[] getNestMembers()`: It will return all nested members in the form of `Class[]`.
- 3) `public boolean isNestmateOf()`: It will check whether a member is nested member or not.

Note: In Java11 the above methods will be executed internally when we access private members of the outer class from nested class and nested members from outer class.



EX:

```
1) package java11features;
2)
3) public class Test {
4)     class NestTest {
5)         private int count = 10;
6)     }
7)
8)     public static void main(String[] args) throws Exception {
9)         Test.NestTest tt = new Test().new NestTest();
10)        System.out.println(tt.count);
11)
12)        java.lang.reflect.Field f = NestTest.class.getDeclaredField("count");
13)        f.setInt(tt, 2000);
14)        System.out.println(tt.count);
15)    }
16} }
```

If we run the above program then we will get `java.lang.IllegalAccessException`.

If we run the above program we will get out put

```
10
2000
```

```
D:\java11practice>set path=C:\Java\jdk-10.0.2\bin;
```

```
D:\java11practice>javac Test.java
```

```
D:\java11practice>java Test
```

```
10
```

```
Exception in thread "main" java.lang.IllegalAccessException: class Test cannot access a
member of class Test$NestTest with modifiers "private"
```

```
D:\java11practice>set path=C:\Java\jdk11\jdk-11\bin;
```

```
D:\java11practice>javac Test.java
```

```
D:\java11practice>java Test
```

```
10
```

```
2000
```



## **HTTP Client:**

Http Client is an API, it can be used to send requests to server side appl or remote appl and to recieve response from Server side appl or Remote applications.

Initially, Http Client API was introduced in JAVA9 version, where it was introduced in a module incubator and it was not fully implemented and it was not in streamlined to use that in Java applications.

JAVA11 version has provided stream lined and standardised implementation for Http Client API in the form of a seperate package "java.net.http".

`java.net.http` contains mainly 3 libraries.

- 1) `HttpClient`
- 2) `HttpRequest`
- 3) `HttpResponse`

### **1) HttpClient:**

--> It repesents a client for Http protocol, it is responsible to send requests and to recieve response from server.

EX: `HttpClient client = HttpClient.newHttpClient();`

### **2) HttpRequest:**

--> It able to manage request details like url of the Request, type of request, timeout configurations.....

```
HttpRequest request = HttpRequest.newBuilder()
    .GET()
    .uri(URI.create("http://www.durgasoft.com"))
    .build();
```

### **3) HttpResponse**

--> With the above configurations , if we submit request to Server side appl , Server will generate some response to client , where at client we have to represent response, for representing response we have to use "HttpResponse".

--> `HttpResponse` is able to provide response details like status code, Response Headers, Response Body

---> `HttpResponse` contains

- a)`statusCode()`: It will return status code of the present response.
- b)`headers()`: It will return headers
- c)`body()`: It will return the actual response

EX: `HttpResponse response = client.send(request);`

```
System.out.println(response.statusCode());
System.out.println(response.headers());
```



```
System.out.println(response.body());
```

EX:

```
1) package java11features;
2)
3) import java.net.URI;
4) import java.net.http.HttpClient;
5) import java.net.http.HttpRequest;
6)
7) import java.net.http.HttpResponse;
8) import java.net.http.HttpResponse.BodyHandlers;
9)
10) public class Test {
11)
12)     public static void main(String[] args) throws Exception {
13)         var client = HttpClient.newHttpClient();
14)         var request = HttpRequest.newBuilder()
15)             .GET()
16)             .uri(URI.create("https://www.google.com"))
17)             .build();
18)
19)         HttpResponse<String> response = client.send(request, BodyHandlers.ofString());
20)
21)         System.out.println("Status Code : "+response.statusCode());
22)         System.out.println();
23)         System.out.println("Response Headers : "+response.headers());
24)         System.out.println();
25)         System.out.println("Response Body : "+response.body());
26)     }
27} }
```

If we want to send Post request with form data then we have to use POST() method instead of GET() and provide request parameters data along with URL like below

<http://localhost:1010/loginapp/login?uname=durga&upwd=abc>

EX: Client.java

```
1) package com.durgasoftware.java11features;
2)
3) import java.io.BufferedReader;
4) import java.io.Console;
5) import java.io.InputStreamReader;
6) import java.net.URI;
7) import java.net.http.HttpClient;
8) import java.net.http.HttpRequest;
```



```
9) import java.net.http.HttpResponse;
10)
11) public class Test {
12)     public static void main(String[] args) throws Exception {
13)         BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
14)
15)         System.out.print("User Name : ");
16)         String uname = br.readLine();
17)         System.out.print("User Password : ");
18)         String upwd = br.readLine();
19)
20)         var client = HttpClient.newHttpClient();
21)         var request = HttpRequest.newBuilder().POST(HttpRequest.BodyPublishers.ofString("")).uri(URI.create("http://localhost:1010/loginapp/login?uname=" + uname + "&upwd=" + upwd)).build();
22)         var response = client.send((HttpRequest) request, HttpResponse.BodyHandlers.ofString());
23)         //System.out.println("Status Code : " + response.statusCode());
24)         //System.out.println();
25)         //System.out.println("Response Headers : " + response.headers());
26)         //System.out.println();
27)         System.out.println("Login Status : " + response.body());
28> }
```

## Server Side Application:

C:\tomcat\webapps  
loginapp  
|---WEB-INF  
 |---classes  
 |---LoginServlet.java  
 |---LoginServlet.class

### LoginServlet.java

```
1) import java.io.*;
2) import javax.servlet.*;
3) import javax.servlet.http.*;
4) import javax.servlet.annotation.*;
5) @WebServlet("/login")
6) public class LoginServlet extends HttpServlet
7) {
8)     public void doPost(HttpServletRequest request, HttpServletResponse response)
throws ServletException, IOException
```



```
9) {
10)     response.setContentType("text/html");
11)     PrintWriter out = response.getWriter();
12)     String uname = request.getParameter("uname");
13)     String upwd = request.getParameter("upwd");
14)     System.out.println("uname :" +uname);
15)     System.out.println("upwd :" +upwd);
16)     String status = "";
17)     if(uname.equals("durga") && upwd.equals("durga"))
18)     {
19)         status = "User Login Success";
20)     }
21)     else
22)     {
23)         status = "User Login Failure";
24)     }
25)     out.println("<html><body>");
26)     out.println("<h1>" +status+ "</h1>");
27)     out.println("</body></html>");
28) }
29}
```

## Reading/Writing Strings to and from the Files:

Java11 version has provided the following four new methods in java.nio.file.Files class to read data from file and to write data to file.

1. **public static Path writeString(Path path, CharSequence csq, OpenOption... options)**  
throws IOException
2. **public static Path writeString(Path path, CharSequence csq, Charset cs, OpenOption... options)**  
throws IOException
3. **public static String readString(Path path)** throws IOException
3. **public static String readString(Path path, Charset cs)** throws IOException

EX:

```
1) package java11features;
2) import java.nio.file.Path;
3) import java.nio.file.Paths;
4) import java.nio.file.StandardOpenOption;
5) import java.nio.file.Files;
6)
7) public class Test {
```



```
8) public static void main(String[] args) throws Exception {  
9)     Path filePath = Paths.get("E:/", "abc/xyz", "welcome.txt");  
10)    Files.writeString(filePath, "Welcome to Durga Software Solutions");  
11)    String content = Files.readString(filePath);  
12)    System.out.println(content);  
13) }  
14}
```

If we want to perform append operation in the file then we have to use StandardOpenOption.APPEND constant like below.

```
Files.writeString(filePath, "Welcome to Durga Software Solutions",  
StandardOpenOption.APPEND);
```

## Flight Recorder:

Java Flight Recorder is a profiling tool used to gather diagnostics and profiling data from a running Java application, it is available for only for Paying Users, not for normal Users.



# JAVA 12 FEATURES



# Java 12 Features

- 1) Switch Expressions
- 2) File mismatch() Method
- 3) Compact Number Formatting
- 4) Teeing Collectors in Stream API
- 5) Java Strings New Methods – indent(), transform(), describeConstable(), and resolveConstantDesc().
- 6) JVM Constants API
- 7) Pattern Matching for instanceof
- 8) Raw String Literals is Removed From JDK 12.

**Note:**

IN Java12 version, some features are preview features, we must compile and execute that preview features by enable preview features. To enable preview features for compilation and execution we have to use the following commands.

```
javac --enable-preview -source 12 Test.java
```

```
java --enable-preview Test
```

### **1) Switch Expressions:**

It is preview feature only, it will be included in the future versions.

Upto Java 11 version, we are able to write switch as statement like below.

```
switch(var)
{
    case val1:
        ----instructions----
        break;
    case val2:
        ----instructions----
        break;
    ----
    ----
    case n:
        ----instructions----
        break;
    default:
        ----instructions----
        break;
}
```



From Java12 version onwards, we are able to use switch in the following two ways.

- 1) Switch Statement
- 2) Switch Expression.

## 1) Switch Statement:

It is almost all same as switch statement before Java12 version, but, in JAVA12 version, we are able to define multi labelled case statements in switch.

### Syntax:

```
switch(val) {  
    case label1, label2,...label_n:  
        -----  
        break;  
        -----  
        -----  
    default:  
        -----  
        break;  
}
```

### EX:

```
1) class Test  
2) {  
3)     public static void main(String[] args)  
4)     {  
5)         var i = 10;  
6)         switch(i){  
7)             case 5,10:  
8)                 System.out.println("Five or Ten");  
9)                 break;  
10)  
11)             case 15, 20:  
12)                 System.out.println("Fifteen or Twenty");  
13)                 break;  
14)  
15)             default:  
16)                 System.out.println("Defasult");  
17)                 break;  
18)         }  
19)     }  
20} }
```

### OUTPUT

Five or Ten



## 2) Switch Expression:

In JAVA12 version, we are able to use switch as an expression, it return a value and it is possible to assign that value to any variable.

In switch expression, switch is able to return values in two ways.

- 1) By Using break statement.
- 2) By using Lambda style syntax.

## 1) By Using break Statement:

### Syntax:

```
var varName = switch(value){  
    case val1:  
        break value;  
    case val2:  
        break value;  
    ----  
    ----  
    case val_n:  
        break value;  
    default:  
        break value;  
}; // ; is mandatory
```

### EX:

```
1) class Test {  
2)     public static void main(String[] args) {  
3)         var i = 10;  
4)         var result = switch(i){  
5)             case 5:  
6)                 break "Five";  
7)             case 10:  
8)                 break "Ten";  
9)             case 15:  
10)                break "Fifteen";  
11)                case 20:  
12)                    break "Twenty";  
13)                default:  
14)                    break "Number not in 5, 10, 15 and 20";  
15)            };  
16)            System.out.println(result);  
17)        }  
18} }
```



## OUTPUT

Ten

**Note:** In switch expression, we are able to provide multi labelled cases.

## EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         var i = 10;
6)         var result = switch(i){
7)             case 5, 10:
8)                 break "Five or Ten";
9)             case 15,20:
10)                break "Fifteen or Twenty";
11)
12)             default:
13)                 break "Number not in 5, 10, 15 and 20";
14)         };
15)         System.out.println(result);
16)     }
17) }
```

## OUTPUT

Five or Ten

## 2) By using Lambda Style Syntax:

### Syntax:

```
DataType result = switch(value){
    case val1 -> Expression;
    ----
    default -> Expression;
};
```

## EX:

```
1) class Test
2) {
3)     public static void main(String[] args)
4)     {
5)         var i = 10;
6)         var result = switch(i){
```



```
7)      case 5 -> "Five";
8)      case 10 -> "Ten";
9)      case 15 -> "Fifteen";
10)     case 20 -> "Twenty";
11)     default -> "Number not in 5, 10, 15 and 20";
12)   };
13)   System.out.println(result);
14) }
15} }
```

## OUTPUT

Ten

**Note:** In switch, Lambda style syntax, we are able to provide multi labelled cases.

### Syntax:

```
DataType result = switch(value){
    case val1,val2,...val_n -> Expression;
    ----
    default -> Expression;
};
```

## EX:

```
1) class Test
2) {
3)   public static void main(String[] args)
4)   {
5)     var i = 10;
6)     var result = switch(i){
7)       case 5, 10 -> "Five or Ten";
8)       case 15,20 -> "Fifteen or Twenty";
9)       default -> "Number not in 5, 10, 15 and 20";
10)    };
11)    System.out.println(result);
12)  }
13) }
```

## OUTPUT

Five or Ten



EX:

```
1) import java.io.*;
2) class Test
3) {
4)     public static void main(String[] args)
5)     {
6)         var day = "";
7)         var dayType = "";
8)         try{
9)             BufferedReader br = new BufferedReader(new InputStreamReader
(System.in));
10)            System.out.print("Enter Day [Upper case letters only] : ");
11)            day = br.readLine();
12)            dayType = switch(day){
13)                case "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY" ->
"EEK DAY";
14)                case "SATURDAY", "SUNDAY" -> "EEKEND";
15)                default -> day+" is an invalid Day";
16)            };
17)        }catch(Exception e){
18)            System.out.println(e.getMessage());
19)        }
20)        System.out.println(day+" is "+dayType);
21)    }
22} }
```

To compile and execute the above code we must enable preview feature along with javac and java

D:\java7\java12Features>javac --enable-preview -source 12 Test.java

Note: Test.java uses preview language features.

Note: Recompile with -Xlint:preview for details.

D:\java7\java12Features>java --enable-preview Test

Enter Day [Upper case letters only] : MONDAY

MONDAY is WEEK DAY

D:\java7\java12Features>java --enable-preview Test

Enter Day [Upper case letters only] : SUNDAY

SUNDAY is WEEKEND

D:\java7\java12Features>java --enable-preview Test

Enter Day [Upper case letters only] : MNDAY

MNDAY is an invalid Day

MNDAY is



## 2) Files mismatch() Method:

JAVA12 version has provided mismatch() method in java.nio.file.Files class, it can be used to check whether two files content is matched or not, if files content is matched then mismatch() method will return -1L value , if the files content is mismatched then mismatch() method will return the position of the mismatched byte.

EX:

```
1) package java12features;
2)
3) import java.nio.file.Files;
4) import java.nio.file.Path;
5) import java.nio.file.Paths;
6)
7) public class Test {
8)
9)     public static void main(String[] args) throws Exception {
10)
11)         Path file1 = Paths.get("E:/", "abc/xyz", "welcome1.txt");
12)         Path file2 = Paths.get("E:/", "abc/xyz", "welcome2.txt");
13)         Files.writeString(file1, "Welcome To Durga Software Solutions");
14)         Files.writeString(file2, "Welcome To Durga Software Solutions");
15)         long val = Files.mismatch(file1, file2);
16)         System.out.println(val);
17)         if(val == -1L) {
18)             System.out.println("Files Content is Matched ");
19)         }else {
20)             System.out.println("Files Content is Mismatched `");
21)         }
22)     }
23} }
```

OUTPUT

-1

Files Content is Matched

EX:

```
1) package java12features;
2)
3) import java.nio.file.Files;
4) import java.nio.file.Path;
5) import java.nio.file.Paths;
6)
7) public class Test {
```



```
8)
9) public static void main(String[] args) throws Exception {
10)
11)     Path file1 = Paths.get("E:/", "abc/xyz", "welcome1.txt");
12)     Path file2 = Paths.get("E:/", "abc/xyz", "welcome2.txt");
13)     Files.writeString(file1, "Welcome To Durga Software Solutions");
14)     Files.writeString(file2, "Welcome To Durgasoft");
15)     long val = Files.mismatch(file1, file2);
16)     System.out.println(val);
17)     if(val == -1L) {
18)         System.out.println("Files Content is Matched ");
19)     }else {
20)         System.out.println("Files Content is Mismatched `");
21)     }
22) }
23} }
```

## OUTPUT

16

Files Content is Mismatched

## 3) Compact Number Formatting:

In General, in java applications, we are able to represent numbers in numeric form, not in Short form like 10K, 100K..... and Long form like 1 Thousand or 100 Thousand, if we want to represent numbers in short form and long form like above JAVA12 version has provided a predefined class in the form of `java.text.CompactNumberFormat`.

By Using `CompactNumberFormat` we are able to perform the following actions.

### 1. Converting Normal Numbers to Short Form and Long Form of Numbers

100 -----> 100

1000 -----> 1 Thousand or 1 k

10000 -----> 10 Thousand or 10 k

### 2. Converting Numbers from Short or long form to Numeric values.

100 -----> 100

1 k -----> 1000

10 K -----> 10000

100 -----> 100

1 thousand -----> 1000

10 thousand -----> 10000



To get CompactNumberFormat object we have to use the following Factory method from NumberFormat class.

```
CompactNumberFormat cnf = NumberFormat.getCompactNumberInstance (Locale l,  
int style)
```

Where Style may be NumberFormat.style.SHORT or NumberFormat.style.LONG

To convert a number to Compact Number we will use the following method.

```
public String format(Number num)
```

To convert compact number to number we will use the following method.

```
public Number parse(String compactNumber)
```

EX:

```
1) package com.durgasoft.java12features;  
2)  
3) import java.text.NumberFormat;  
4) import java.util.Locale;  
5)  
6) public class Test {  
7)  
8)     public static void main(String[] args)throws Exception {  
9)         // Converting value from Normal number to Compact Number in Short form  
10)        NumberFormat shortForm = NumberFormat.getCompactNumberInstance(new  
11)            Locale("en", "US"),NumberFormat.Style.SHORT);  
12)        System.out.println("1000 ----> "+shortForm.format(1000));  
13)        System.out.println("10000 ---> "+shortForm.format(10000));  
14)        System.out.println("100000 --> "+shortForm.format(100000));  
15)  
16)         // Converting value from Normal number to Compact Number in Long form  
17)        NumberFormat longForm = NumberFormat.getCompactNumberInstance(new  
18)            Locale("en", "US"),NumberFormat.Style.LONG);  
19)        System.out.println("1000 ----> "+longForm.format(1000));  
20)        System.out.println("10000 ---> "+longForm.format(10000));  
21)        System.out.println("100000 --> "+longForm.format(100000));  
22)  
23)         // Converting value from Compact number in Long Form to Normal Number  
24)        System.out.println();  
25)        NumberFormat numFormat = NumberFormat.getCompactNumberInstance  
26)            (new Locale("en", "US"),NumberFormat.Style.LONG);  
27)        System.out.println("1 thousand ----> "+numFormat.parse("1 thousand"));  
28)        System.out.println("10 thousand ---> "+numFormat.parse("10 thousand"));  
29)        System.out.println("100 thousand --> "+numFormat.parse("100 thousand"));  
30)  
31)         // Converting value from Compact number in Short Form to Normal Number
```



```
30) System.out.println();
31) NumberFormat numFmt = NumberFormat.getCompactNumberInstance
   (new Locale("en", "US"), NumberFormat.Style.SHORT);
32) System.out.println("1k ----> "+numFmt.parse("1k "));
33) System.out.println("10k ----> "+numFmt.parse("10k"));
34) System.out.println("100k --> "+numFmt.parse("100k"));
35) }
36} }
```

## OUTPUT

1000 ----> 1K

10000 ---> 10K

100000 --> 100K

1000 ----> 1 thousand

10000 ---> 10 thousand

100000 --> 100 thousand

1 thousand ----> 1000

10 thousand ---> 10000

100 thousand --> 100000

1k -----> 1

10k ---> 10

100k --> 100

## 4) Teeing Collectors in Stream API

JAVA 12 version has provided Teeing collector in Stream API, its main purpose is to take two streams and performing BiFunction then generating results.

public static Collector teeing (Collector stream1, Collector stream2, BiFunction merger);  
Where stream1 and Stream2 are two Streams and merger is able to merge the result of both Collectors and generating result.

EX:

```
1) package java12features;
2)
3) import java.util.stream.Collectors;
4) import java.util.stream.Stream;
5)
6) public class Test {
7)
8)     public static void main(String[] args)throws Exception {
9)         double mean = Stream.of(1,2,3,4,5,6,7,8,9,10).collect(
```



```
10)     Collectors.teeing(
11)         Collectors.summingDouble(x->x),
12)         Collectors.counting(),
13)         (sum,count)->sum/count));
14)     System.out.println("Mean : "+mean);
15) }
16} }
```

## OUTPUT

Mean : 5.5

## 5) Java Strings New Methods

JAVA 12 version has introduced the following new functions in String class.

- 1) indent(),
- 2) transform()

### 1) public String indent(int count)

The main intention of indent() method is to add spaces from a string or remove spaces to the string.

- 1) If count < 0 then spaces will be removed at the begining of each and every line.
- 2) If count > 0 then spaces will be added at beginning of String.
- 3) If negative count > the existed spaces then all spaces are removed.

EX:

```
1) package java12features;
2) public class Test {
3)     public static void main(String[] args) throws Exception {
4)         String str1 = "Durga\nSoftware\nSolutions";
5)         System.out.println(str1);
6)         String newString1 = str1.indent(5);
7)         System.out.println(newString1);
8)         System.out.println();
9)         String str2 = " Durga\n Software\n Solutions";
10)        System.out.println(str2);
11)        String newString2 = str2.indent(-5);
12)        System.out.println(newString2);
13)        System.out.println();
14)        String str3 = " Durga\n Software\n Solutions";
15)        System.out.println(str3);
16)        String newString3 = str3.indent(-5);
17)        System.out.println(newString3);
18)        System.out.println();
```



```
19) String str4 = " Durga\n Software\n Solutions";
20) System.out.println(str4);
21) String newString4 = str4.indent(0);
22) System.out.println(newString4);
23) }
24) }
```

## OUTPUT

Durga  
Software  
Solutions  
Durga  
Software  
Solutions

Durga  
Software  
Solutions

Durga  
Software  
Solutions

Durga  
Software  
Solutions

Durga  
Software  
Solutions

Durga  
Software  
Solutions  
Durga  
Software  
Solutions

## 2) public R transform(Function f)

It will take a Function as parameter and it will transform string into some other form and return that result.



EX:

```
1) package java12features;
2) public class Test {
3)     public static void main(String[] args) throws Exception {
4)         String str = "Durga Software Solutions";
5)         String newString = str.transform(s->s.toUpperCase());
6)         System.out.println(newString);
7)     }
8) }
```

OUTPUT

DURGA SOFTWARE SOLUTIONS

## 6) JVM Constants API:

JAVA12 version has introduced constants API, it includes two interfaces like `java.lang.constant.Constable` and `java.lang.constant.ConstableDesc`.

In JAVA12 versions, the classes like `String`, `Integer`, `Byte`, `Float`,.... are implementing these two interfaces and they are providing the following two methods.

- 1) `public Optional<Constable> describeConstable()`
- 2) `public String resolveConstable()`

Both the methods are representing their objects themselves.

EX:

```
1) package com.durgasoftware.java12features;
2)
3) public class Test {
4)     public static void main(String[] args) throws Exception {
5)         String str = "Durga Software Solutions";
6)         System.out.println(str.describeConstable().get());
7)         System.out.println(str.resolveConstantDesc(null));
8)     }
9) }
```

OUTPUT

Durga Software Solutions  
Durga Software Solutions



## 7) Pattern Matching for instanceof Operator

In JAVA12 version, it is preview feature, its original implementation we will see in Java14 version.

In general, instanceof operator will check whether a reference variable is representing an instance of a particular class or not.

EX:

```
1) package java12features;
2) import java.util.List;
3) public class Test {
4)     public static void main(String[] args) throws Exception {
5)         Object obj = List.of(1,2,3,4,5);
6)         if(obj instanceof List) {
7)             for(int x: (List<Integer>)obj) {
8)                 System.out.println(x);
9)             }
10)        }else {
11)            System.out.println("Content is not matched");
12)        }
13)    }
14} }
```

IN the above example, if we want to do any manipulation with obj then we must convert that obj to List type then only it is possible.

IN JAVA12 version, it is not required to perform type casting directly we are able to get reference variable to use .

EX:

```
1) package java12features;
2) import java.util.List;
3) public class Test {
4)     public static void main(String[] args) throws Exception {
5)         Object obj = List.of(1,2,3,4,5);
6)         if(obj instanceof List list) {
7)             for(int x: list) {
8)                 System.out.println(x);
9)             }
10)        }else {
11)            System.out.println("Content is not matched");
12)        }
13)    }
14} }
```



**Note:** It will not run in JAVA12 version, it will execute in JAVA14 version.

### 8) Raw String Literals is Removed From JDK 12

It is a preview feature, it will not be executed in JAVA12 version.

Prior to JAVA12:

```
String html = "<html>\n" +  
    "  <body>\n" +  
    "      <p>Hello World.</p>\n" +  
    "  </body>\n" +  
"</html>\n";
```

In JAVA12 , we can remove this raw strings, in place of this we will write like below.

```
String html = `<html>  
  <body>  
    <p>Hello World.</p>  
  </body>  
</html>  
`;
```



# JAVA 13 FEATURES



# Java 13 Updations

- 1) Text Blocks
- 2) New Methods in String Class for Text Blocks
- 3) Switch Expressions Enhancements
- 4) Reimplement the Legacy Socket API
- 5) Dynamic CDS Archive
- 6) ZGC: Uncommit Unused Memory
- 7) FileSystems.newFileSystem() Method

## **1)Text Blocks:**

Upto Java12 version, if we want to write String data in more than one line then we have to use \n character in middle of the String.

EX:

```
1) package java13features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)         String address = "\tDurga Softweare Solutions\n"+
7)                         "\t202, HMDA, Mitryvanam\n"+
8)                         "\tAmeerpet, Hyd-38";
9)         System.out.println(address);
10)    }
11) }
```

### **OUTPUT**

Durga Softweare Solutions  
202, HMDA, Mitryvanam  
Ameerpet, Hyd-38

In the above code, to bring string data into new line we have to use \n character and to provide space before each and every line we have to use \t, these escape symbols may or may not support by all the operating systems.

To overcome the above problems, JAVA13 version has provided a new preview feature called as "Text Blocks", where we can write String data in multiple lines without providing \n and \t escape symbols.



## Syntax:

```
String varName = """  
    String Data in line1  
    String Data in line2  
----  
----
```

EX:

```
1) package java13features;  
2)  
3) public class Test {  
4)  
5)     public static void main(String[] args) {  
6)         String address = """  
7)             Durga Softweare Solutions  
8)             202, HMDA, Mitryvanam\n  
9)             Ameerpet, Hyd-38""";  
10)        System.out.println(address);  
11)    }  
12)  
13} }
```

## OUTPUT

Durga Softweare Solutions  
202, HMDA, Mitryvanam  
Ameerpet, Hyd-38

**Note:** If we want to execute preview features we have to enable preview Features first before execution, for thsi we have to use the following approaches.

1. If we want to execute program through Command prompt then use the following commands.

```
javac --enable-preview --release 13 Test.java  
java --enable-preview Test
```

2. In Eclipse IDE:

- 1) Right Click on Project
- 2) Properties
- 3) Java Compiler
- 4) Uncheck "Use Compliance from Execution Evenvironment JAVASE13."
- 5) Select "use --release option"
- 6) Unselect "Use default complaince Settings"
- 7) Select "Enable preview Features on Java13 version".



--> In Text Block, we have to provide data in the next line after """" and we can provide end """" either in the same line of data or in the next line, Data must not be provided in the same line of """".

**EX:**

String str1 = """Durga Software Solutions"; --> Invalid.

String str2 = """

Durga Software Solutions"""; ---> Valid

String str3 = """

Durga Software Solutions

"""; -----> Valid

--> In Java, If we provide same data in normal "" and the data in """" """" then we are able to compare by using equals() and == operators , in both the cases we are able to get true value.

**EX:**

```
1) package java13features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)         String data = "Durga Software Solutions";
7)         String newData = """
8)             Durga Software Solutions""";
9)         System.out.println(data);
10)        System.out.println(newData);
11)        System.out.println(data.equals(newData));
12)        System.out.println(data == newData);
13)    }
14) }
```

**OUTPUT**

Durga Software Solutions

Durga Software Solutions

true

true

--> It is possible to perform concatenation operation between the strings which are represented in "" and """" """.



EX:

```
1) package java13features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)         String str1 = "Durga ";
7)         String str2 = "Software";
8)         String str3 = " Solutions";
9)         String result1 = str1+str2+str3;
10)        String result2 = str1.concat(str2).concat(str3);
11)        System.out.println(result1);
12)        System.out.println(result2);
13)
14)    }
15)}
```

## OUTPUT

Durga Software Solutions

Durga Softare Solutions

--> It is possible to pass Text Blocks as parameters to the methods and it is possible to return Text Blocks from the methods.

EX1:

```
1) package java13features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)         System.out.println("Durga Software Solutions
7)                           202, HMDA, Mitryvanam
8)                           Ameerpet,Hyd-38
9)                           ");
10)
11)    }
12)}
```

## OUTPUT

Durga Software Solutions

202, HMDA, Mitryvanam

Ameerpet,Hyd-38



EX:

```
1) package java13features;
2)
3) class Employee{
4)     public String getEmpDetails(String str) {
5)         String result = "Employee Details:
6)             -----
7)             "+str+
8)             "ESAL : 15000
9)             EADDR : Hyd
10)            EEMAIL : durga@dss.com
11)            EMOBILE: 91-9988776655
12)            ";
13)
14)        return result;
15)    }
16) }
17) public class Test {
18)
19)     public static void main(String[] args) {
20)         String str = "EID : E-111
21)             ENAME : Durga
22)             ";
23)
24)         Employee emp = new Employee();
25)         String result = emp.getEmpDetails(str);
26)         System.out.println(result);
27)     }
28} }
```

## OUTPUT

Employee Details:

-----  
EID : E-111  
ENAME : Durga  
ESAL : 15000  
EADDR : Hyd  
EEMAIL : durga@dss.com  
EMOBILE: 91-9988776655

Text Block is very much usefull feature to represent Html or JSON code in Java applications.



EX:

```
1) package java13features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)         String html = """
7)             <html>
8)                 <body>
9)                     <p>Durg Software Solutions</p>.
10)                 </body>
11)             </html>
12)             """;
13)         System.out.println(html);
14)     }
15) }
```

## OUTPUT

```
<html>
    <body>
        <p>Durg Software Solutions</p>.
    </body>
</html>
```

IN the above Html code, how much spaces are provided at each and every line the same no of spaces will be provided in the output.

EX:

```
1) package java13features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)         String html = """
7)             <html>
8)                 <body>
9)                     <p>Durg Software Solutions</p>.
10)                 </body>
11)             </html>
12)             """;
13)         System.out.println(html);
14)     }
15) }
```



## OUTPUT

```
<html>
    <body>
        <p>Durg Software Solutions</p>
    </body>
</html>
```

--> On Text Blocks we can apply intend() method inorder to mange spaces.

EX:

```
1) package java13features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)         String data1 = """
7)             Durga Software Solutions
8)             """;
9)         System.out.println(data1);
10)        System.out.println(data1.indent(5));
11)        System.out.println();
12)        String data2 = """
13)             Durga Software Solutions
14)             """;
15)         System.out.println(data2);
16)         System.out.println(data2.indent(-5));
17)
18)    }
19} }
```

## OUTPUT

Durga Software Solutions

Durga Software Solutions

Durga Software Solutions

Durga Software Solutions

--> It is possible to provide ' '[Single quotations] and "" "[Double quotations] directly, but, we can provide """ "[Triple Quotations] with \ .



EX:

```
1) package java13features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)         String str = """
7)             'Durga Software Solutions'
8)             "Ameerpet"
9)             \"""
10)            """;
11)         System.out.println(str);
12)
13)     }
14) }
```

## OUTPUT

'Durga Software Solutions'  
"Ameerpet"  
"""Hyderabad"""

## 2) New Methods in String Class for Text Blocks

For Text Blocks, JAVA13 version has provided the following three methods in String class.

- 1) public String formatted(Object ... values)
- 2) public String stripIndent()
- 3) public String translateEscapes()

### 1) public String formatted(Object ... values):

In Text Blocks we can provide data formating by using the following method.

public String formatted(val1,val2,...val\_n)

EX:

```
1) package java13features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)
7)         String empDetails = """
8)             Employee Details:
9)             -----
10)            Employee Number : %d
```



```
11) Employee Name : %s
12) Employee Salary : %f
13) Employee Address : %s
14)     """;
15) System.out.println(empDetails.formatted(111,"Durga",25000.0f,"Hyd"));
16) System.out.println();
17) System.out.println(empDetails.formatted(222,"Anil",35000.0f,"Hyd"));
18) System.out.println();
19) System.out.println(empDetails.formatted(333,"Ravi",45000.0f,"Hyd"));
20} 
21}
```

## OUTPUT

**Employee Details:**

-----  
Employee Number : 111  
Employee Name : Durga  
Employee Salary : 25000.000000  
Employee Address : Hyd

**Employee Details:**

-----  
Employee Number : 222  
Employee Name : Anil  
Employee Salary : 35000.000000  
Employee Address : Hyd

**Employee Details:**

-----  
Employee Number : 333  
Employee Name : Ravi  
Employee Salary : 45000.000000  
Employee Address : Hyd



## 2) public String stripIndent():

If we have any spaces at beginning and Ending of String.

EX:

```
1) package java13features;
2)
3) public class Test {
4)
5)     public static void main(String[] args) {
6)         String data1 = "\t\tDurga Software Solutions \t\t";
7)         String data2 = "\t\tHyderabad\t\t";
8)         System.out.println(data1+data2);
9)         System.out.println(data1.stripIndent()+data2.stripIndent());
10)
11)    }
12) }
```

### OUTPUT

Durga Software Solutions  
Durga Software SolutionsHyderabad

Hyderabad

## 3) public String translateEscapes()

It able to remove Escape symbols like \ in our String data.

EX:

```
1) package java13features;
2)
3) import java.io.FileInputStream;
4) import java.io.FileOutputStream;
5)
6) public class Test {
7)
8)     public static void main(String[] args)throws Exception {
9)
10)        String str1 = "Durga\nSoftware\nSolutions";
11)        String str2 = "Durga\\nSoftware\\nSolutions";
12)        System.out.println(str1);
13)        System.out.println();
14)        System.out.println(str2);
15)        System.out.println();
16)        System.out.println(str2.translateEscapes());
17)    }
```



| 18 ) }

## OUTPUT

Durga  
Software  
Solutions

Durga\nSoftware\nSolutions

Durga  
Software  
Solutions

## **3) Switch Expressions Enhancements:**

Switch Expression in JAVA12 was introduced as preview feature, in JAVA13 also it was very same except break statement. In JAVA13 version, we can use "yield" in place of break statement to return a value.

### Note:

In JAVA13 version, we are able to use switch as statement and as Expression.

In Switch statement we will use "yield" keyword to return a value in place of break statement.

IN Switch Expression, we will use -> to return value.

EX:

```
1) public class Hello {  
2)     public static void main(String[] args) {  
3)         var val = "SUNDAY";  
4)         var result = switch (val){  
5)             case "MONDAY", "TUESDAY", "WENSDAY", "THURSDAY", "FRIDAY":  
6)                 yield "Week Day";  
7)             case "SATURDAY", "SUNDAY":  
8)                 yield "Weekend";  
9)             default:  
10)                 yield "Not a Day";  
11)         };  
12)         System.out.println(result);  
13)     }  
14) }
```

## OUTPUT

Weekend



EX:

```
1) public class Hello {  
2)     public static void main(String[] args) {  
3)         var val = "SUNDAY";  
4)         var result = switch (val){  
5)             case "MONDAY","TUESDAY","WENSDAY","THURSDAY","FRIDAY" ->  
6)                 "Week Day";  
7)             case "SATURDAY","SUNDAY" -> "Weekend";  
8)             default -> "Not a Day";  
9)         };  
10)        System.out.println(result);  
11)    }
```

## OUTPUT

Weekend

## **4) Reimplement the Legacy Socket API:**

Upto JAVA12 version, Socket API contains old library which was provided before JDK1.0 version which uses `java.net.Socket` and `java.net.ServerSocket`, JAVA13 version has provided new Implementation for Socket API which is based on `NioSocketImpl` inplace of `PlainSocketImpl`.

New Socket API used `java.util.concurrent` locks rather than synchronized methods.

If you want to use the legacy implementation, use the java option -  
`-Djdk.net.usePlainSocketImpl` for backword compatibility.

## **5) Dynamic CDS Archive:**

Class Data Sharing was introduced in JAVA10 version, it was difficult to prepare cds, but, JAVA13 version simplifies CDS creation by using the following command.

```
java -XX:ArchiveClassesAtExit=my_app_cds.jsa -cp my_app.jar
```

```
java -XX:SharedArchiveFile=my_app_cds.jsa -cp my_app.jar
```

## **6) ZGC: Uncommit Unused Memory:**

Z Garbage Collector was introduced in JAVA11 version, it has provide small span of time before clean up memory and the unused memory was not returned to Operating System, but, JAVA13 version has provided an enhancement over Z garbage Collector, it will return uncommitted and unused memory to Operating System.



### 7) FileSystems.newFileSystem() Method

Java1e3 version has provided three new methods have been added to the FileSystems class to make it easier to use file system providers, which treats the contents of a file as a file system.

```
newFileSystem(Path)
newFileSystem(Path, Map<String, ?>)
newFileSystem(Path, Map<String, ?>, ClassLoader)
```



# JAVA 14 FEATURES



# Java 14 Updations

Developers Required Features:

- 1) Switch Expressions (Standard)
- 2) Pattern Matching for instanceof (Preview)
- 3) Helpful NullPointerExceptions
- 4) Records (Preview)
- 5) Text Blocks (Second Preview)

## 1) Switch Expressions (Standard):

JAVA12 version has introduced switch expression with break statement and lambda syntaxes, which allows multiple case labels.

Syntaxes:

```
var result = switch(value){  
    case l1,l2,l3:  
        break value;  
    ---  
    default:  
        break value;  
}
```

```
var result = switch(value){  
    case l1,l2,l3 -> value;  
    ---  
    default -> value;  
}
```

In Java 13 version, we must use "yield" keyword instead of "break" to return value from switch.

```
var result = switch(value){  
    case l1,l2,l3:  
        yield value;  
    ---  
    default:  
        yield value;  
}
```

JAVA14 version made switch statement as standard feature with lambda syntax and "yield" keyword.



## Syntax:

```
var result = switch(value){  
    case l1,l2,l3 ->{  
        yield value;  
    }  
    ----  
    default ->{  
        yield value;  
    }  
}
```

EX:

```
1) package java14features;  
2)  
3) import java.util.Scanner;  
4)  
5) public class Test {  
6)     public static void main(String[] args) {  
7)         var scanner = new Scanner(System.in);  
8)         System.out.print("Enter Day Of WEEK : ");  
9)         var value = scanner.next().toUpperCase();  
10)        var result = switch (value){  
11)            case "MONDAY", "TUESDAY", "WEDNESDAY", "THURSDAY", "FRIDAY" ->  
12)                "Weekday";  
13)            case "SATURDAY", "SUNDAY" -> "Weekend";  
14)            default ->{  
15)                if(value.isEmpty()){  
16)                    yield "value to switch is empty, please check";  
17)                }else{  
18)                    yield value+"";  
19)                Value must be in  
20)                1.MONDAY  
21)                2.TUESDAY  
22)                3.WEDNESDAY  
23)                4.THURSDAY  
24)                5.FRIDAY  
25)                6.SATURDAY  
26)                7.SUNDAY  
27)                "";  
28)            }  
29)        }  
30)    };  
31)    System.out.println("You Entered :" +result);
```



```
32) }  
33) }
```

## OUTPUT

Enter Day Of WEEK : MONDAY

You Entered :Weekday

## OUTPUT

Enter Day Of WEEK : SUNDAY

You Entered :Weekend

## OUTPUT

Enter Day Of WEEK : XXXDAY

You Entered :XXXDAY

Value must be in

- 1.MONDAY
- 2.TUESDAY
- 3.WEDNESDAY
- 4.THURSDAY
- 5.FRIDAY
- 6.SATURDAY
- 7.SUNDAY

**Note:** To run the above Example we must enable Java14 preview features, for this, we have to use the following changes in IntelliJ Idea

- 1) Right Click on Project.
- 2) Open Module Settings
- 3) At Language Level, Select "14(preview) Records, Patterns, Text Blocks" option
- 4) Click on "OK" button.

## 2) Pattern Matching for instanceof (Preview)

It was introduced in JAVA12 version as preview feature, it allows instanceof operator with pattern matching capability.

JAVA14 is also making this feature as preview feature, no changes in its feature.

Before Java14 and 12, 'instanceof' operator:

```
1) package java14features;  
2) public class Test {  
3)     public static void main(String[] args) {  
4)         Object obj = "Durga Software Solutions";  
5)         if(obj instanceof String){  
6)             String str = (String)obj;
```



```
7)     System.out.println(str.toUpperCase());
8) }else{
9)     System.out.println("obj is not having String instance");
10) }
11)
12}
```

## OUTPUT

DURGA SOFTWARE SOLUTIONS

In JAVA12 and JAVA14 version:

```
1) package java14features;
2)
3) public class Test {
4)     public static void main(String[] args) {
5)         Object obj = "Durga Software Solutions";
6)         if(obj instanceof String str){
7)             System.out.println(str.toUpperCase());
8)         }else{
9)             System.out.println("obj is not having String instance");
10)        }
11)
12}
```

## OUTPUT

DURGA SOFTWARE SOLUTIONS

### **3) Helpful NullPointerException:**

In JAVA14 version, NullpointerException was redesigned with more meaningfull manner.

EX:

```
1) package java14features;
2)
3) import java.util.Date;
4)
5) public class Test {
6)     public static void main(String[] args) {
7)         Date d = null;
8)         System.out.println("To Day : "+d.toString());
9)     }
10}
```



If we run the above application Before JAVA14 version:

Exception in thread "main" java.lang.NullPointerException  
at java14features.Test.main(Test.java:8)

In Java14 version, if we want to get NullPointerException helpfull message then we have to execute Java program with "-XX:+ShowCodeDetailsInExceptionMessages" flag along with "java" command.

D:\java14features>javac Test.java

D:\java14features>java -XX:+ShowCodeDetailsInExceptionMessages Test  
Exception in thread "main" java.lang.NullPointerException: Cannot invoke  
"java.util.Date.toString()" because "<local1>" is null  
at Test.main(Test.java:7)

D:\java14features>

**Note:** To run the above program and to get NullPointerException Helpfull Message then we have to set "-XX:+ShowCodeDetailsInExceptionMessages" flag in VM Option in Run/Debug Configurations.

- 1) Select "Run" in Menubar
- 2) Select "Edit Configurations".
- 3) Provide "-XX:+ShowCodeDetailsInExceptionMessages" in Edit Configurations.
- 4) Click on "Apply" then "OK" buttons.

Run the above program we will get the following output.

java.lang.NullPointerException: Cannot invoke "java.util.Date.toString()" because "d" is null  
at java14features.Test.main(Test.java:9)

## 4) Records (Preview):

In general, in Java applications, we are able to write data carriers like Java bean class, where in Java bean classes we are able to provide the following elements.

- 1) Private properties
- 2) Public accessor and mutator methods
- 3) Inplace of mutator methods we may declare parameterized constructors depending on application requirements
- 4) equals(), hashCode() and toString() methods as per the requirement.



EX:

```
1) package java14features;
2)
3) class Employee{
4)     private int eno;
5)     private String ename;
6)     private float esal;
7)     private String eaddr;
8)
9)     Employee(int eno, String ename, float esal, String eaddr){
10)         this.eno = eno;
11)         this.ename = ename;
12)         this.esal = esal;
13)         this.eaddr = eaddr;
14)     }
15)
16)     public int getEno() {
17)         return eno;
18)     }
19)
20)     public String getName() {
21)         return ename;
22)     }
23)
24)     public float getEsal() {
25)         return esal;
26)     }
27)
28)     public String getEaddr() {
29)         return eaddr;
30)     }
31)}
32) public class Test {
33)     public static void main(String[] args) {
34)         Employee emp = new Employee(111,"AAA",5000,"Hyd");
35)         System.out.println("Employee Details");
36)         System.out.println("-----");
37)         System.out.println("Employee Number : "+emp.getEno());
38)         System.out.println("Employee Name : "+emp.getName());
39)         System.out.println("Employee Salary : "+emp.getEsal());
40)         System.out.println("Employee Address : "+emp.getEaddr());
41)
42)     }
43) }
```



## OUTPUT

### Employee Details

```
Employee Number : 111
Employee Name : AAA
Employee Salary : 5000.0
Employee Address : Hyd
```

In the above example, in Employee class we have provided lot of Boilarplatcode like parameterized constructor, accessor methods,.....

To remove the boilarplatcode in the above application JAVA14 version has provided a preview feature that is "Record".

Record is a simple Data carrier, it will include properties, constructor , accessor methods ,  
`toString()`,  
`hashCode()`, `equals()`,.....

#### Syntax:

```
record Name(parameterList){  
}
```

#### EX:

```
record Employee(int eno, String ename, float esal, String eaddr){  
}
```

If we compile the above code the compiler will translate the above code like below.

```
final class Employee extends java.lang.Record {  
    public Employee(int, java.lang.String, float, java.lang.String);  
    public java.lang.String toString();  
    public final int hashCode();  
    public final boolean equals(java.lang.Object);  
    public int eno();  
    public java.lang.String ename();  
    public float esal();  
    public java.lang.String eaddr();  
}
```

We are able to get the above code with "javap" command after the compilation.



From the above, we will conclude that,

1. Every record must be converted to a final class

**Note:** If record is final class then it is not possible to define inheritance between records.

2. Every record should be a child class to `java.lang.Record`, where `java.lang.Record` is providing `hashCode()`, `equals()`, `toString()` methods to record.

```
public abstract class java.lang.Record {  
    protected java.lang.Record();  
    public abstract boolean equals(java.lang.Object);  
    public abstract int hashCode();  
    public abstract java.lang.String toString();  
}
```

3. Every records contains a parameterized constructor called as Canonical Constructor, where parameters of canonical constructor are same as the parameter types which we provided in record declaration.

4. In record, all the parameters are converted as private and final, we are unable to access them in out side of the record and we are unable to modify their values.

5. In record, for each and every property a seperate accessor method is provided , where accessor methods names are same as the property names like `propertyName()` , it will not follow `getXXX()` methods convention.

6. In Records, `toString()` method was implemented insuch a way to return a string contains "`RecordName[Prop1=Value1,Prop2=Value2,...]`"

7. In Records, `equals()` method was implemented in such a way to perform content comparision instead of references comparision.

**Note:** In JAVA/J2EE applications, we are able to use records as an alternatives to Java Bean classes.

EX:

```
1) package java14features;  
2) record Employee(int eno, String ename, float esal, String eaddr){  
3)  
4) }  
5) public class Test {  
6)     public static void main(String[] args) {  
7)         Employee emp = new Employee(111,"AAA",5000,"Hyd");
```



```
8) System.out.println("Employee Details");
9) System.out.println("-----");
10) System.out.println("Employee Number : "+emp.eno());
11) System.out.println("Employee Name : "+emp.ename());
12) System.out.println("Employee Salary : "+emp.esal());
13) System.out.println("Employee Address : "+emp.eaddr());
14) }
15) }
```

## OUTPUT

Employee Details

```
-----  
Employee Number : 111  
Employee Name : AAA  
Employee Salary : 5000.0  
Employee Address : Hyd
```

EX:

```
1) package java14features;
2)
3) record Employee(int eno, String ename, float esal, String eaddr){
4)
5) }
6) public class Test {
7)     public static void main(String[] args) {
8)         Employee emp1 = new Employee(111,"Durga", 5000, "Hyd");
9)         Employee emp2 = new Employee(111,"Durga", 5000, "Hyd");
10)        Employee emp3 = new Employee(222,"BBB", 6000, "Hyd");
11)        System.out.println(emp1);
12)        System.out.println(emp2);
13)        System.out.println(emp3);
14)        System.out.println(emp1.equals(emp2));
15)        System.out.println(emp1.equals(emp3));
16)
17)    }
18) }
```

## Output

```
Employee[eno=111, ename=Durga, esal=5000.0, eaddr=Hyd]
Employee[eno=111, ename=Durga, esal=5000.0, eaddr=Hyd]
Employee[eno=222, ename=BBB, esal=6000.0, eaddr=Hyd]
true
false
```



In record we are able to declare our own variables in the body part, but, it must be static, because, Record body is not allowing instance variables.

EX:

```
1) package java14features;
2) record Employee(int eno, String ename, float esal, String eaddr){
3)     static String eemail;
4)
5)     public static String getEmail() {
6)         return eemail;
7)     }
8)
9)     public static void setEmail(String eemail) {
10)        Employee.eemail = eemail;
11)    }
12)
13) public class Test {
14)     public static void main(String[] args) {
15)         Employee emp = new Employee(111,"AAA",5000,"Hyd");
16)         emp.setEmail("aaa@gmail.com");
17)         System.out.println("Employee Details");
18)         System.out.println("-----");
19)         System.out.println("Employee Number : "+emp.eno());
20)         System.out.println("Employee Name : "+emp.ename());
21)         System.out.println("Employee Salary : "+emp.esal());
22)         System.out.println("Employee Address : "+emp.eaddr());
23)         System.out.println("Employee Email Id : "+emp.getEmail());
24)
25)     }
26)}
```

## OUTPUT

Employee Details

-----  
Employee Number : 111  
Employee Name : AAA  
Employee Salary : 5000.0  
Employee Address : Hyd  
Employee Email Id : aaa@gmail.com

In Records, we are able to manipulate accessor methods body by writing explicitly.



EX:

```
1) package java14features;
2)
3) record User(String fname, String lname, String address){
4)     public String fname(){
5)         return "My First Name : "+fname;
6)     }
7)     public String lname(){
8)         return "My Last Name : "+lname;
9)     }
10)    public String address(){
11)        return "My Address is : "+address;
12)    }
13)
14) public class Test {
15)     public static void main(String[] args) {
16)         User user = new User("Java14 version", "JAVA", "Oracle");
17)         System.out.println("User Details");
18)         System.out.println("-----");
19)         System.out.println(user.fname());
20)         System.out.println(user.lname());
21)         System.out.println(user.address());
22)     }
23} }
```

## OUTPUT

User Details

-----  
My First Name : Java14 version  
My Last Name : JAVA  
My Address is : Oracle

In records, if we want to declare our own constructors then it is possible with the following conditions.

1. First we have to implement canonical constructor explicitly.
2. Declare our own constructor and it must access canonical constructor by using this keyword.



EX:

```
1) package java14features;
2) record Employee(int eno, String ename, float esal, String eaddr){
3)     public Employee(){
4)         this(111,"AAA",5000,"Hyd");
5)     }
6)     public Employee(int eno, String ename, float esal){
7)         this(eno,ename,esal,"Hyd");
8)     }
9)     public Employee(int eno, String ename, float esal, String eaddr){
10)        this.eno = eno;
11)        this.ename = ename;
12)        this.esal = esal;
13)        this.eaddr = eaddr;
14)    }
15)
16}
17) public class Test {
18)     public static void main(String[] args) {
19)         Employee emp = new Employee();
20)         System.out.println("Employee Details");
21)         System.out.println("-----");
22)         System.out.println("Employee Number : "+emp.eno());
23)         System.out.println("Employee Name : "+emp.ename());
24)         System.out.println("Employee Salary : "+emp.esal());
25)         System.out.println("Employee Address : "+emp.eaddr());
26)         System.out.println();
27)         Employee emp1 = new Employee(111,"AAA", 5000);
28)         System.out.println("Employee Details");
29)         System.out.println("-----");
30)         System.out.println("Employee Number : "+emp1.eno());
31)         System.out.println("Employee Name : "+emp1.ename());
32)         System.out.println("Employee Salary : "+emp1.esal());
33)         System.out.println("Employee Address : "+emp1.eaddr());
34)     }
35}
```



EX:

```
1) package java14features;
2) record Employee(int eno, String ename, float esal, String eaddr){
3)     public Employee(){
4)         this(111, "AAA", 5000, "Hyd");
5)     }
6)     public Employee(int eno){
7)         this(eno, "AAA", 5000, "Hyd");
8)     }
9)     public Employee(int eno, String ename){
10)        this(eno, ename, 5000, "Hyd");
11)    }
12)    public Employee(int eno, String ename, float esal){
13)        this(eno, ename, esal, "Hyd");
14)    }
15)    public Employee(int eno, String ename, float esal, String eaddr){
16)        this.eno = eno;
17)        this.ename = ename;
18)        this.esal = esal;
19)        this.eaddr = eaddr;
20)    }
21)    public void getEmpDetails(){
22)        System.out.println("Employee Details");
23)        System.out.println("-----");
24)        System.out.println("Employee Number : "+eno);
25)        System.out.println("Employee Name : "+ename);
26)        System.out.println("Employee Salary : "+esal);
27)        System.out.println("Employee Address : "+eaddr);
28)    }
29) }
30) public class Test {
31)     public static void main(String[] args) {
32)         Employee emp1 = new Employee();
33)         emp1.getEmpDetails();
34)         System.out.println();
35)         Employee emp2 = new Employee(222);
36)         emp2.getEmpDetails();
37)         Employee emp3 = new Employee(333, "BBB");
38)         emp3.getEmpDetails();
39)         System.out.println();
40)         Employee emp4 = new Employee(444, "CCC", 7000);
41)         emp4.getEmpDetails();
42)         Employee emp5 = new Employee(555, "DDD", 8000, "Chennai");
43)         emp5.getEmpDetails();
44)     }
```



---

| 45 ) }

## OUTPUT

### Employee Details

---

Employee Number : 111  
Employee Name : AAA  
Employee Salary : 5000.0  
Employee Address : Hyd

### Employee Details

---

Employee Number : 111  
Employee Name : AAA  
Employee Salary : 5000.0  
Employee Address : Hyd

### Employee Details

---

Employee Number : 333  
Employee Name : BBB  
Employee Salary : 5000.0  
Employee Address : Hyd

### Employee Details

---

Employee Number : 444  
Employee Name : CCC  
Employee Salary : 7000.0  
Employee Address : Hyd

### Employee Details

---

Employee Number : 555  
Employee Name : DDD  
Employee Salary : 8000.0  
Employee Address : Chennai



EX:

```
1) package java14features;
2) record Employee(int eno, String ename, float esal, String eaddr){
3)     public Employee(){
4)         this(111);
5)     }
6)     public Employee(int eno){
7)         this(eno, "AAA");
8)     }
9)     public Employee(int eno, String ename){
10)        this(eno, ename, 5000);
11)    }
12)    public Employee(int eno, String ename, float esal){
13)        this(eno, ename, esal, "Hyd");
14)    }
15)    public Employee(int eno, String ename, float esal, String eaddr){
16)        this.eno = eno;
17)        this.ename = ename;
18)        this.esal = esal;
19)        this.eaddr = eaddr;
20)    }
21)    public void getEmpDetails(){
22)        System.out.println("Employee Details");
23)        System.out.println("-----");
24)        System.out.println("Employee Number : "+eno);
25)        System.out.println("Employee Name : "+ename);
26)        System.out.println("Employee Salary : "+esal);
27)        System.out.println("Employee Address : "+eaddr);
28)    }
29)
30) public class Test {
31)     public static void main(String[] args) {
32)         Employee emp1 = new Employee();
33)         emp1.getEmpDetails();
34)     }
35} }
```

## OUTPUT

Employee Details

-----  
Employee Number : 111  
Employee Name : AAA  
Employee Salary : 5000.0



Employee Address : Hyd

**Note:** In the above example, we accessed one constructor to another constructor by using 'this()' keyword, so Records are allowing "Constructor Chaining".

**Note:** in the above all constructors , we have provided more than one constructor in single record , so that, Records are allowing "Constructor Overloading".

In Record, we are able to declare a constructor with out parameter and with out () then that constructor is called as "Compact Constructor", it will be accessed just before executing canonical constructor and it will be used mainly for Data validations.

EX:

```
1) package java14features;
2) record A(int i, int j){
3)     public A{
4)         System.out.println("Compact Constructor");
5)     }
6)
7)     public void add(){
8)         System.out.println("ADD : "+(i+j));
9)     }
10}
11) public class Test {
12)     public static void main(String[] args) {
13)         A a = new A(10,20);
14)         a.add();
15)     }
16}
```

## OUTPUT

ADD : 30

EX:

```
1) package java14features;
2) record User(String name, String pwd, int age, String email){
3)     public User{
4)         if(name == null || name.equals("")){
5)             System.out.println("Name is Required");
6)         }
7)         if(pwd == null || pwd.equals("")){
8)             System.out.println("Password is Required");
9)         }else{
10)            if(pwd.length() < 6){
```



```
11)     System.out.println("Password length must be minimum 6 characters");
12) }
13) if(pwd.length() > 10){
14)     System.out.println("Password length must be maximum 10 characters");
15) }
16) }
17) if(age <= 0){
18)     System.out.println("Age is required and it must be +ve value");
19) }else{
20)     if(age < 18 || age > 25){
21)         System.out.println("Age must be in the range from 18 to 25");
22)     }
23) }
24) if(email == null || email.equals("")){
25)     System.out.println("Email Id is Required");
26) }else{
27)     if(!email.endsWith("@durgasoft.com")){
28)         System.out.println("Email Id is Invalid");
29)     }
30) }
31) }
32) public void getUserDetails(){
33)     System.out.println("User Details");
34)     System.out.println("-----");
35)     System.out.println("User Name : "+name);
36)     System.out.println("User Password : "+pwd);
37)     System.out.println("User Age : "+age);
38)     System.out.println("User Email : "+email);
39) }
40) }
41) public class Test {
42)     public static void main(String[] args) {
43)         User user = new User("Durga","durga123",22,"durga@durgasoft.com");
44)         user.getUserDetails();
45)     }
46) }
```

In Java, records are not extending any class,because, records are final classes and it is not possible to provide inheritance relation between records.

EX:

```
record Person() {
}
record Employee() extends Person {
}
Status: Invalid, Compilation Error
```



In Java, records can implement interface/Interfaces.

EX:

```
1) package java14features;
2)
3) import java.io.Serializable;
4)
5) interface Car{
6)     public void getCarDetails();
7)
8) record FordCar(String model, String type, int price) implements Car, Serializable {
9)     @Override
10)    public void getCarDetails() {
11)        System.out.println("Ford car Details");
12)        System.out.println("-----");
13)        System.out.println("Car Model : "+model());
14)        System.out.println("Car Type : "+type());
15)        System.out.println("Car Price : "+price());
16)    }
17)
18) public class Test {
19)     public static void main(String[] args) {
20)         Car fordCar = new FordCar("2015","EchoSport",1200000);
21)         fordCar.getCarDetails();
22)     }
23} }
```

## OUTPUT

Ford car Details

-----  
Car Model : 2015  
Car Type : EchoSport  
Car Price : 1200000

In Java14, we are able to get Records details by using Reflection API.

IN Java14, `java.lang.Class` has provided the following methods.

1. `public boolean isRecord()`

--> It will check whether the component is Record or not.

2. `public RecordComponent[] getRecordComponents()`

--> It able to return all parameters of the Records in the form of `RecordComponent[]`.

**Note:** Where `RecordComponent` is able to provide Single Record Component metadata.



EX:

```
1) package java14features;
2)
3) import java.lang.reflect.RecordComponent;
4)
5) record Customer(String cid, String cname, String caddr) {
6) }
7) public class Test {
8)     public static void main(String[] args) {
9)         System.out.println("Is Customer Record? : "+Customer.class.isRecord());
10)        System.out.print("Customer Record Components : ");
11)        for(RecordComponent comp: Customer.class.getRecordComponents()){
12)            System.out.print(comp.getName()+" ");
13)        }
14)    }
15) }
```

## OUTPUT

Is Customer Record? : true  
Customer Record Components : cid cname caddr

## 5) Text Blocks (Second Preview)

Text Blocks are introduced in JAVA13 as Preview version, it allows to declare a string value in more than one line and it is very much usefull when we want tp write html code or JSON code or Sql queries in Java programs.

JAVA14 version made Text Blocks still Preview feature, but, it has allowed to use \[Back Slash] symbol to improve look and feel and it will provide multiple lines of String into single line and \s to preserve trainling spaces in the lines.

EX:

```
1) package java14features;
2)
3) import java.lang.reflect.RecordComponent;
4) public class Test {
5)     public static void main(String[] args) {
6)         String address = """";
7)             Durga Software Solutions\
8)             202, HMDA, Mitrivanam\
9)             Ameerpet\
10)            Hyderabad-38\
11)            """;;
12)         System.out.println(address);
```



```
13) System.out.println();
14)
15)}
```

## OUTPUT

Durga Software Solutions202, HMDA, MitrivanamAmeerpetHyderabad-38

EX:

```
1) package java14features;
2)
3) import java.lang.reflect.RecordComponent;
4) public class Test {
5)   public static void main(String[] args) {
6)     String address = "Durga Software Solutions\n"
7)                 + "202, HMDA, Mitrivanam\n"
8)                 + "Ameerpet\n"
9)                 + "Hyderabad-38\n"
10)                + "";
11)
12)   System.out.println(address);
13)   System.out.println();
14)
15)}
```

## OUTPUT

Durga Software Solutions  
202, HMDA, Mitrivanam  
Ameerpet  
Hyderabad-38

**Note:** In the above , Every line has Single Space at end.

In Text Block, we can use both \s for space and \ for keeping mnultiple lines in single line.

EX:

```
1) package java14features;
2)
3) import java.lang.reflect.RecordComponent;
4) public class Test {
5)   public static void main(String[] args) {
6)     String address = "Durga Software Solutions\s\s\
7)                 + "202, HMDA, Mitrivanam\s\s\"
```



```
9)      Ameerpet\\s\\s\\
10)     Hyderabad-38\\s\\s\\
11)     """;
12)     System.out.println(address);
13)     System.out.println();
14)
15)   }
16} }
```

## OUTPUT

Durga Software Solutions 202, HMDA, Mitrivanam Ameerpet Hyderabad-38