

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>
```

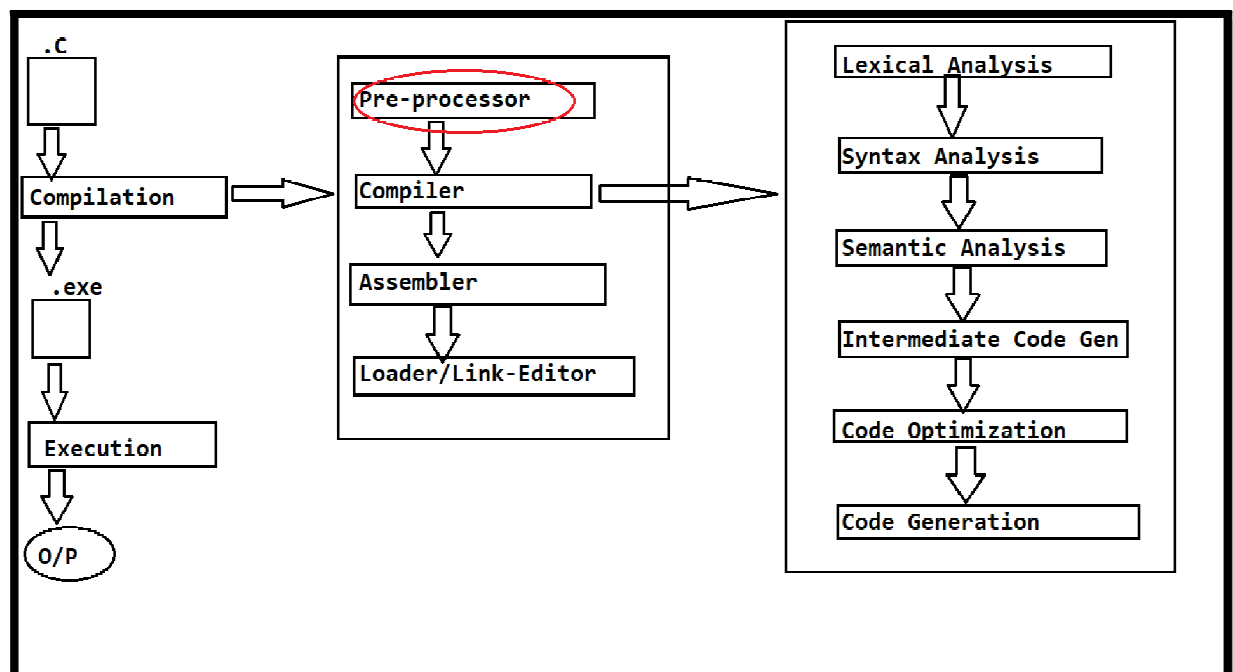
```
#include<conio.h>
```

```
#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:

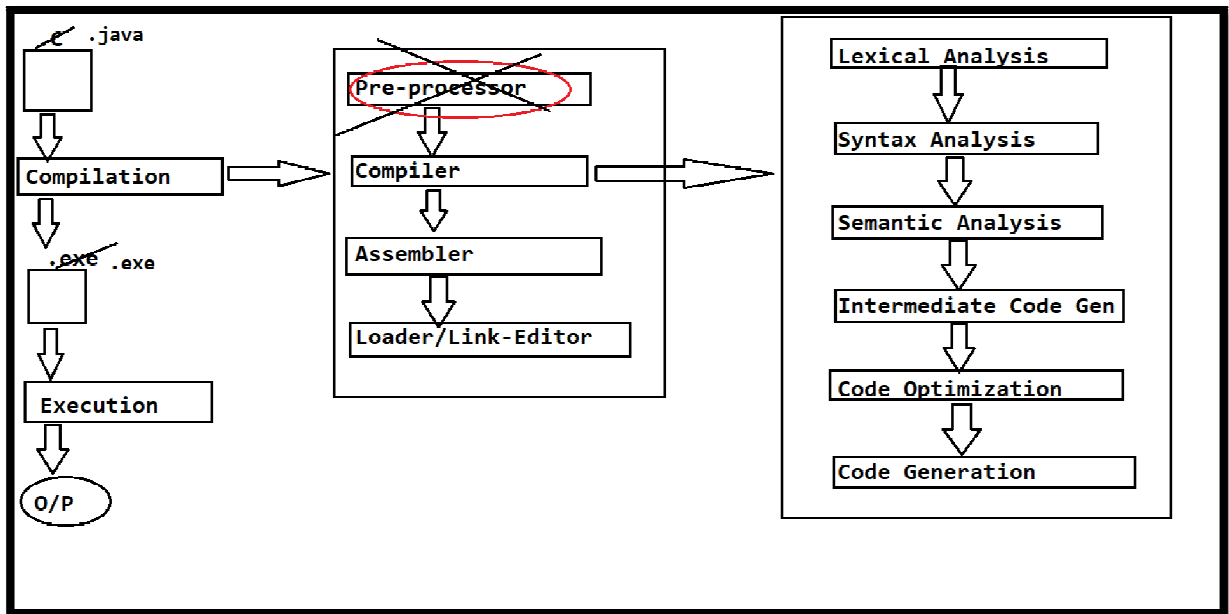
**import java.io.*;
import java.util.*;
import java.sql.*;**

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>
```

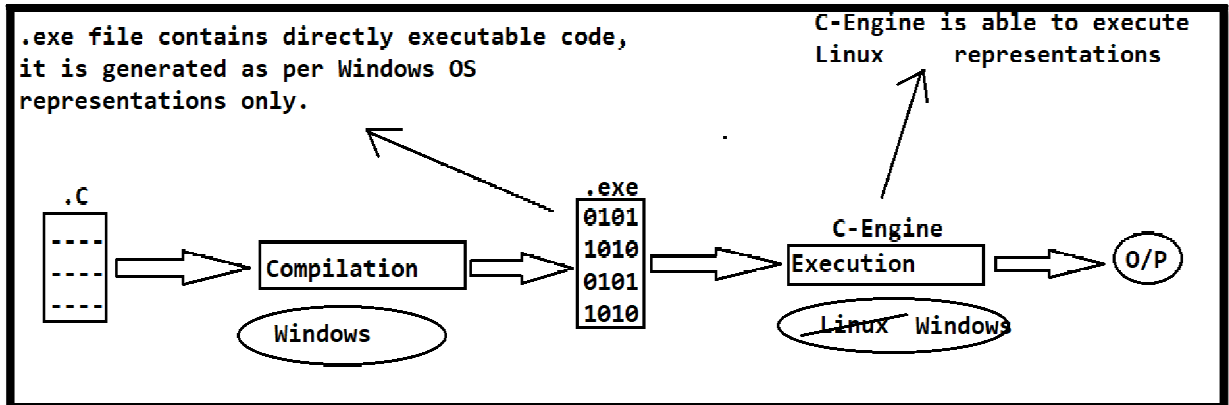
By using single import statement we are able to include more than one class or more than one interface of the same package.

EX: import java.io.*;

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

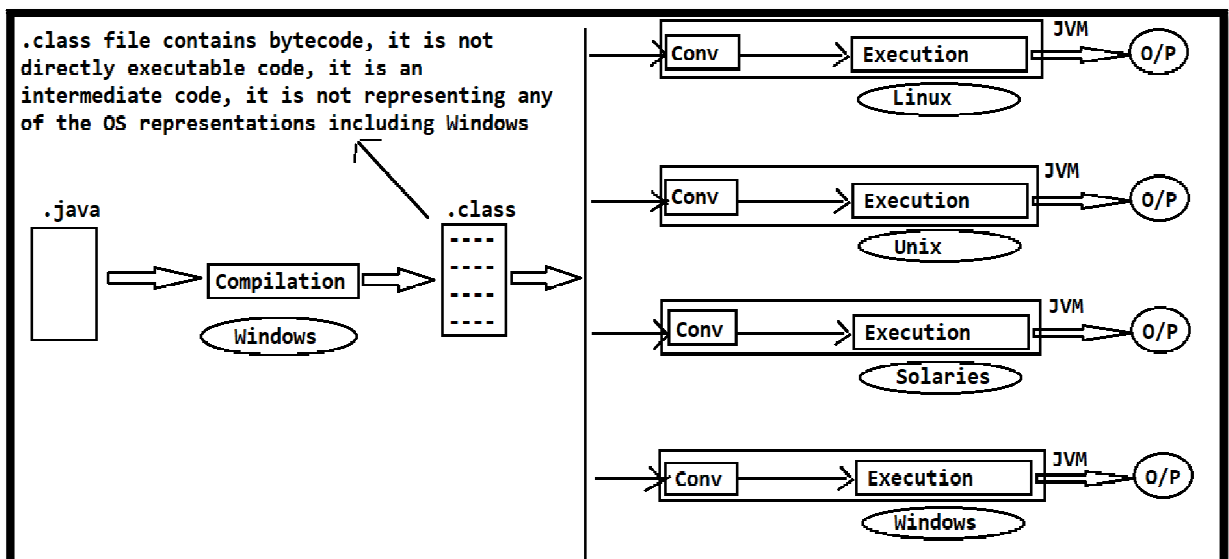
If any PL allows its applications to perform compilation and execution on the same Operating System then that PL is called as Platform Dependent PL.

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 Programming 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 variable called as "Pointer" variable.

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

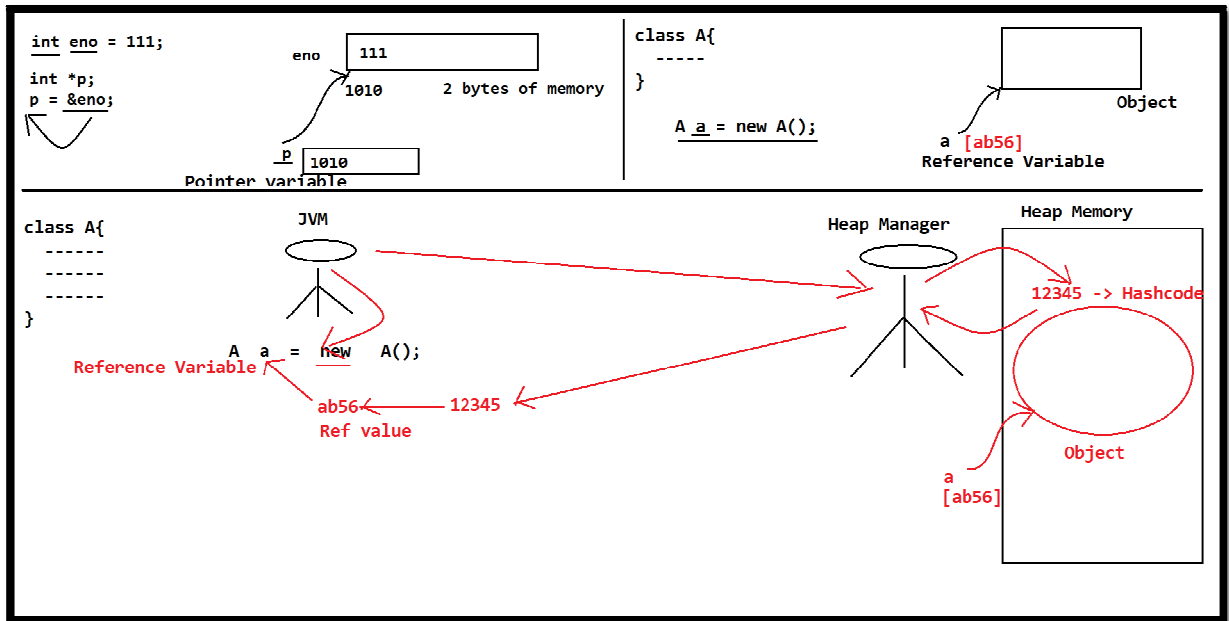
Q)What are the differences between pointer variables and reference variables?

- 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 powerful 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] in order 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.

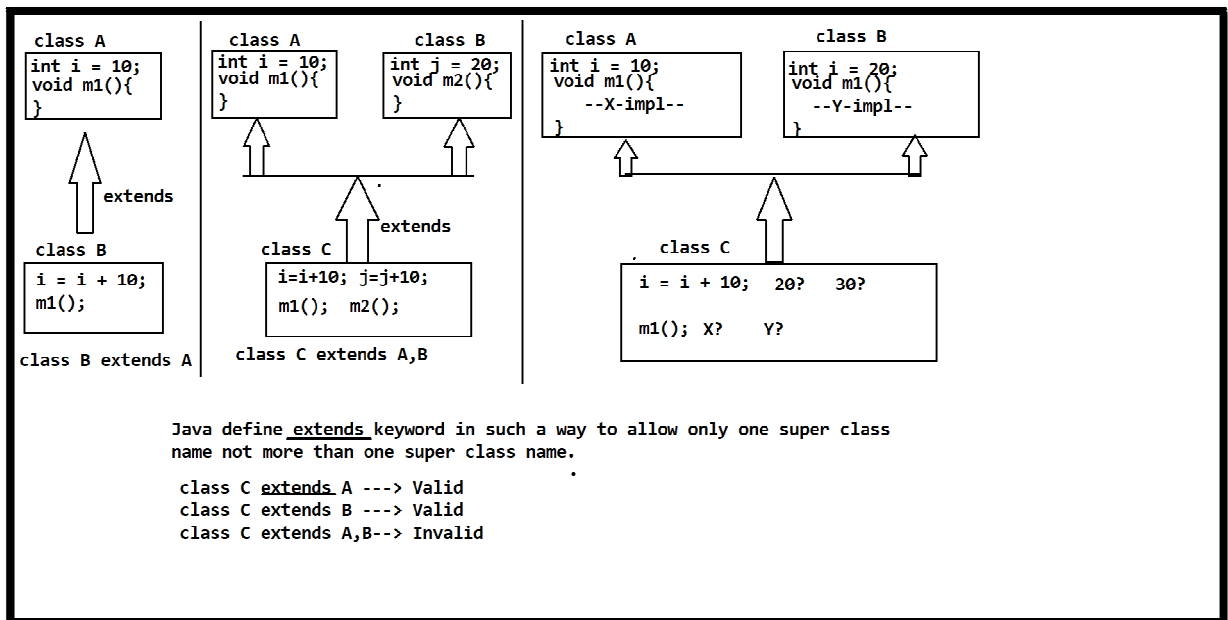
If we declare same variable with different values and same method with different implementation in both the super classes and if we access that common variable and method at sub class then which super class variable and method will be accessed is big confusion for the compiler and JVM, JAVA is simple 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 ne 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 repersent 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 memroy 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 Threadadd
- 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 seperate middleware service in the form of JAAS [Java Authetication And Aurtherization Service] inorder 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 with out 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 separate 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 is 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 is 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 interpreter inside JVM.

12) High Performance:

JAVA is high performance programming language due to its rich set of features like Platform independent, Arch Neutral, 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 for 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 programatically, 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 XML 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 applicatinos, 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

Annotation Based Tech [XML documents Optional]

- 1) JDK5.0 and above
- 2) JDBC4.0
- 3) Servlets3.0
- 4) Struts2.x
- 5) JSF2.x
- 6) EJBs3.x
- 7) Spring3.x

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 repersent 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,

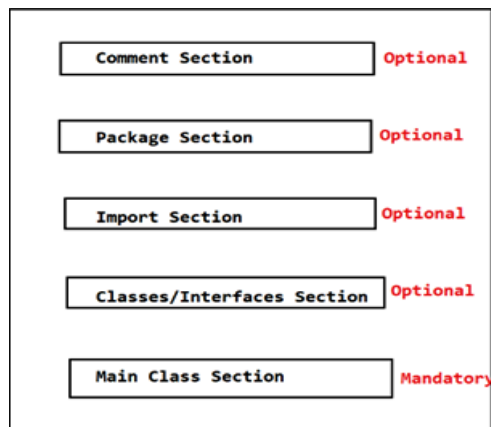
1.To manage application logic which we want to execute by JVM directly we have to use main() method.

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 VArables.." 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 prederfined 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 voilate 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 Compiler Software is activated then Java Compiler 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.

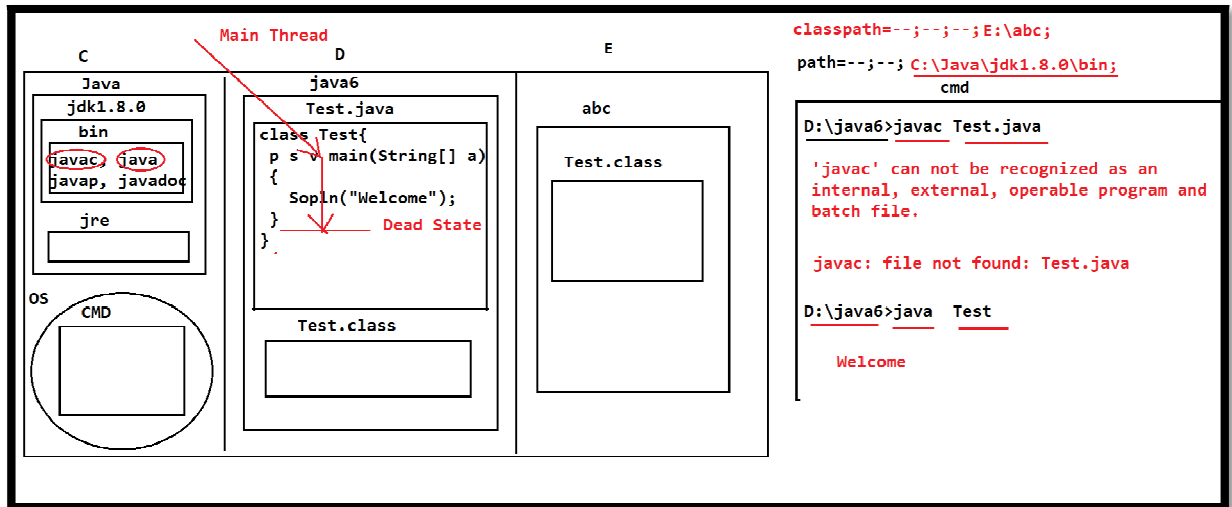
```
JAVA6: java.lang.NoSuchMethodError:main
```

```
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

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 boudaries, 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 email="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 moddle 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

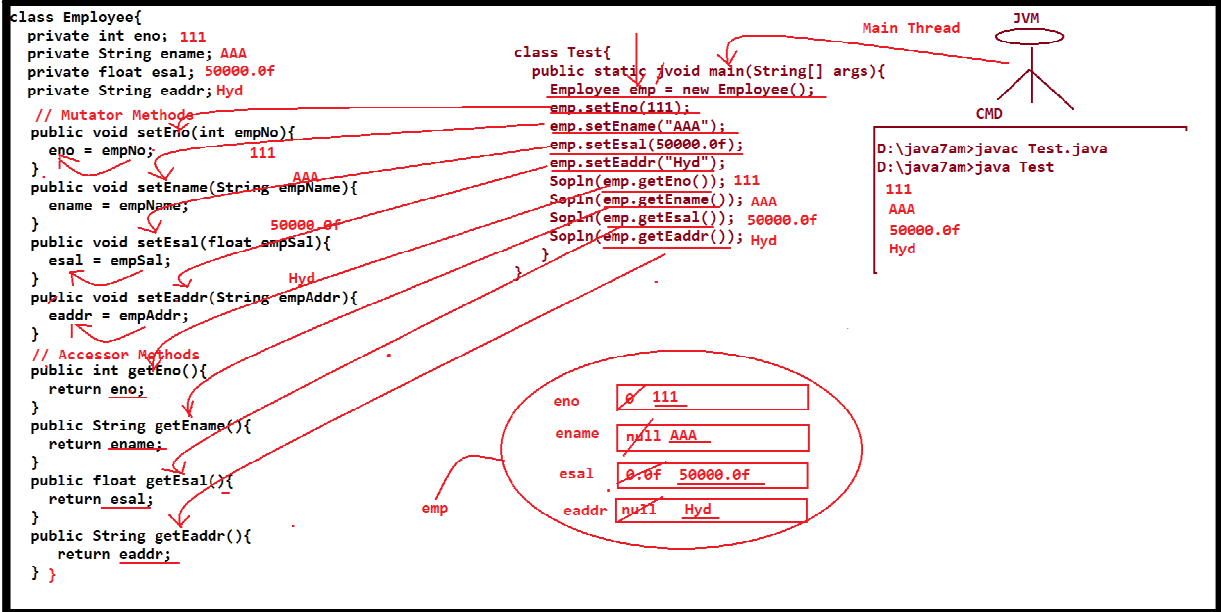
Q) What are the differences between *Mutator methods* and *Accessor methods*?

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-Arg 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-Arg Method" provided by JDK5.0 version.

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

Syntax for Var-Arg Parameter:

Data_Type ... Var_Name

EX: for var-Arg method:

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

When we access Var-Arg 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-Arg methods, normal parameters are allowed along with Var-Arg parameter but they must be provided before var-Arg parameter, not after var-Arg parameter, because, var-Arg parameter must be last parameter in var-Arg method.

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

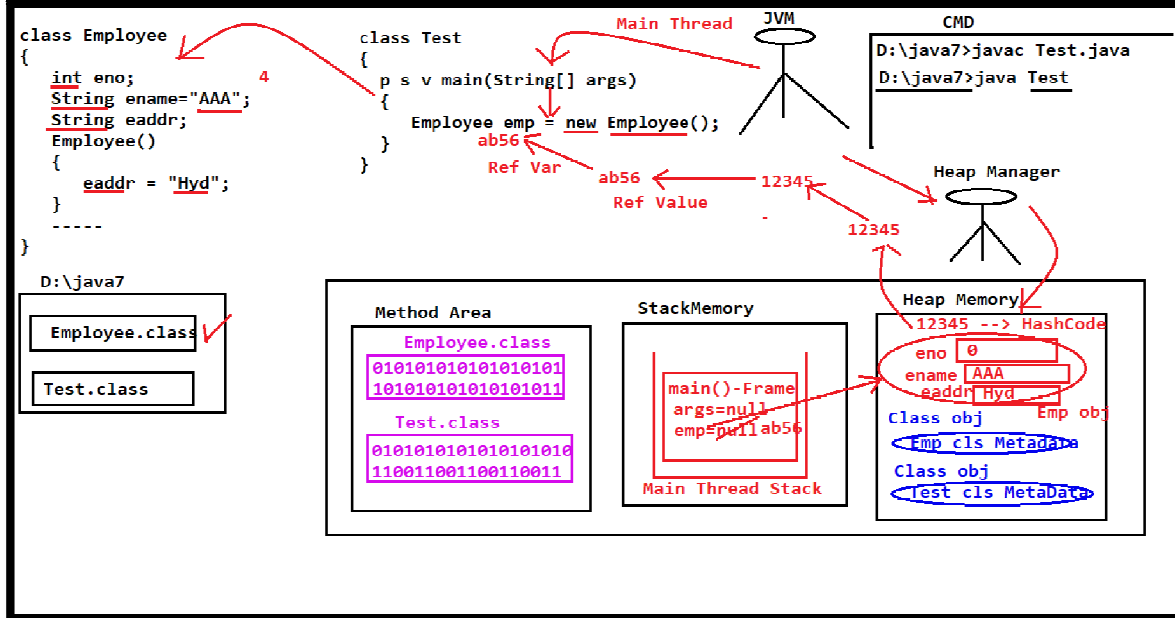
After getting Object reference value, JVM will assign that reference value to a variable called as "Reference Variable".

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 email="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  :"+email);
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)
```