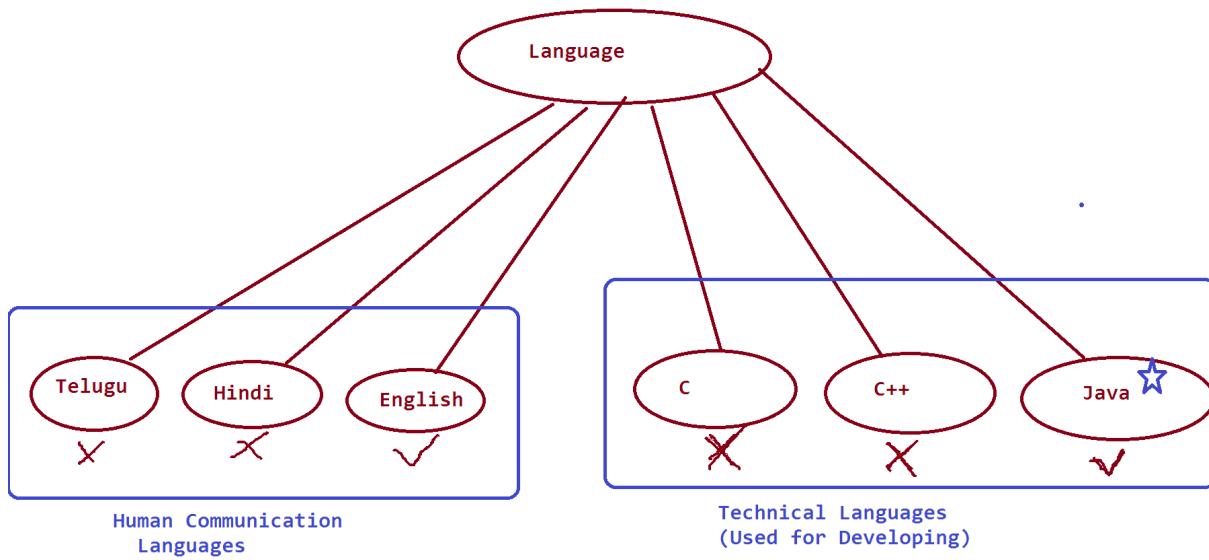


# Introduction to Java



## Process of Learning Languages:

1. Alphabets
2. Grammer/syntax
3. Construction (Program construction)

### Note:

=>Every Language will have its own Alphabets, Grammer and Construction

## Java Language:

=>From the vendor(Owner of the product),the Java Language is available in the following three parts:

1.JavaSE

2.JavaEE

3.JavaME

### 1.JavaSE:

=>JavaSE stands for "Java Standard Edition" and which is used to

**construct StandAlone Applications.**

**define StandAlone applications?**

=>The applications which are installed in one computer and performs actions in the same Computer are known as StandAlone applications.

**Note:**

=>According to developer StandAlone application means

**No HTML input**

**No Server Environment**

**No DataBase Connection**

**2.JavaEE:**

=>JavaEE stands for "Java Enterprise Edition" and which is used to develop Web Applications.

**define Web Applications?**

=>The applications which are running in Web environment or Internet environment are known as Web Applications or Internet Applications.

**Note:**

=>According to developer Web Applications means

**HTML input**

**Server Environment**

**DataBase Connection**

**3.JavaME:**

=>JavaME stands for "Java Micro Edition" and which is used to develop Embedded Applications and Mobile Applications.

**Note:**

=>JavaME is also known as "Java Machine Edition" and "Java Mobile Edition".

=>In realtime JavaME is less used when compared to JavaSE and JavaEE.

=>According to realtime requirement the Java Language is categorized into two parts:

(a)Core Java

(b)Advanced Java

**(a)Core Java:**

=>Core Java provides the fundamental Building blocks which are used in constructing Java program.These fundamental Building blocks are also known as Java Alphabets or Programming components.

**List of Programming components:**

**1.Variables**

**2.Methods**

**3.Blocks**

**4.Constructors**

**5.Classes**

**6.Interfaces**

**7.Abstract Classes.**

=>CoreJava also provides the following programming concepts:

**1.Object Oriented Programming**

**2.Exception Handling**

**3.MultiThreading**

**4.Java Collection Framework(JCF)**

## **5.GUI Programming**

**=>CoreJava also provides the following Object Oriented programming features:**

- 1.Class**
- 2.Object**
- 3.Abstraction**
- 4.Encapsulation**
- 5.PolyMorphism**
- 6.Inheritance.**

**Note:**

**=>Using CoreJava knowledge we can construct StandAlone applications.**

**(b)Advanced Java:**

**=>Advanced Java provides the following technologies to construct WebApplications:**

- (i)JDBC**
- (ii)Servlet**
- (iii)JSP**

**(i)JDBC:**

**=>JDBC stands for "Java Data Base Connection" and which is used to control DataBase Product.**

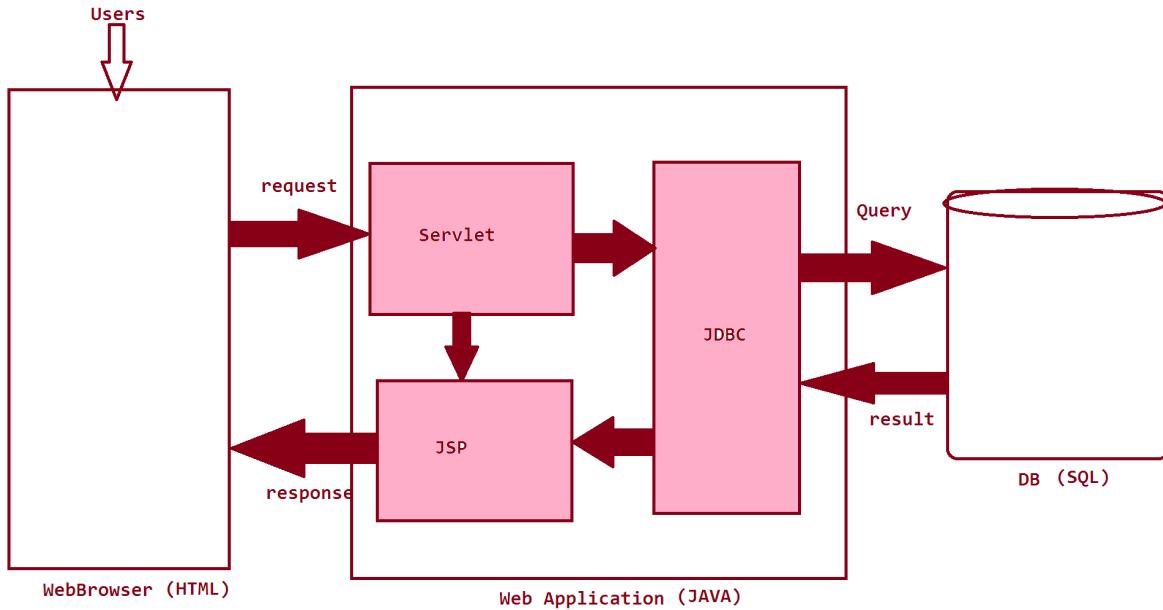
**(ii)Servlet:**

**=>Servlet means "Server program" and which is used to accept the request from the WebBrowser.**

**(iii)JSP:**

**=>JSP stands for "Java Server Page" and which is response from the**

## Web Application.



faq:

**wt is the diff b/w**

- (i)Language**
- (ii)Technology**
- (iii)Framework**

**(i)Language:**

=>"The system of Communication" is known as Language. And Language provides Alphabets, Grammer and Constructing rules.

**Note:**

=>Core Java comes under Language category.

**(ii)Technology:**

=>The process of applying the knowledge in realtime for development

is known as Technology.

Note:

=>Advanced Java comes under Technology category.

JDBC

Servlet

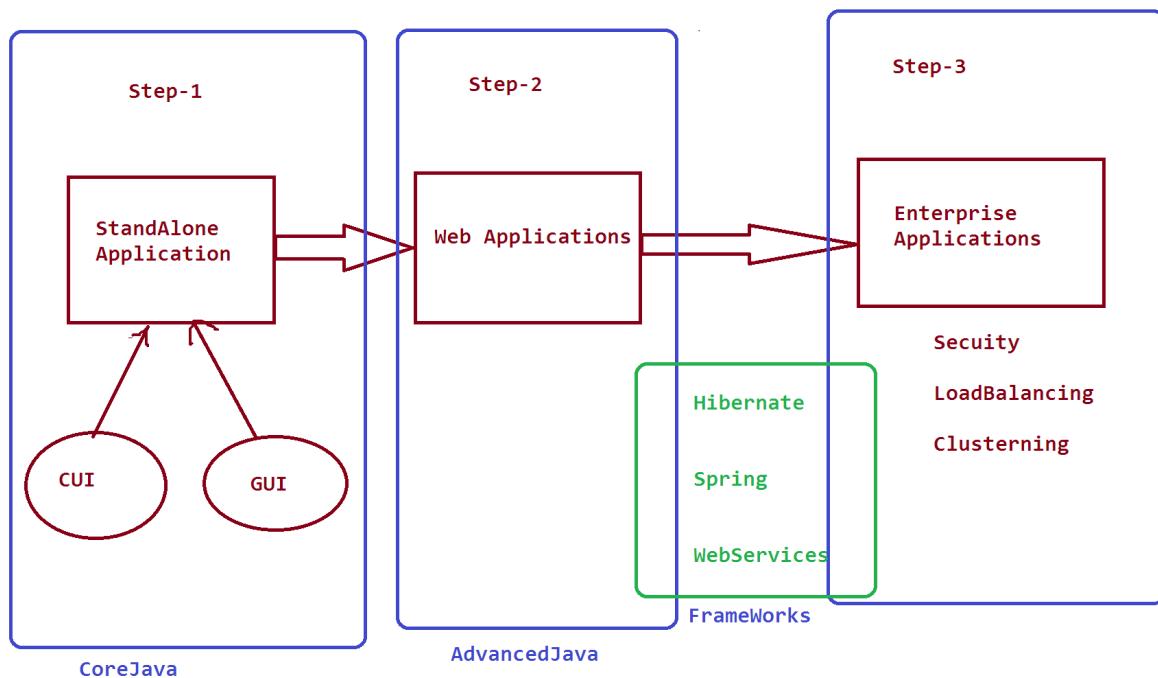
JSP

(iii)Framework:

=>The essential supporting structure which is already available for constructing applications is known as Framework.

Note:

Hibernate, Spring and WebService comes under Framework category.



---

\*imp

**define Program?**

=>Program is a "set-of-Instructions".

**Note:**

=>After writing the program, the program is saved with language extention

**Exp:**

**Test.c**

**Test.cpp**

**Test.java**

=>The program will have the following two stages:

**1.Compilation stage**

**2.Execution Stage**

**1.Compilation stage:**

=>The process of checking the program whether it is constructed within the rules of language or not, is known as Compilation process.

**Note:**

=>In Compilation process c,c++ and Java languages use compiler.

=>After Compilation process is Successfull c and c++ languages generate Objective Code and, the Java Language generate Byte Code.

**2.Execution Stage:**

=>The process of running compiled codes and checking the required output is generated or not, is known as Execution process.

=>This execution process internally having the following two

**SubStages:**

**(i) Loading process**

**(ii)Linking process**

**(i)Loading process:**

=>The process of loading the required files into current running program using "Loader" is known as Loading process.

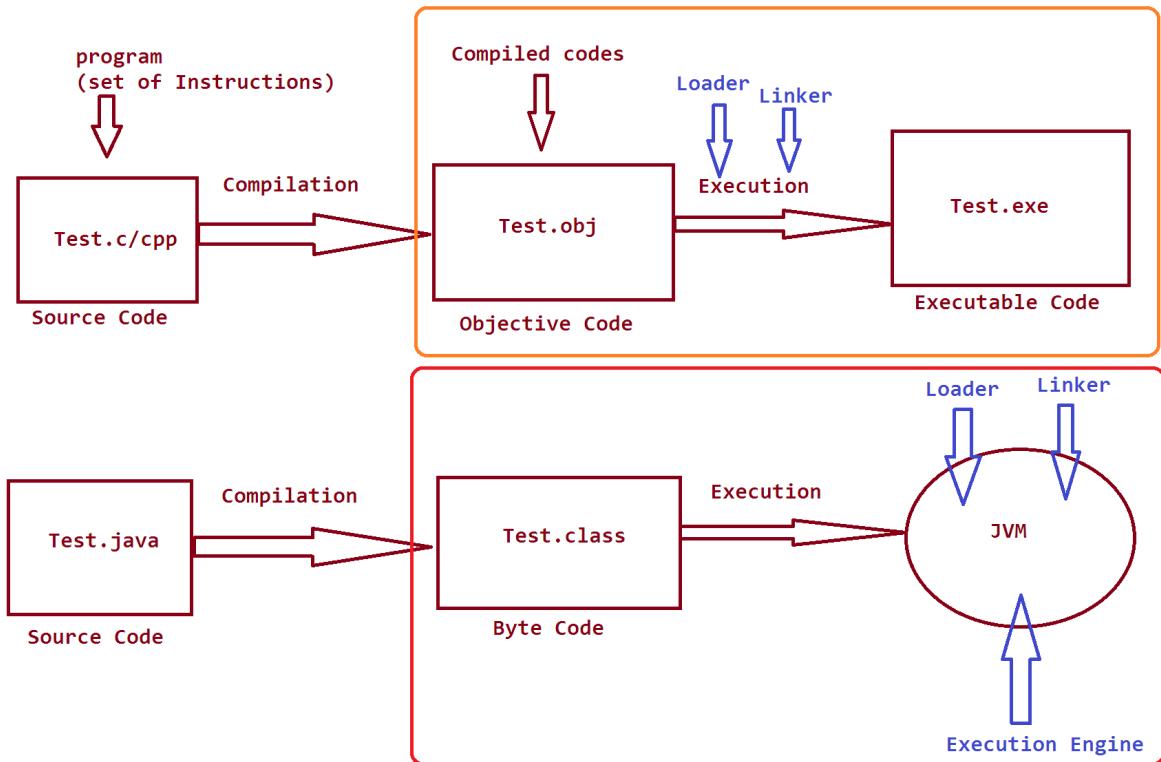
**(ii)Linking process:**

=>The process of linking the loaded files into current running program where they are needed using "Linker" is known as Linking process.

**Note:**

=>In C and C++ languages, after loading and linking process the Objective code is converted into Executable code and which is executed.

=>In Java Language the Byte code is executed on JVM (Java Virtual Machine), which internally having Loader, Linker and Execution Engine.



Dt : 2/11/2020(Regular Course Content)

faq:

wt is the diff b/w

(i)Objective Code

(ii)Byte Code

(i)Objective Code:

=>The compiled code which is generated from c and c++ languages is known as Objective code.

=>while Objective code generation OperatingSystem is participated because of this reason Objective code is PlatForm dependent code.

DisAdvantage:

=>The Objective Code which is generated from one PlatForm cannot be executed on other PlatForms.

Note:

=>The c and c++ languages which are generating objective code are PlatForm dependent Languages and these languages are not preferable to develop Internet applications.

---

(ii)Byte Code:

=>The Compiled code which is generated from Java Language is known as Byte Code.

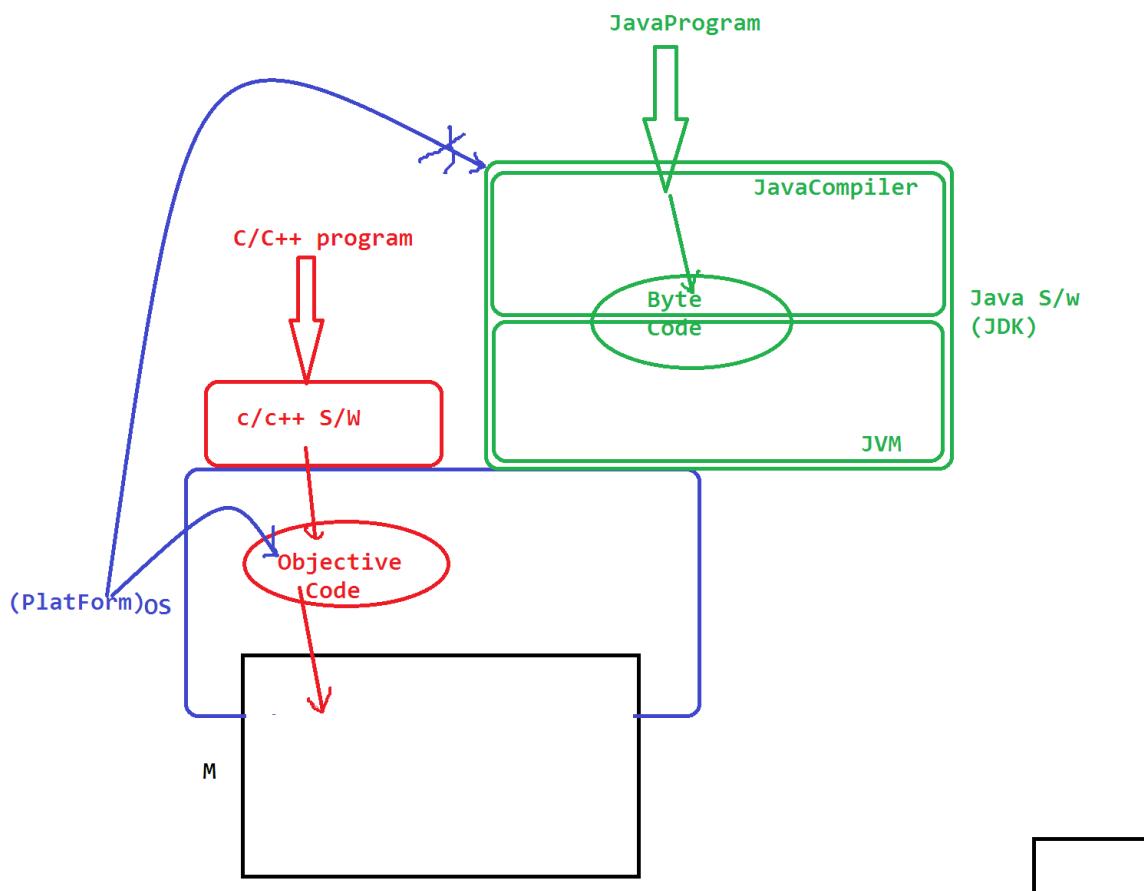
=>while Byte Code generation OperatingSystem is not participated because of this reason the ByteCode is PlatForm Independent code.

Advantage:

=>The ByteCode which is generated from one platform can be executed on all platforms.

Note:

=>The java language which is generating ByteCode is Platform Independent language and which is preferable for internet application development.



faq: define Translator?

=>Translators are used to translate the High Level Language codes into Low Level Language codes and vice-versa.

Translators are categorized into two types:

**(i)Compilers**

**(ii)Interpreters**

**(i)Compilers:**

=>Compiler will translate the total program at-a-time.

**(ii)Interpreters:**

=>Interpreter will translate the program line-by-line.

---

Dt : 3/11/2020

faq:

**wt is the diff b/w**

**(i)High Level Languages**

**(ii)Low Level Languages**

**(i)High Level Languages:**

=>The language programs which are constructed from User understandable words is known as High Level language.

**Exp:**

c,c++,Java

**(ii)Low Level Languages:**

=>The language programs which are constructed using the code not understandable by the users is known as Low level language.

**Exp:**

**Assembly language**

**Machine Language**

**define Assembler?**

=>Assembler is also a translator,which is used to translate Assembly

**language code into Machine Language code and vice versa.**

---

### **Java History:**

=> James Gosling, Mike Sheridan, and Patrick Naughton initiated the Java language project in June 1991. The small team of sun engineers called Green Team.

=> Initially designed for small, embedded systems in electronic appliances like set-top boxes.

=> Firstly, it was called "Greentalk" by James Gosling, and the file extension was .gt.

**Exp:**

**ProgName.gt**

=>The language named as "Java" and released first version by 1995.

### **Java Versions:**

**1995 - Java Alpha&Beta**

**1996 - JDK 1.0**

**1997 - JDK 1.1**

**1998 - JDK 1.2**

**2000 - JDK 1.3**

**2002 - JDK 1.4**

---

**2004 - Java5**

**=>JDK 1.5**

**=>JRE 1.5**

**2006 - Java6**

=>JDK 1.6

=>JRE 1.6

**2011 - Java7**

=>JDK 1.7

=>JRE 1.7

---

**2014 - Java8**

=>JDK 1.8

=>JRE 1.8

**2017 - Java9**

=>JDK 1.9

=>JRE 1.9

**2018,2019,2020**

->**Java10,Java11,Java12,Java13,Java14**

---

**faq: wt is the diff b/w**

**(i)JDK**

**(ii)JRE**

**(i)JDK:**

**=>JDK stands for 'Java Developer Kit' and which is having**

**JavaCompiler,JavaLibrary and JVM**

**JavaCompiler - JavaCompiler will translate SourceCode into ByteCode.**

**JavaLibrary - JavaLibrary provides Built-In components which are**

**used in constructing application.**

**JVM**      -JVM will execute Java ByteCode.

**define Java Library?**

=>Java Library is represented by a word "java".

=>JavaLib is a collection of 'packages'.

=>packages are collection of classes and Interfaces.

=>Classes and Interfaces are collection of variables and methods.

The following are some packages from JavaLib:

**java.lang**   - Language package

**java.util**   - Utility package

**java.io**   - IO and Files packages

**java.net**   - Networking package

---

**java.awt**   - Abstract Window Toolkit package(GUI)

**javax.swing** - Swing package(GUI)

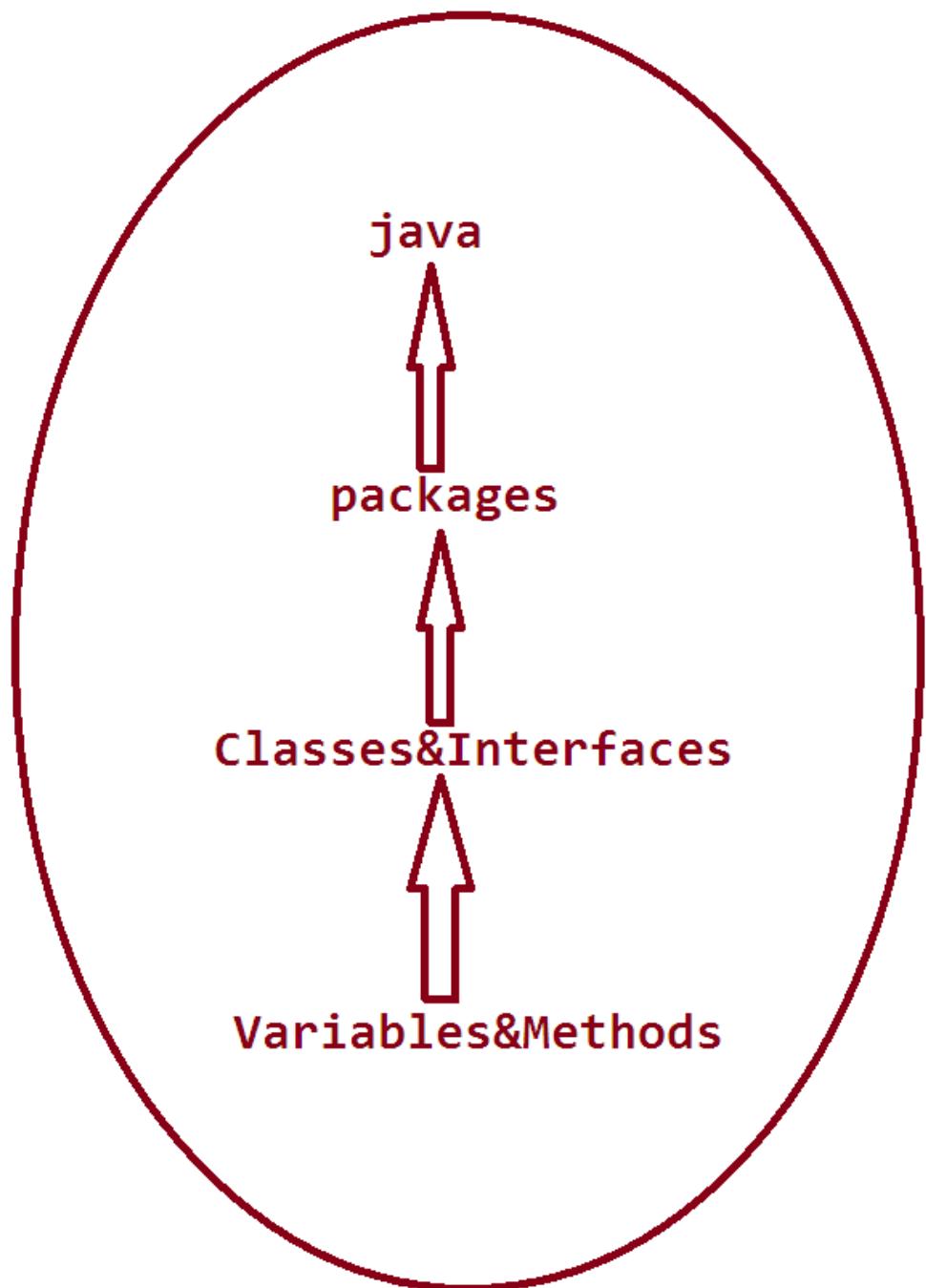
**java.applet** - Applet programming package(GUI)

---

**java.sql**   - DataBase package

**javax.servlet** - Servlet Programming package

**javax.servlet.jsp** - JSP programming package



Classes&Interfaces

Variables&Methods

java

packages

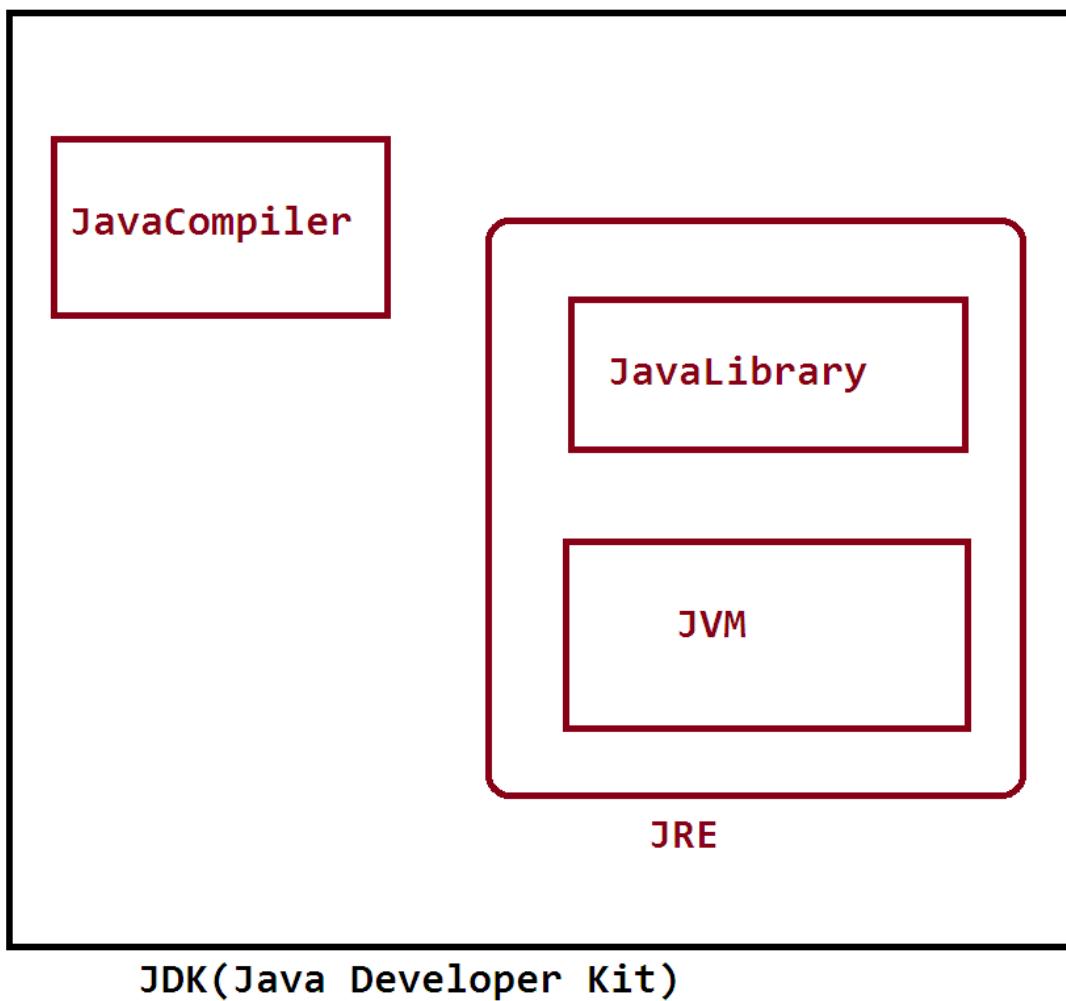
Dt : 4/11/2020

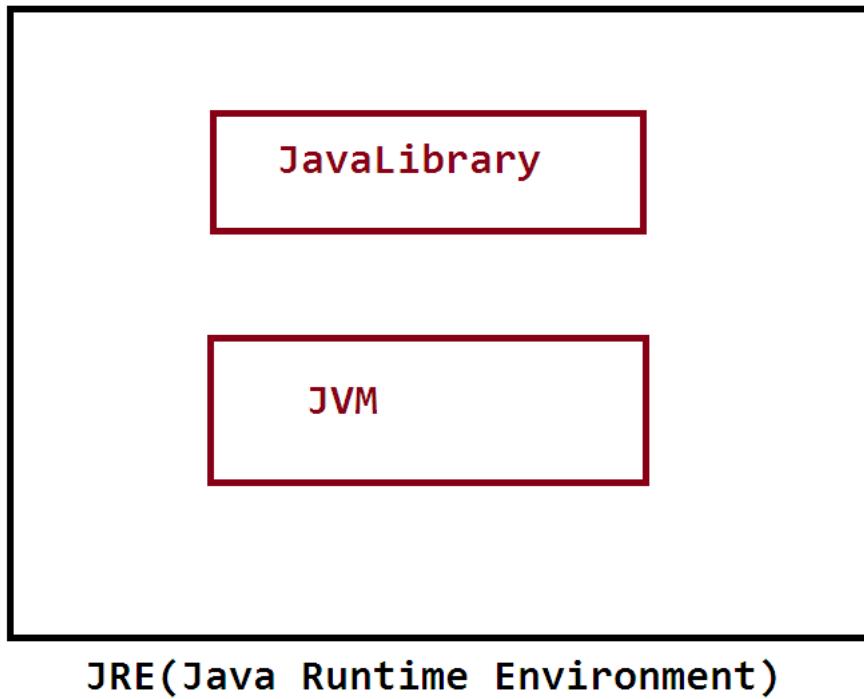
(ii)JRE:

=>JRE stands for 'Java Runtime Environment' and which is having only JavaLibrary and JVm. There is no JavaCompiler part of JRE.

Note:

=>JRE is used part of WebServers and ApplicationServers to execute WebApplications.





---

\*imp

**Installing Java S/W(JDK) and Setting path:**

**step1 : we must know the System\_type where we are installing the S/w**

**RightClick on MyComputer->Properties,then we can see System\_type**

**System type : 32-bit Operating System**

**step2 : Download JDK from Oracle WebSite**

**download JDK based on the System\_type**

**step3 :Install JDK**

=>After installation process is successfull,we can find one folder with name "java" in "programfiles"

C:\Program Files\Java

**step4 : Set JavaPath in Environment variables.**

**RightClick on MyComputer->Properties->Advanced System Settings->**

**Environment variables,click 'new' from System variables:**

**Variable name : path**

**Variable value : C:\Program Files\Java\jdk1.8.0\_251\bin;**

**step5 : Click "ok" for three times then the path is setted.**

---

**Note:**

=>Open Command prompt and check the following two commands are working or not.

**javac - this command is used for Compilation process.**

**java - this command is used for execution process**

---

Dt : 6/11/2020

### **Basic Components needed to construct program:**

=>The following are three basic components needed in constructing a program:

1.Variables    2.functions(methods)    3.main()

#### **1.Variables:**

=>Variables are the data holders and which are used to hold the data in the programs.

#### **2.functions(methods):**

=>functions are the actions which are used to generate the required result.These functions are called as methods in Java.

#### **3.main():**

=>main() is used to start the program execution.(Starting point)

\*imp define Class?

=>Class is a 'Structured Layout' and which generates Objects.

=>Class is a collection of Variables,methods(functions) and main().

=>we use 'class' keyword to declare classes.

### **Structure of class declaration:**

```
class Class_name
{
    //Variables
    //Methods
    //main()
}
```

\*imp

**define Object?**

=>Object is a memory related to a class and holding the members of class.

=>we use "new" keyword or operator to create objects.

**syntax of creating object using 'new' operator:**

**Class\_name obj\_name = new Class\_name();**

---

**Note:**

=>To construct programs in Java Language we use 'classes'.

---

**Ex program:**

**wap to add two numbers and display the result?**

**Variables : a,b,c**

**a=10**

**b=20**

**c=a+b**

**Methods : add()**

**Starting point : public static void main(String args[])**

=>This is defined standard main() method format in JavaLang to start the execution process,if not execution will not be started.

```
import java.lang.System;  
import java.lang.String;
```

```
class Addition
{
    int a,b,c;
    void add()
    {
        c=a+b;
        System.out.print(c);
    }
    public static void main(String args[])
    {
```

```
    Addition ad = new Addition();
```

```
    ad.a = 10;
```

```
    ad.b = 20;
```

```
    ad.add();
```

```
}
```

```
}
```

Dt : 7/11/2020

**Writing,Saving,Compiling and Executing Java program:**

**step1 : Create one folder(Destination folder) in any drive**

E:\OnlineData\CoreJava\Demo108

**step2 : Open any TextEditor(EditPlus)**

**step3 : Create JavaProgram**

**Click on File->new->Java and type the program**

**step4 : Save the program in Destination folder(Demo108)**

**Click on File->save->Browse the destination folder and name the file**

**(Addition.java)->click 'save'.**

**Note:**

=>Compile and execute Java program from CommandPrompt.

**To open CommandPrompt,goto destination folder->type 'cmd' in AddressBar and press 'enter'.**

**step5 : Compile the program as follows**

**syntax:       javac Class\_name.java**

**Exp:**

**javac Addition.java**

**step6 : Execute the program as follows**

**syntax:       java Class\_name**

**Exp:**

**java Addition**

=====

Dt : 9/11/2020

### JVM Architecture:(JVM Internals)

=>JVM stands for "Java Virtual Machine" and which is used to execute Java Byte Code.

=>The S/w Component which internally having the behaviour like Machine is known as Virtual Machine.

=>The following are the parts of JVM:

1.Class Loader SubSystem

2.Runtime Data Area

3.Execution Engine

#### 1.Class Loader SubSystem:

=>This Class Loader SubSystem will load the Byte Code(class) onto JVM,while loading the Class Loader SubSystem internally uses the following components:

(a)Loader                    (b)Linker                    (c)Initiate

#### (a)Loader:

=>Loader will load the required files into current running program.

#### Note:

=>According to Java Developer the required files are available in the following three locations:

=>JavaLib                    =>"ext" folder                    =>classpath

=>To get the required files from these three locations the loader internally uses the following SubLoaders:

(i)BootStrap Class Loader

**(ii)Extention Class Loader**

**(iii)Application Class Loader**

**(i)BootStrap Class Loader:**

=>BootStrap CL will load the required files from JavaLib.

**(ii)Extention Class Loader:**

=>Extention CL will load the required files from the "ext" folder.

**(iii)Application Class Loader:**

=>Application CL will load the required files from the "classpath".

**(b)Linker:**

=>The Linker will link the loaded files into current running program.

=>while Linking process the Linker internally uses the following components:

**(i)Verify**

**(ii)Prepare**

**(iii)Resolve**

**(i)Verify:**

=>Verify Component will perform verification process,in this process it check the loaded and required files are same or not.

**(ii)Prepare:**

=>Prepare Component will perform decoding process,in this process it identify the type of programming components.(Variable,method,class...)

**(iii)Resolve:**

=>This Resolve component will identify the programming components

are static or Nonstatic based on "static" keyword.

Note:

=>based on "static" keyword in java the programming components are categorized into two types:

(a)static programming components

(b)NonStatic programming components

(a)static programming components:

=>The programming components which are declared with 'static' keyword are known as Static programming components.

Note:

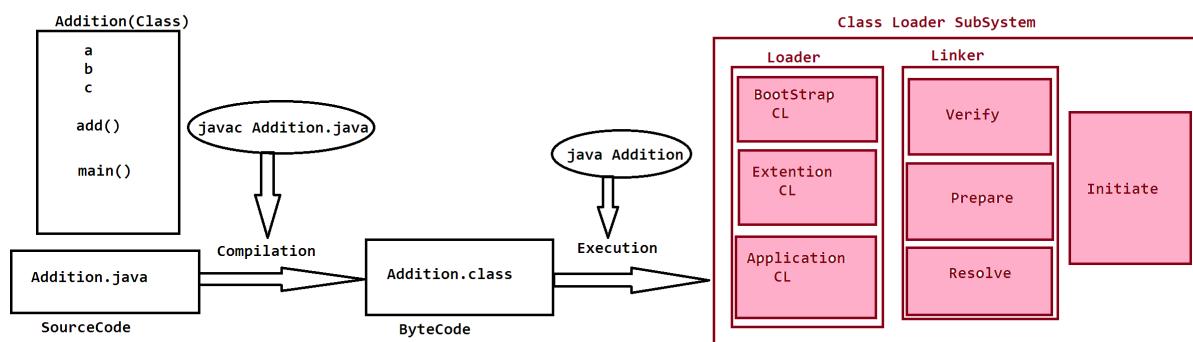
=>These static programming components will get the memory within the class while class loading and access with class\_name.

(b)NonStatic programming components:

=>The programming components which are declared without static keyword are known as NonStatic programming components.

Note:

=>These NonStatic programming components will get the memory within the object while object creation and access with object\_name.



Dt : 10/11/2020

**(c)Initiate:**

=>Initiate component will perform initialization process and in this process one memory is created known as "Runtime Data Area" and which is loaded with class.

---

**2.Runtime Data Area:**

=>This Runtime Data Area internally divided into the following partitions:

- (a)Method Area**
- (b)Heap Area**
- (c)Java Stack Area**
- (d)PC Register Area**
- (e)Native Method Area**

**(a)Method Area:**

=>The memory block where the class is loaded is known as **Method\_Area**.

=>while class loading the static programming components of the class will get the memory within the class.

=>Once main() method got the memory within the class then it is automatically copied onto Java Stack Area to start the execution process.

=>once main() method copied onto Java Stack Area then it is identified by the ExecutionEngine and starts the execution process.

**(b)Heap Area:**

=>The memory block where the objects are created is known as **Heap\_Area**.

**Behaviour of 'new' keyword:**

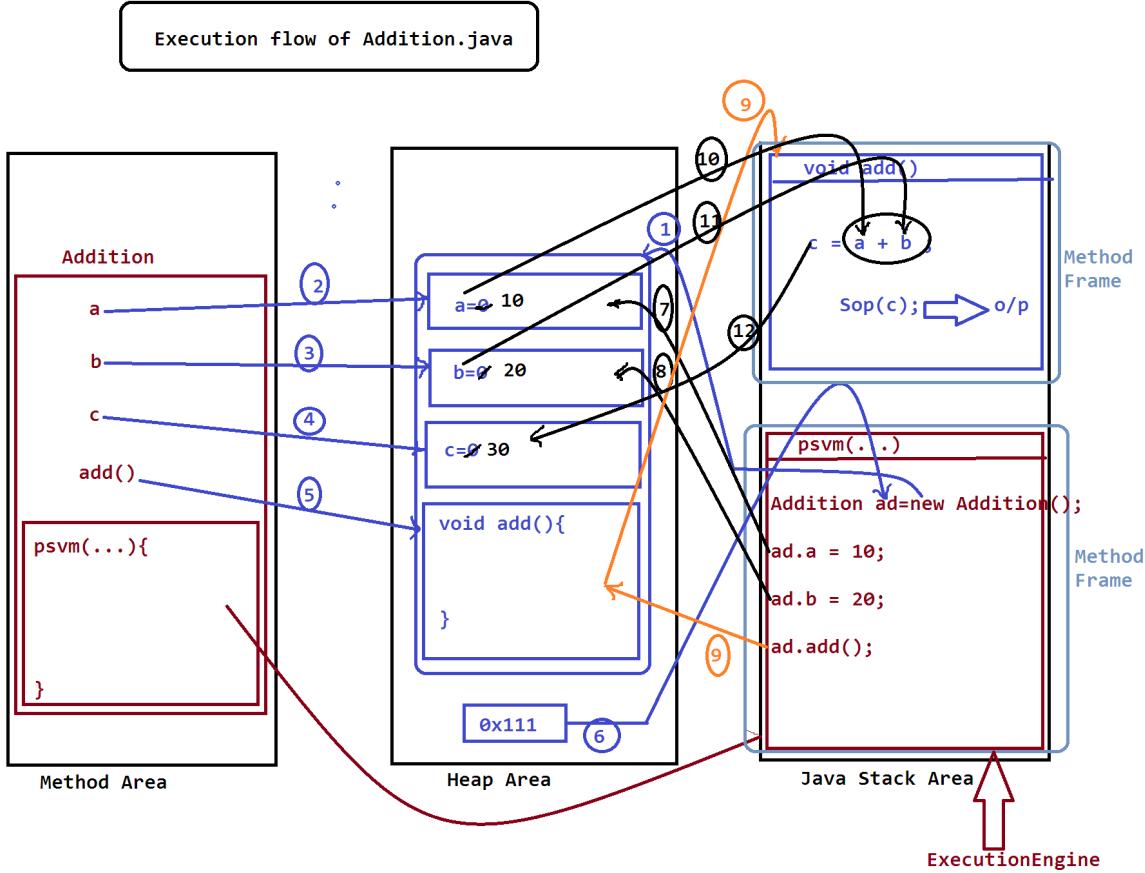
=>This 'new' keyword will specify the execution control to create one reference part of **Heap\_Area**.

=>'new' keyword specifies the execution control to check the required class is available on **Method\_Area** or not.

=>'new' keyword will specify the execution control to find the NonStatic members of the class and bind to the reference.

(bind to the reference means give memory at reference)

=>'new' keyword will specify the execution control to load the reference on to reference variable(ad) or **Object\_name**.



Dt : 11/11/2020

faq:

**define Method Frame?**

=>The partition of JavaStackArea where the method is copied for execution is known as Method Frame.

**Note:**

=>After Method execution completed,the related method\_frame will be destroyed automatically.

---

**while constructing Java Application we use one MainClass and can have any number of SubClasses.**

**MainClas - The Class which holds main() method is known as MainClass  
SubClasses \_ The Classes which are declared part of JavaAppl other than MainClass are known as SubClasses.**

**Exp program:**

**program to demonstrate Employee data.**

**EmpDetails**

=>emplId,empName,empDesg  
=>void getEmpDetails()

**EmpAddress**

=>hNo,sName,city,pinCode  
=>void getEmpAddress()

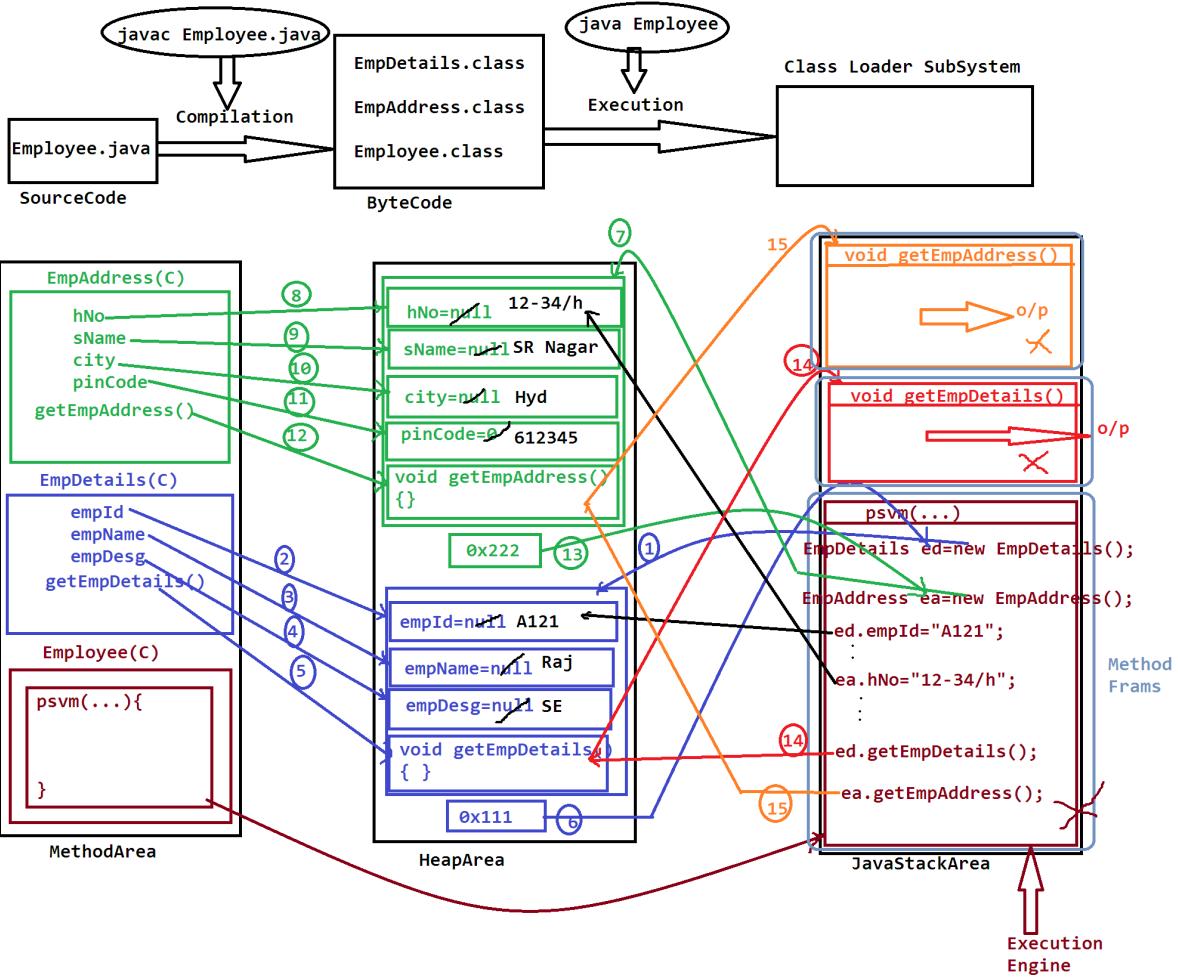
**Employee**

=>public static void main(String[] args)  
/\*program to demonstrate Employee details\*/

```
import java.lang.String;
import java.lang.System;
class EmpDetails //SubClass
{
    String empId,empName,empDesg;
    void getEmpDetails()
    {
        System.out.println("Empld="+empId);
        System.out.println("EmpName="+empName);
        System.out.println("EmpDesg="+empDesg);
    }
}
class EmpAddress //SubClass
{
    String hNo,sName,city;
    int pinCode;
    void getEmpAddress()
    {
        System.out.println("HNO="+hNo);
        System.out.println("SName="+sName);
        System.out.println("City="+city);
        System.out.println("PinCode="+pinCode);
    }
}
class Employee //MainClass
```

```
{  
    public static void main(String[] args)  
    {  
        EmpDetails ed = new EmpDetails();//Object  
        EmpAddress ea = new EmpAddress();//Object  
  
        ed.empId = "A121";  
        ed.empName = "Raj";  
        ed.empDesg = "SE";  
  
        ea.hNo = "12-34/h";  
        ea.sName = "SR Nagar";  
  
        ea.city = "Hyd";  
        ea.pinCode = 612345;  
  
        ed.getEmpDetails();//method call  
        ea.getEmpAddress();//method call  
    }  
}
```

**Execution flow of above program:**





Dt : 17/11/2020

### Naming Conventions in Java:

=>The rules which are used to construct java program are known as

### Naming Conventions in Java.

#### packages:

=>The packages must be in LowerCase.

#### Classes and Interfaces:

=>In Classes and Interfaces the starting letter of every word must be capital.

#### Exp:

Employee

EmpAddress

StudentAddress

#### Variables and Methods:

=>In Variables and methods the first word must be in lowercase and from second word onwards the starting letter must be capital.

#### ExP;

empld

rollNo

panCard

getEmpDetails()

readLine()

upperCase()

#### keywords:

=>The Built-In words from the JavaLib are known as **Keywords** or **Built-In words**.

=>These **Keywords** must be in **LowerCase**.

Exp:

void

int

import

static

=====

\*imp

**DataTypes in Java;**

=>The types of data which we are expecting as input to java program are known as **DataTypes in java**.

=>**DataTypes in Java** are categorized into two types:

1.Primitive DataTypes

2.NonPrimitive DataTypes

**1.Primitive DataTypes:**

=>The datatypes which are existing in the form of 'single valued data format' are known as **Primitive datatypes**.

=>**Primitive DataTypes** are categorized into the following:

(a)Integer datatypes

(b)Float datatypes

(c)Character datatype

(d)Boolean datatype

**(a)Integer datatypes:**

=>The numeric data without decimal point representation are known  
**Integer datatypes.**

**Types:**

**byte** - 1byte(8bits)

**short** - 2bytes

**int** - 4bytes

**long** - 8bytes

**(b)Float datatypes:**

=>The numeric data with decimal point representation are known as  
**Float datatypes.**

**Types:**

**float** - 4bytes

**double** - 8bytes

**(c)Character datatype:**

=>The 'single valued character' which is represented with single  
quotes is known as **Character datatype.**

**Types:**

**char** - 2bytes

**(d)Boolean datatype:**

=>The datatype which is existing in the form of true or false is  
known as **Boolean datatype.**

**Types:**

**boolean** - 1bit

**Exp Program:**

```
/*program to demonstrate Primitive DataTypes...*/
```

```
import java.lang.String;  
import java.lang.System;  
  
class DataTypes  
{  
    public static void main(String[] args)  
    {  
        byte by = 127;  
        short sh = 32767;  
        int i = 327689;  
        long l = 9898989898L;  
        float f = 12.34F;  
        double d = 6756453456.786789;  
        char ch = 'A';  
        boolean b = true;  
        String name = "Raj";
```

```
System.out.println("by:"+by);  
System.out.println("sh:"+sh);  
System.out.println("i:"+i);  
System.out.println("l:"+l);  
System.out.println("f:"+f);  
System.out.println("d:"+d);  
System.out.println("ch:"+ch);  
System.out.println("b:"+b);
```

```
System.out.println("name:"+name);  
}  
}
```

**Note:**

=>while assigning long value we use "L" or "l" on the RHS of declaration.

=>while assigning float value we use 'F' or "f" in the RHS of Declaration.

---

## 2.NonPrimitive DataTypes:

=>The datatypes which are existing in the form of 'group valued data format' are known as NonPrimitive datatypes or referential datatypes.

=>NonPrimitive DataTypes are categorized into the following:

- (a)Class
- (b)Interface
- (c)Array
- (d)Enum

**faq:**

**define "String"?**

=>The sequenced collection of characters which are represented in double quotes is known as String.

**Exp:**

"nit","raj",...

=>"String" is a class in Java and comes under NonPrimitive datatypes

but which can be used like primitive datatype.

---

Dt : 18/11/2020

**Operators in Java:**

=>Operator is a special symbol or keyword which performs operations.

The following are some important operators used in Java lang:

**1.Arithmetic Operators**

**2.Relational Operators**

**3.Logical Operators**

**4.Increment-Decrement operators**

**1.Arithmetic Operators**

=>The operators which perform basic operations are known as

Arithmetic operators.

**Operator      Meaning**

**+**      Addition

**-**      Subtraction

**\***      Multiplication

**/**      Division

**%**      Modulo Division

**2.Relational Operators:**

=>The operators which are used to perform Comparisons are known as Relational Operators.

**Operator      Meaning**

**>**      Greater Than

**>=** Greater Than or Equal to

**<** Less Than

**<=** Less Than or Equal to

**==** Is Equal to

**!=** Not Equal to

### **3. Logical Operators:**

**=>**The Operators which compare two comparisons are known as  
**Logical Operators.**

#### **Operator    Meaning**

**&&** Logical AND

**||** Logical OR

**!** Logical NOT

#### **Logical AND(&&):**

A   B   A&&B

T   T   T

F   T   F

T   F   F

F   F   F

#### **Logical OR(||):**

A   B   A||B

T   T   T

F   T   T

T   F   T

F   F   F

**Logical NOT(!):**

A !A

T F

F T

#### **4.Increment-Decrement operators:**

=>The Operators which increment the value by 1 or decrement the value by 1 are known as Increment-Decrement operators.

**Operator      Meaning**

**++      Increment**

**--      Decrement**

---

#### **Control Statements in Java:**

=>The statements which are used to control the part of the program are known as Control Structures.

=>These Control Structures are categorized into the following:

**1.Selection Statements**

**2.Iterative Statements**

**3.Branching Statements**

#### **1.Selection Statements:**

=>The statements which are used to select one part of the program on some condition are known as Selection statements or Control Statements.

#### **List of Selection Statements:**

**(a)Simple if**

**(b)if-else**

(c)Nested if

(d)Ladder if-else

(e)switch-case

## 2.Iterative Statements:

=>The statements which are used to execute some lines of the program repeatedly on some condition are known as Iterative statement.

(Looping Structures)

### List of Iterative Statements:

(a)while

(b)do-while

(c)for

## 3.Branching Statements:

=>The statements which are used to transfer the execution control from one location to another location are known as Branching Statements or Transfer Statements.

### List of Branching Statements:

(a)break

(b)continue

(c)return

(d)exit

=====

\*imp

## Object Oriented Programming:

=>The process of constructing application using class-object concept

is known as Object Oriented programming.

=>In Object Oriented programming we control the following NonPrimitive

DataTypes:

- (a)Class
- (b)Interface
- (c)Array
- (d)Enum

(a)Class:

=>Class is a 'Structured layout' and which generates Objects.

=>Class is a collection of Variables and methods.

=>Class is loaded on Method\_Area of JVM.

=>In JavaLang these classes are categorized into two types:

- (i)User defined Classes
- (ii)Built-In classes

(i)User defined Classes:

=>The classes which are defined by the programmer are known as

User defined classes.

Exp:

above programs

    Addition.java

    Employee.java

---

(ii)Built-In classes:

=>The classes which are available from JavaLib are known as

**Buil-In classes.**

**Exp:**

**System**

**String**

**Scanner**

...

---

**"Scanner" class:**

**=>"Scanner" class will provide methods which are used to read data into Java program.**

**=>"Scanner" is available from "util" package.**

**=>"util" package is available from JavaLib.**

**=>The following are some important methods from "Scanner" class:**

**nextByte()**

**nextShort()**

**nextInt()**

**nextLong()**

**nextFloat()**

**nextDouble()**

**nextBoolean()**

**nextLine()**

**Method Signatures:**

**public byte nextByte();**

**public short nextShort();**

```
public int nextInt();
public long nextLong();
public float nextFloat();
public double nextDouble();
public boolean nextBoolean();
public String nextLine();
```

**syntax of Object creation for Scanner class:**

```
Scanner s = new Scanner(System.in);
```

**Exp program:**

**Assignment1:(Solution)**

**wap to read and display Product details?**

**Product**

```
=>pCode,pName,pPrice,pQty
```

```
=>void getProduct()
```

**MainClass1**

```
=>public static void main(String[] args)
```

```
/*Program to read and display Product details*/
```

```
import java.lang.String;
import java.lang.System;
import java.util.Scanner;
class Product //SubClass
{
    String pCode,pName;
```

```
float pPrice;
int pQty;
void getProduct()
{
System.out.println("ProdCode:"+pCode);
System.out.println("ProdName:"+pName);
System.out.println("ProdPrice:"+pPrice);
System.out.println("ProdQty:"+pQty);
}
class MainClass1 //MainClass
{
    public static void main(String[] args)
    {
Scanner s = new Scanner(System.in);//Built-Class object
Product p = new Product();//User defined class object
System.out.println("Enter the PCode:");
p.pCode = s.nextLine();
System.out.println("Enter the PName:");
p.pName = s.nextLine();
System.out.println("Enter the PPrice:");
p.pPrice = s.nextFloat();
System.out.println("Enter the PQty:");
p.pQty = s.nextInt();
p.getProduct();//Method call
```

```
}
```

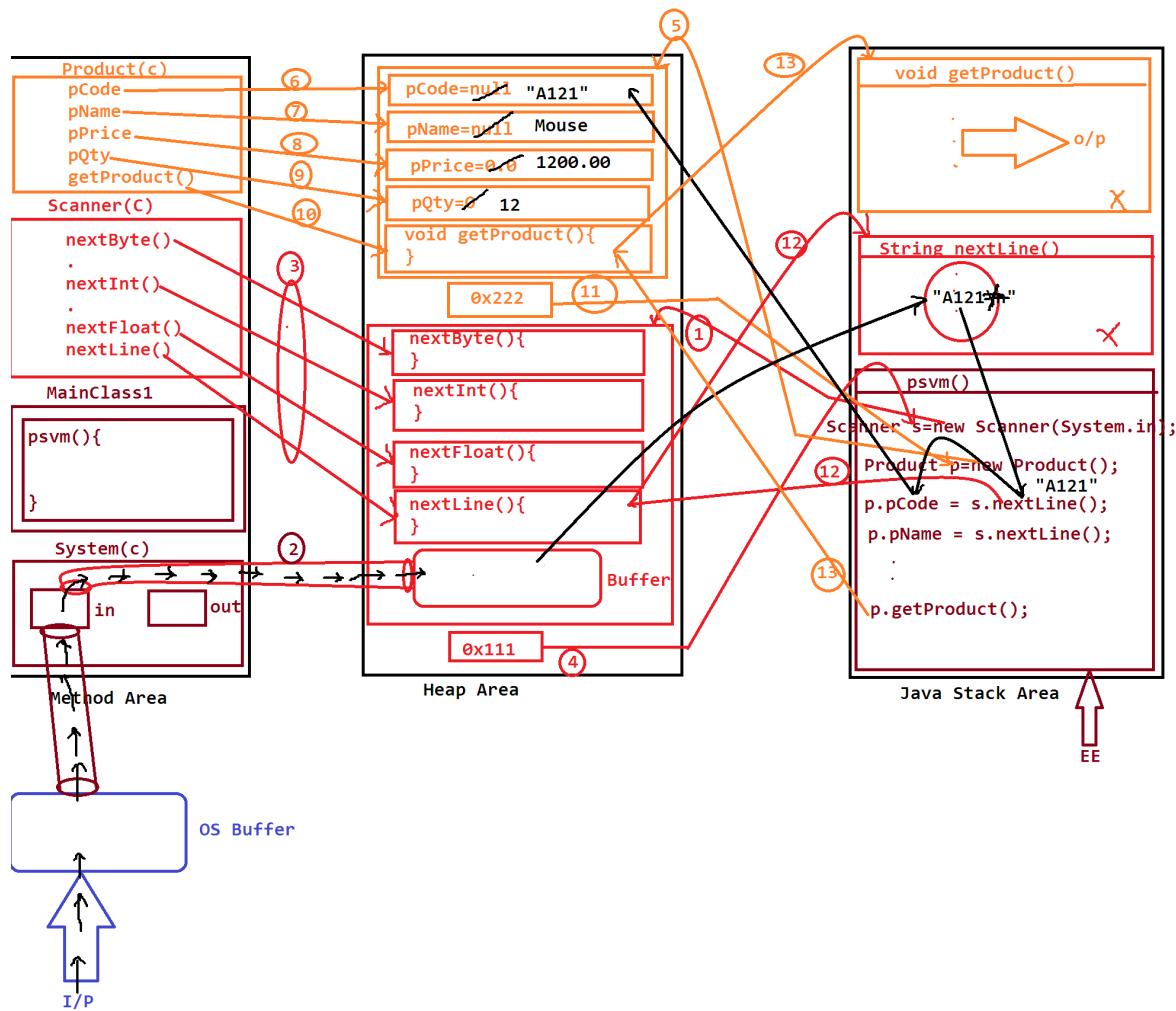
Dt : 19/11/2020

Execution flow of above program:

**ClassFiles:**

**Product.class**

**MainClass1.class**



**Assignment2:(Use Scanner Class)**

wap to read and display Book details?

**BookData**

=>bCode,bName,bAuthor,bPrice,bQty

=>void getBookData()

### **BMainClass**

=>public static void main(String[] args)

### **Assignment3:(Use Scanner class)**

#### **StuDetails**

=>rollNo,stuName,stuBr

=>void getStuDetails()

#### **StuAddress**

=>hNo,sName,city,pinCode

=>void getStuAddress()

### **SMainClass**

=>public static void main(String[] args)

---



Dt : 20/11/2020

Exp program:

WAP to read user details and display the same?

UserDetails

-userName

-phoneNo

-mailId

```
import java.lang.String;
import java.lang.System;
import java.lang.Long;
import java.util.Scanner;
class UserDetails //SubClass
{
    String userName,mailId;
    long phoneNo;
    void getUserDetails()
    {
        System.out.println("UserName:"+userName);
        System.out.println("PhoneNo:"+phoneNo);
        System.out.println("MailId:"+mailId);
    }
}
class MainClass2 //MainClass
{
```

```
public static void main(String[] args)
{
Scanner s = new Scanner(System.in);//Built-In class object
UserDetails ud = new UserDetails();//User defined class object
System.out.println("Enter the UserName:");
ud.userName = s.nextLine();
System.out.println("Enter the phoneNo:");
ud.phoneNo = Long.parseLong(s.nextLine());
System.out.println("Enter the MailId:");
```

```
ud.mailId = s.nextLine();
```

```
ud.getUserDetails();//Method call
```

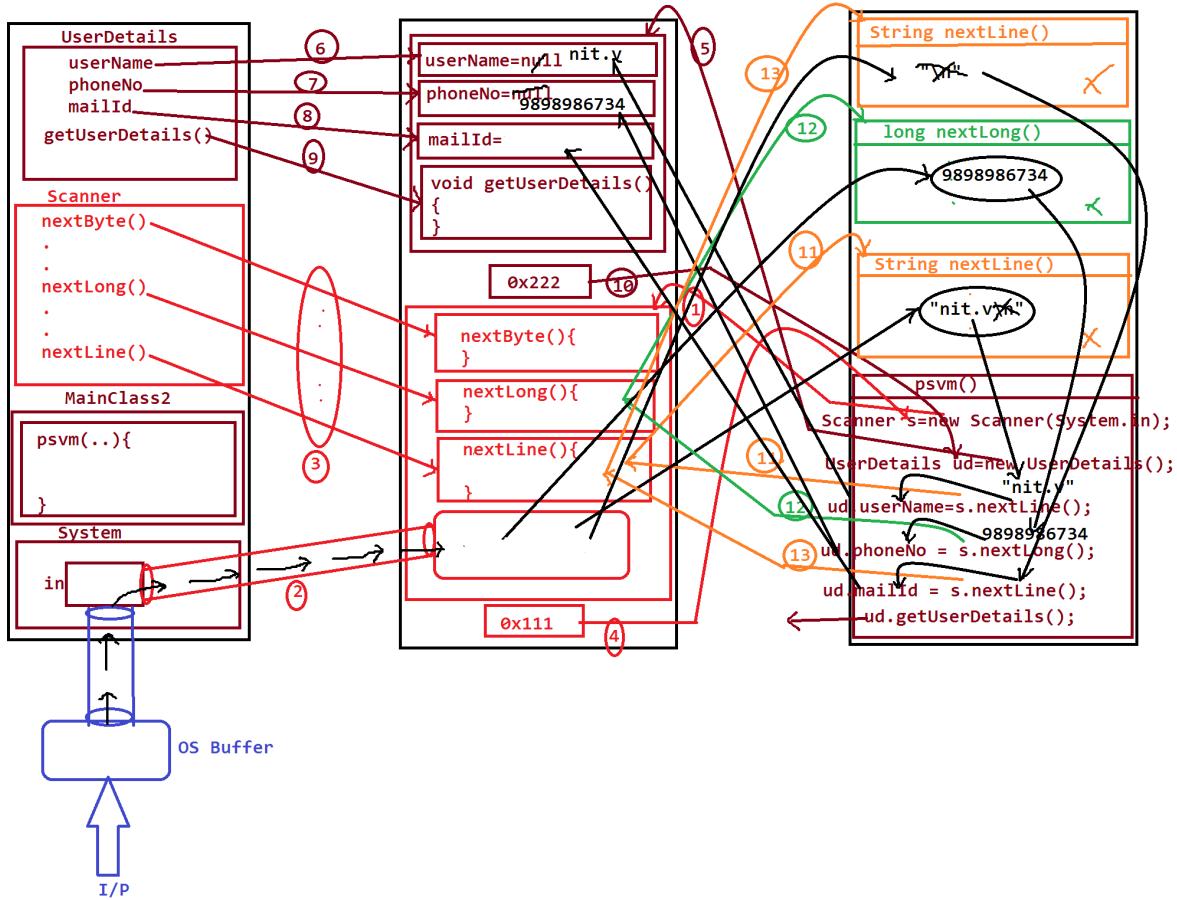
```
}
```

**Execution flow of above program:**

**ClassFiles:**

**UserDetails.class**

**MainClass2.class**



#### Note:

=>when we read String data after reading numeric data,then the reading of String is skipped because "\n" is available in the buffer.

=>This can be overcomed using parseXXXX() methods,which are used to convert String data to numeric data.

#### syntax:

```
byte b = Byte.parseByte(s.nextLine());
```

```
short s1 =Short.parseShort(s.nextLine());
```

```
int i = Integer.parseInt(s.nextLine());
```

```
long l = Long.parseLong(s.nextLine());
```

```
float f = Flaot.parseFloat(s.nextLine());  
double d = Double.parseDouble(s.nextLine());  
=>Byte,Short,Integer,Long,Float,Double are wrapper Classes and which  
provide pareseXXXX() method
```

---

Dt : 23/11/2020

**Assignment2:(Solution)**

wap to read and display Book details?

**BookData**

```
=>bCode,bName,bAuthor,bPrice,bQty  
=>void getBookData()
```

**MainClass3**

```
=>public static void main(String[] args)  
/*program to read and display Book details...*/
```

```
import java.lang.String;  
import java.lang.System;  
import java.util.Scanner;  
class BookData //SubClass  
{  
    String bCode,bName,bAuthor;  
    float bPrice;  
    int bQty;  
    void getBookData()  
    {  
        System.out.println("BookCode:"+bCode);
```

```
System.out.println("BookName:"+bName);
System.out.println("BookAuthor:"+bAuthor);
System.out.println("BookPrice:"+bPrice);
System.out.println("BookQty:"+bQty);
}

}

class MainClass3 //MainClass
{
    public static void main(String[] args)
    {
Scanner s = new Scanner(System.in);
BookData bd = new BookData();
System.out.println("Enter the BookCode:");
bd.bCode = s.nextLine();
System.out.println("Enter the BookName:");
bd.bName = s.nextLine();

System.out.println("Enter the BookAuthor:");
bd.bAuthor = s.nextLine();
System.out.println("Enter the BookPrice:");
bd.bPrice = s.nextFloat();
System.out.println("Enter the BookQty:");
bd.bQty = s.nextInt();
bd.getBookData();//Method call
    }
}
```

### **Assignment3:(Solution)**

#### **StuDetails**

**=>rollNo,stuName,stuBr**

**=>void getStuDetails()**

#### **StuAddress**

**=>hNo,sName,city,pinCode**

**=>void getStuAddress()**

#### **MainClass4**

**=>public static void main(String[] args)**

**/\*program to read and display Student details...\*/**

```
import java.util.Scanner;

class StuDetails //SubClass

{
    String rollNo,stuName,stuBr;

    void getStuDetails()

    {
        System.out.println("RollNo:"+rollNo);
        System.out.println("StuName:"+stuName);
        System.out.println("StuBranch:"+stuBr);
    }
}
```

```
}
```

```
class StuAddress //SubClass

{
```

```
String hNo,sName,city;

int pinCode;

void getStuAddress()

{

System.out.println("HNO:"+hNo);

System.out.println("SName:"+sName);

System.out.println("City:"+city);

System.out.println("PinCode:"+pinCode);

}

}

class MainClass4 //MainClass

{

    public static void main(String[] args)

    {

Scanner s = new Scanner(System.in);

StuDetails sd = new StuDetails();

StuAddress sa = new StuAddress();

System.out.println("Enter the RollNo:");

sd.rollNo = s.nextLine();

System.out.println("Enter the StuName:");

sd.stuName = s.nextLine();

System.out.println("Enter the Branch:");

sd.stuBr = s.nextLine();

System.out.println("Enter the HNO:");

```

```
sa.hNo = s.nextLine();

System.out.println("Enter the sName:");

sa.sName = s.nextLine();

System.out.println("Enter the city:");

sa.city = s.nextLine();

System.out.println("Enter the PinCode:");

sa.pinCode = s.nextInt();

sd.getStuDetails();

sa.getStuAddress();

}

}
```

**Note:**

=>Draw the Execution flow for above programs

---

**Exp program:**

wap to read two int values and perform arithmetic operation based on  
used choice?

**/\*program to perform arithmetic operation based on user choice\*/**

```
import java.util.Scanner;

class Addition //SubClass
```

```
{  
    int add(int x,int y)  
    {  
        return x+y;  
    }  
}  
  
class Subtraction //SubClass  
{  
    int sub(int x,int y)  
    {  
        return x-y;  
    }  
}  
  
class Multiplication //SubClass  
{  
    int mul(int x,int y)  
    {  
        return x*y;  
    }  
}  
  
class Division //SubClass  
{  
    float div(int x,int y)  
    {  
        return (float)x/y;//TypeCasting
```

```
    }
}

class ModDivision //SubClass

{
    int modDiv(int x,int y)
    {
        return x%y;
    }
}

class MainClass5 //MainClass

{
    public static void main(String[] args)
    {
Scanner s = new Scanner(System.in);
System.out.println("Enter the value1:");
int v1 = s.nextInt();
System.out.println("Enter the value2:");
int v2 = s.nextInt();
System.out.println("====Choice====");
System.out.println("1.ads\n2.sub\n3.mul\n4.div\n5.modDiv");
System.out.println("Enter the Choice:");
int choice = s.nextInt();
switch(choice)
{
}
```

```
    case 1:  
  
Addition ad = new Addition();  
  
System.out.println("sum:"+ad.add(v1,v2));  
        break;  
  
    case 2:  
  
Subtraction sb = new Subtraction();  
  
System.out.println("Sub:"+sb.sub(v1,v2));  
        break;  
  
    case 3:  
  
Multiplication ml = new Multiplication();  
  
System.out.println("Mul:"+ml.mul(v1,v2));  
        break;  
  
    case 4:  
  
Division dv = new Division();  
  
System.out.println("div:"+dv.div(v1,v2));  
        break;  
  
    case 5:  
  
ModDivision md = new ModDivision();  
  
System.out.println("modDiv:"+md.modDiv(v1,v2));  
        break;  
  
    default:  
  
System.out.println("InValid Choice...");  
}  
//end of switch  
}
```

=====

Dt : 24/11/2020

### **Variables in Java:**

=>Variables are the data holders and which are used to hold the data in the programs.

=>Based on the datatypes the variables are categorized into two types:

**1.Primitive DataType variables**

**2.NonPrimitive DataType Variables**

#### **1.Primitive DataType variables:**

=>The variables which are declared as byte,short,int,long,float, double,char or boolean are known as Primitive DataType variables.

=>These primitive datatype variables will hold values.

#### **2.NonPrimitive DataType Variables:**

=>The variables which are declared as class,Interface,Array or Enum are known as NonPrimitive DataTypes or Referential datatypes.

=>These NonPrimitive DataType variables will hold references or addresses.

---

Based on static keyword the variables are categorized into two types:

**(a)static Variables**

**(b)NonStatic Variables**

**(a)static Variables:**

**def:**

=>The variables which are declared with static keyword are known as Static variables or Class variables.

**Memory Location:**

=>These Static variables will get the memory within the class while class loading and can be accessed with the class\_name.

**Scope and Visibility:**

=>These Static variables are accessed by both static and NonStatic methods.

**LifeTime:**

=>These static variables are available until the class is available on Method\_area.

**Note:**

=>If the static variables are not assigned with any values then they are assigned with default values.

---

**(b)NonStatic Variables:**

=>The variables which are declared without static keyword are known as NonStatic variables.

=>These NonStatic variables are categorized into two types:

(i)Instance variables

(ii)Local Variables

**(i)Instance variables:**

**def:**

=>The NonStatic variables which are declared outside the methods are known as Instance variables.

**Memory Location:**

=>These Instance variables will get the memory within the Object while object creation and can be accessed with Object\_name.

**Scope and Visibility:**

=>These Instance variables can be accessed by the methods available in the same object.

**LifeTime:**

=>These Instance variables are available until the objects are available on Heap\_Area.

**Note:**

=>If Instance variables are not assigned with any values they are assigned with default values.

---

**(ii)Local Variables:**

**def:**

**=>The NonStatic variables which are declared inside the methods are known as Local Variables.**

**Memory Location:**

**=>These Local variables will get the memory within the methodFrame while method execution.**

**Scope and Visibility:**

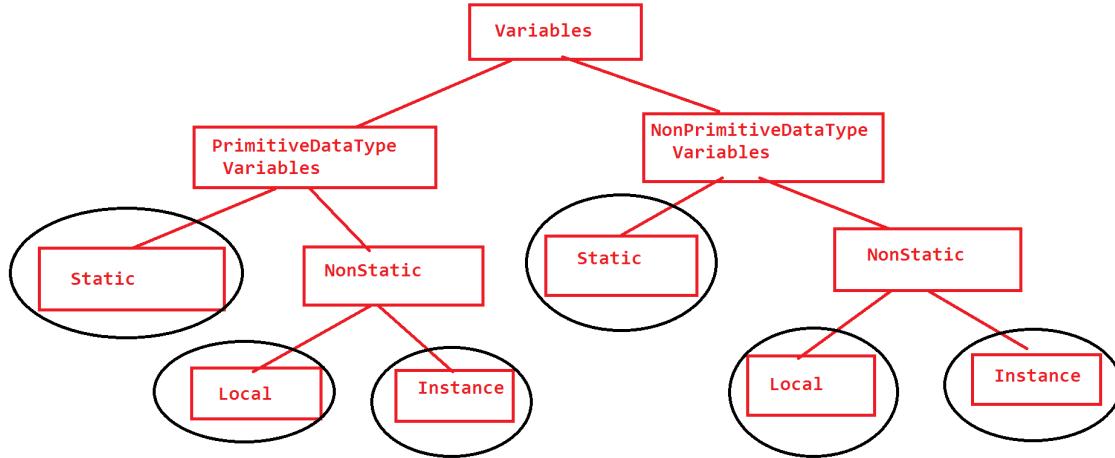
**=>These local variables are accessed only inside the methods.**

**LifeTime:**

**=>Local variables are available until the method\_frame is available.  
(Method frame is destroyed automatically when the method execution is completed)**

**Note:**

**=>Local variables will not get default values.**



\*imp

### Methods in Java:

=>Methods are the actions which are performed to generate results.

=>Based on the static keyword the methods are categorized into two types:

1.Static methods

2.NonStatic methods(Instance methods)

### 1.Static methods:

=>The methods which are declared with static keyword are known as static methods.

=>These methods will get the memory within the class while class loading and access with the Class\_name.

**Coding Rule:**

=>These Static methods can access static variables directly but cannot access instance variables.

=>These static methods are categorized into two types:

- (i)Built-In Static methods
- (ii)User defined Static methods

(i)Built-In Static methods:

=>The static methods which are available from JavaLib Classes and Interfaces are known Built-In static methods.

(ii)User defined Static methods:

=>The static methods which are defined by the programmer are known as User defined Static methods.

**structure of User defined static method:**

```
static return_type method_name(para_list)
{
    //method_body
}
```

---

**2.NonStatic methods(Instance methods):**

=>The which are declared without static keyword are known as

**NonStatic methods or Instance methods**

=>These methods will get the memory within the object while Object creation and access with object\_name.

**Coding Rule:**

=>These Instance methods can access both Static variables and Instance variables.

=>These Instance methods are categorized into two types:

(i)Built-In instance methods

(ii)User defined Instance methods

**(i)Built-In instance methods:**

=>The instance methods which are available from JavaLib Classes and Interfaces are known as Built-In Instance methods.

**EXP**

'Scanner' class methods

`nextInt()`

`nextFloat()`

...

**Note:**

=>we use "javap" command in CommandPrompt to view all the built-in methods of Classes and Interfaces.

**syntax:**

**javap java.pack\_name.CName/IName**

**Exp:**

**javap java.util.Scanner**

**(ii)User defined Instance methods:**

**=>The Instance methods which are defined by the programmer are known as User defined Instance methods.**

**structure of User defined Instance methods:**

```
return_type method_name(para_list)
{
    //method_body
}
```

---

**faq:**

**define parameters?**

**=>Parameters are the variables which are used to transfer the data**

from one method to another method.

=>Based on Parameters, the methods are categorized into two types:

(a)Method without parameters

(b)Method with parameters.

(a)Method without parameters:

=>The methods which are declared without parameters are known as 0-parameter methods or Methods without parameters.

Ex:

MainClass1, MainClass2, MainClass3, MainClass4

(b)Method with parameters.:

=>The methods which are declared with parameters are known as Parameterized methods or methods with parameters.

Exp:

MainClass5

---

faq:

define return\_type of method?

=>return\_type specify the methods will return the value after method execution or not.

=>Based on return\_Type the methods are categorized into two types:

(a)Non Return\_type methods

### **(b)Return\_type methods**

#### **(a)Non Return\_type methods:**

=>The methods which do not return any value after method execution are known as Non Return\_type methods.

**Ex:**

**MainClass1,MainClass2,MainClass3,MainClass4**

#### **Note:**

=>The methods which are declared with 'void' are known as Non return\_type methods.

#### **(b)Return\_type methods:**

=>The methods which return the value after method execution are known as Return\_type methods.

**Exp:**

**MainClass5**

#### **Note:**

=>In return\_type we use 'return' statement to return the values, the returned values will come back to the method\_call

Dt : 25/11/2020

\*imp

**Class generating multiple objects:**

=>In Java Lang the class can generate any number of objects without restriction.

=>The Multiple objects which are generated from the class are independent by their memory location on heap area.

=>The modification which is done in one object will not effect other objects.

---

**Exp program:**

**/\*program to demonstrate variables,methods and multiple Objects\*/**

```
class Display //SubClass
{
    static int a=10;//static variable
    int b=20;//Instance variable
    static void m1()
    {
        int c=30;//Local variable
        a++;
        //b++;//Compilation Error
    System.out.println("==>m1()===");
    System.out.println("The value a:"+a);
    //System.out.println("The value b:"+b);//Compilation Error
    System.out.println("The value c:"+c);
```

```
}

void m2()
{
    int c=40;//Local variable
    a++;
    b++;

System.out.println("==m2()==");
System.out.println("The value a:"+a);
System.out.println("The value b:"+b);
System.out.println("The value c:"+c);

}
}

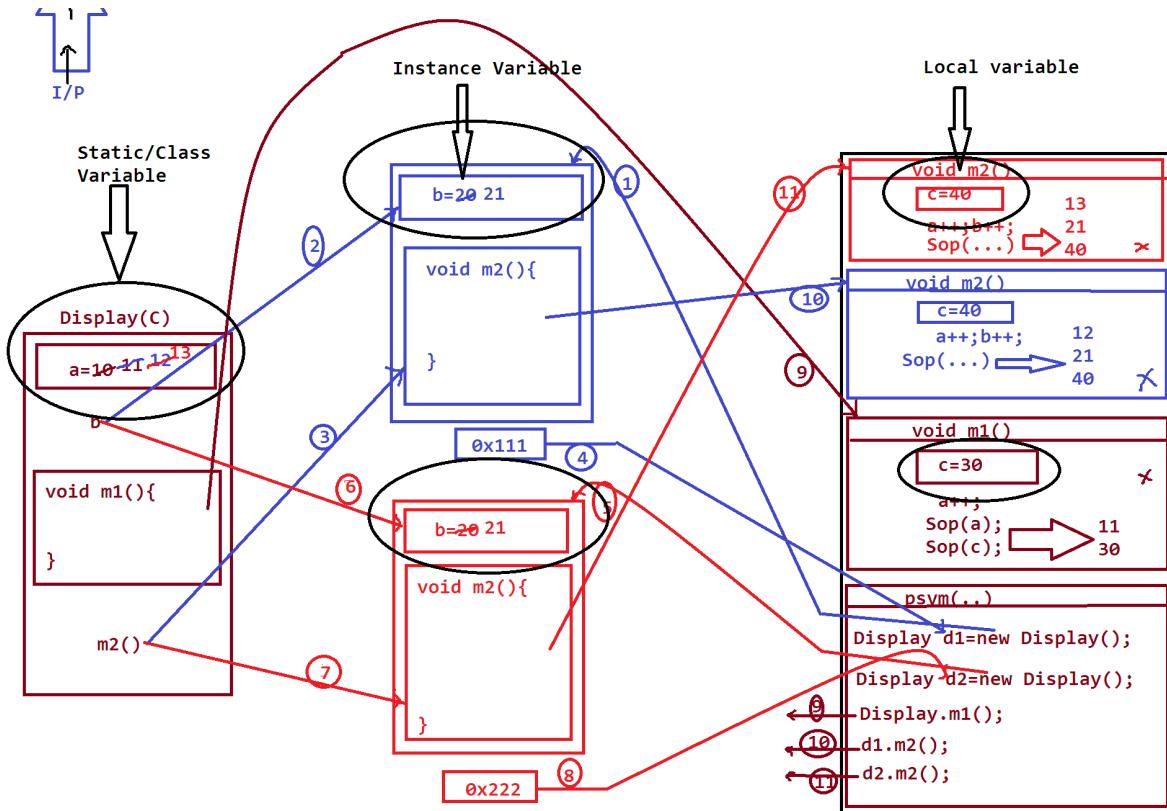
class MainClass6 //MainClass
{
    public static void main(String[] args)
    {
        Display d1 = new Display();//Object1
        Display d2 = new Display();//Object2
        Display.m1();//Static method call
        d1.m2();//Instance method call
        d2.m2();//Instance method call
    }
}
```

**Execution flow of above program:**

**ClassFiles:**

## Display.class

## MainClass6.class



**Note:**

=>The class loads only once onto method\_area in the process of generating multiple objects.

**Assignment:**

wap to read six sub marks and calculate totMarks,per and result?

**I/P:**

**s1=**

**s2=**

**s3=**

**s4=**

**s5=**

**s6=**

**Calculate:**

**totM=**

**per=**

**result=**

**per b/w 70 and 100 =>Distinction**

**per b/w 60 and 70 =>First class**

**per b/w 50 and 60 =>Second class**

**per b/w 35 and 50 = Third Class**

**Else fail**

**O/p: totMarks=**

**per=**

**Result=**

-----

Dt : 26/11/2020

Execution flow of above program:(MainClass5)

ClassFiles:

Addition.class

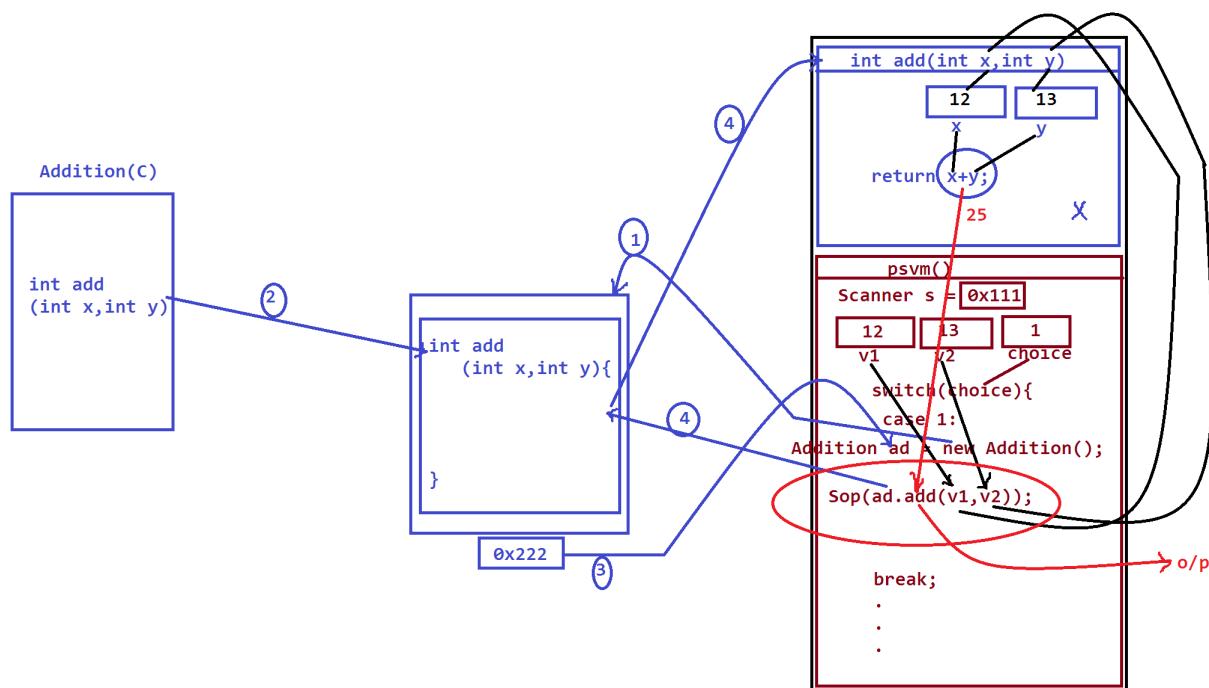
Subtraction.class

Multiplication.class

Division.class

ModDivision.class

MainClass5.class(MainClass)



Note:

=>`v1, v2, x` and `y` are known as parameters because they participated in transferring the data from one method to another method.

=>v1 and v2 will hold original data and passed as parameters while method\_call,are known as 'Actual parameters'.

=>x and y are declared part of method signature and hold intermediate data,are known as 'Formal Parameters'.

=>we can have same parameter names in Actual parameters and Formal parameters.

=>In the above program only one subClass will be loaded based on its requirement in object creation.

---

**define switch-case statement?**

=>switch-case statement is used when we want to select one from multiple available options or cases.

**Structure:**

**switch(value)**

**{**

**case 1:...**

**break;**

**case 2:...**

**break;**

**.**

**.**

**.**

**default:**

**...**

**}**

**Note:**

=>The switch-value is compared with the available options or cases, if the switch-value is matched with any option or case then it is executed and stops the execution using 'break'.

=>If the switch-value is not matched with any available options then the 'default' is executed.

---

**define 'break'?**

=>'break' is a branching statement and which is used to stop the execution of switch-case statement.

**define 'return'?**

=>'return' also known as branching statement and which is used to transfer the data to method call after method execution.

---

**Assignment:(Solution)**

wap to read six sub marks and calculate totMarks,per and result?

**I/P:**

s1=

s2=

s3=

s4=

s5=

s6=

**Calculate:**

**totM=**

**per=**

**result=**

**per b/w 70 and 100 =>Distinction**

**per b/w 60 and 70 =>First class**

**per b/w 50 and 60 =>Second class**

**per b/w 35 and 50 = Third Class**

**Else fail**

**O/p: totMarks=**

**per=**

**Result=**

**Exp program:**

**/\*Program to calculate Student result\*/**

```
import java.util.Scanner;  
  
class BranchCheck//SubClass  
{  
    boolean k;  
    boolean verify(String br)  
    {  
        switch(br)  
        {  
            case "CSE":k=true;
```

```
        break;

        case "EEE":k=true;

        break;

        case "ECE":k=true;

        break;

    default:k=false;

}

}//end of switch

return k;

}

}

class RollNoValidate //SubClass

{

    String branch;

    boolean z;

    boolean verify(String br,String code)

    {

        switch(code)

        {

            case "05":branch="CSE";

            break;

            case "02":branch="EEE";

            break;

            case "04":branch="ECE";

            break;

        }

    }//end of switch
```

```
if(branch!=null)
{
    if(br.equals(branch))
    {
        z=true;
    }
    return z;
}

}

class SResult //SubClass
{
    float per;
    String result;
    void cal(int p,int totM)
    {
        per=(float)totM/6;
        if(p==1)
        {
            result="Fail";
        }
        else if(per>=70 && per<=100)
        {
            result="Distinction";
        }
    }
}
```

```
else if(per>=60 && per<70)
{
    result="FirstClass";
}

else if(per>=50 && per<60)
{
    result="SecondClass";
}

else if(per>=35 && per<50)
{
    result="ThirdClass";
}

void getResult()
{
System.out.println("Per:"+per+"\nResult:"+result);
}

}

class MainClass7 //MainClass
{
    public static void main(String[] args)
    {
Scanner s = new Scanner(System.in);
System.out.println("Enter the stuName:");
String name = s.nextLine();
```

```
System.out.println("Enter the branch:");

String br = s.nextLine().toUpperCase();

BranchCheck bc = new BranchCheck();

boolean k = bc.verify(br);

if(k)

{

System.out.println("Enter the rollNO:");

String rollNo = s.nextLine();

if(rollNo.length()==10)

{

RollNoValidate rnv = new RollNoValidate();

boolean z = rnv.verify(br,rollNo.substring(6,8));

if(z)

{

int p=0,totM=0,i=1;

System.out.println("====Enter six sub marks====");

while(i<=6){

System.out.println("Enter the sub"+i);

int sub = Integer.parseInt(s.nextLine());

i++;

if(sub<0 || sub>100)

{

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

i--;

continue;
}
}
}
}
}
```

```
    }

if(sub>=0 && sub<=34)

{
    p=1;

}

totM=totM+sub;

}//end of loop

System.out.println("StuName:"+name);

System.out.println("Branch:"+br);

System.out.println("RollNo:"+rollNo);

System.out.println("TotM:"+totM);

SResult sr = new SResult();

sr.cal(p,totM);

sr.getResult();

}

//end of if

else

{

System.out.println("RollNo not matched with the branch...");

}

//end of if

else

{

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

}
```

```
//end of if  
else  
{  
System.out.println("InValid branch...");  
}  
}  
}
```

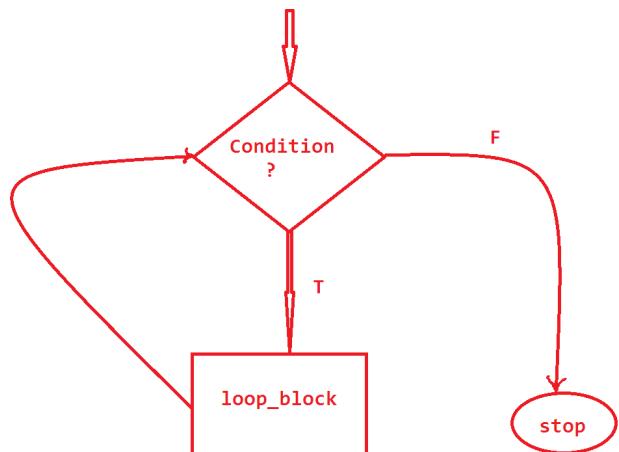
Dt : 27/11/2020

**define while loop?**

=>In while looping structure the condition is checked first,if the condition is true then the loop block is executed.This process is repeated untill the condition is false.

**syntax:**

```
while(condition)
{
    //loop_block
}
```



**wap to display the numbers from 1 to 10?**

**start value : 1**

**End value : 10**

```
/*program to display 1 to 10*/
```

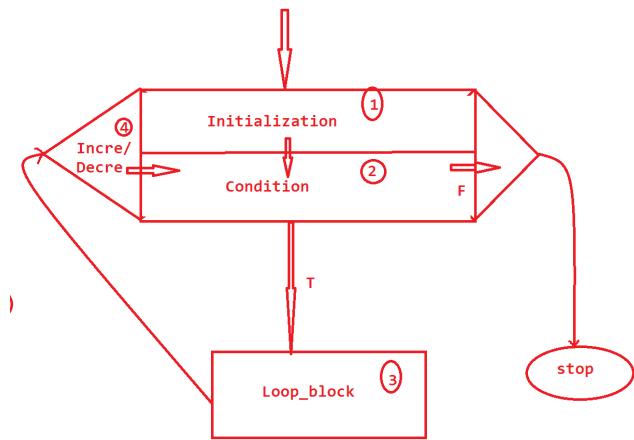
```
class DLoop1
{
    public static void main(String[] args)
    {
        int i=1;//Initialization
        while(i<=10)//Condition
        {
            System.out.print(i+" ");
            i++;//Increment
        }
    }
}
```

**define 'for' loop?**

=>In 'for' looping structure the Initialization,Condition and  
Incre/Decre declared in the same line and which is more simple in  
representation.

**syntax:**

```
for(Initialization;Condition;Incre/Decre)
{
    //Loop_block
}
```



wap to display the numbers from 1 to 10 using for loop?

**start value : 1**

**End value : 10**

**/\*program to display 1 to 10\*/**

```

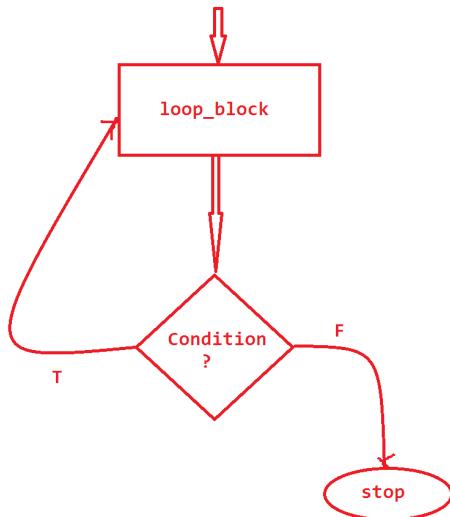
class DLoop2
{
    public static void main(String[] args)
    {
        for(int i=1;i<=10;i++)
        {
            System.out.print(i+" ");
        }
    }
}
  
```

**define do-while loop?**

=>In do-while looping Structure the **loop\_block** is executed first and then the condition is checked.If the condition is true the process is repeated untill the condition is false.

**syntax:**

```
do
{
    //loop_block
}
while(condition);
```



```
/*program to display 1 to 10*/
```

```
class DLoop3
```

```
{  
    public static void main(String[] args)  
    {  
        int i=1;  
        do  
        {  
            System.out.print(i+" ");  
            i++;  
        }  
        while (i<=10);  
    }  
}
```

**Note:**

=>while loop is more efficient when compared to do-while loop and we use while loop in realtime.

=>while loop is known as exit loop and do-while loop is known as' entry loop.

---

Dt: 28/11/2020

**Execution flow of above program:(MainClass7.java)**

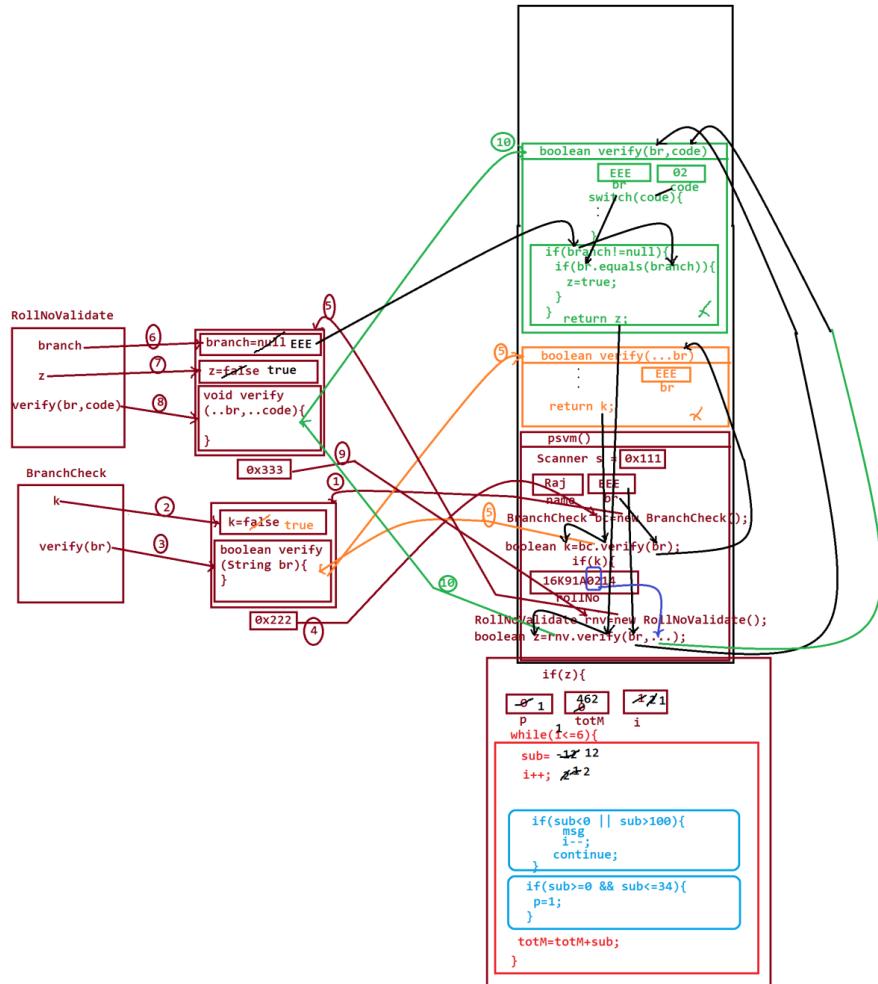
**ClassFiles:**

**BranchCheck.class**

**RollNoValidate.class**

**SResult.class**

## MainClass7.class



Dt : 30/11/2020

faq:

wt is the diff b/w

(i)static method

(ii)Instance method

=>static method will get the memory within the class and Instance

method will get the memory within the object.

=>static methods can access only static variables but instance methods

can access both static variables and Instance variables.

---

**Blocks in Java:**

def:

=>The set of statements which are declared with flower brackets({})

is known as block and which is executed automatically.

These blocks are categorized into two types:

1.static blocks

2.Instance blocks(NonStatic blocks)

**1.static blocks:**

=>The blocks which are declared with 'static' keyword are known as static blocks.

**syntax:**

**static**

```
{  
    //set-of-statements;  
}
```

**Execution behaviour of 'static' block:**

=>static blocks are executed while class loading.

=>static block executes only once because class loads only once.

**Exp program:**

```
/*program to demonstrate static block*/
```

```
class DStatic //SubClass  
{  
    static  
    {  
        System.out.println("==SubClass Static block==");  
    }  
}
```

```
class MainClass8 //MainClass  
{
```

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

```
    {
```

```
        DStatic ob = new DStatic();
```

```
    }
```

```
}
```

**Note:**

=>In realtime static block is used to hold DB Connection code part

of DAO(Data Access Object) Layer in MVC(Model View Controller).

## 2.Instance blocks(NonStatic blocks):

=>The blocks which are declared with out static keyword are known as Instance blocks or NonStatic blocks.

**syntax:**

```
{  
//set-of-statements;  
}
```

**Execution behaviour of 'NonStatic' block:**

=>NonStatic blocks are executed while object creation and these NonStatic blocks are executed for all the Multiple object creations.

**Exp program:**

```
/*program to demonstrate NonStatic block*/
```

```
class DStatic //SubClass  
{  
    static  
    {  
  
        System.out.println("==SubClass Static block==");  
    }  
}
```

```
{  
System.out.println("==SubClass NonStatic block==");  
}  
void dis()  
{  
System.out.println("==display==");  
}  
}  
class MainClass9 //MainClass  
{  
    public static void main(String[] args)  
    {
```

```
DStatic ob1 = new DStatic();
```

```
ob1.dis();
```

```
DStatic ob2 = new DStatic();
```

```
DStatic ob3 = new DStatic();
```

```
DStatic ob4 = new DStatic();
```

```
DStatic ob5 = new DStatic();
```

```
}
```

```
}
```

**Note:**

=>In realtime NonStatic block is less used when compared to static  
block.

---

**faq:**

**wt is the diff b/w**

**(i)method**

**(ii)block**

**=>Methods are executed on method\_call, but blocks are executed without call.**

**=>Blocks will have highest priority in execution than methods.**

**Note:**

**=>static blocks will have highest priority in execution than static methods.**

**=>Instance blocks will have highest priority in execution than Instance methods.**

---

**\*imp**

**Constructors in Java:**

**=>Constructor is a method having the same name of the class and executed while object creation because the constructor call is available within the object creation syntax.**

**Note:**

**=>while declaring constructor we must not use return\_type because the constructors will have class\_return\_type by default.**

**Based on the parameters constructors are categorized into two types:**

**1.Constructors without parameters**

**2.Constructors with parameters**

**1.Constructors without parameters:**

=>The constructors which are declared without parameters are known as **0-parameter constructors or Constructors without parameters.**

**Exp program:**

**/\*program to demonstrate 0-parameter constructor\*/**

```
class Test1
{
    Test1()//Constructor
    {
        System.out.println("==constructor==");
    }
    void dis()//Instance method
    {
        System.out.println("==Instance method==");
    }
}
class DCon1 //MainClass
{
    public static void main(String[] args)
```

```
{  
Test1 ob = new Test1();  
ob.dis();  
}  
}
```

## 2. Constructors with parameters:

=>The constructors which are declared with parameters are known as  
**Parameterized constructors or Constructors with parameters.**

### Exp program:

```
/*program to demonstrate parameterized constructor*/
```

```
class Test2  
{  
    Test2(int x)//Constructor  
    {  
        System.out.println("==constructor==");  
        System.out.println("The value x:"+x);  
    }  
    void dis(int y)//Instance method  
    {  
        System.out.println("==Instance method==");  
        System.out.println("The value y:"+y);  
    }  
}  
class DCon2 //MainClass
```

```
{  
    public static void main(String[] args)  
    {  
        Test2 ob = new Test2(101);//con call  
        ob.dis(102);//method call  
    }  
}
```

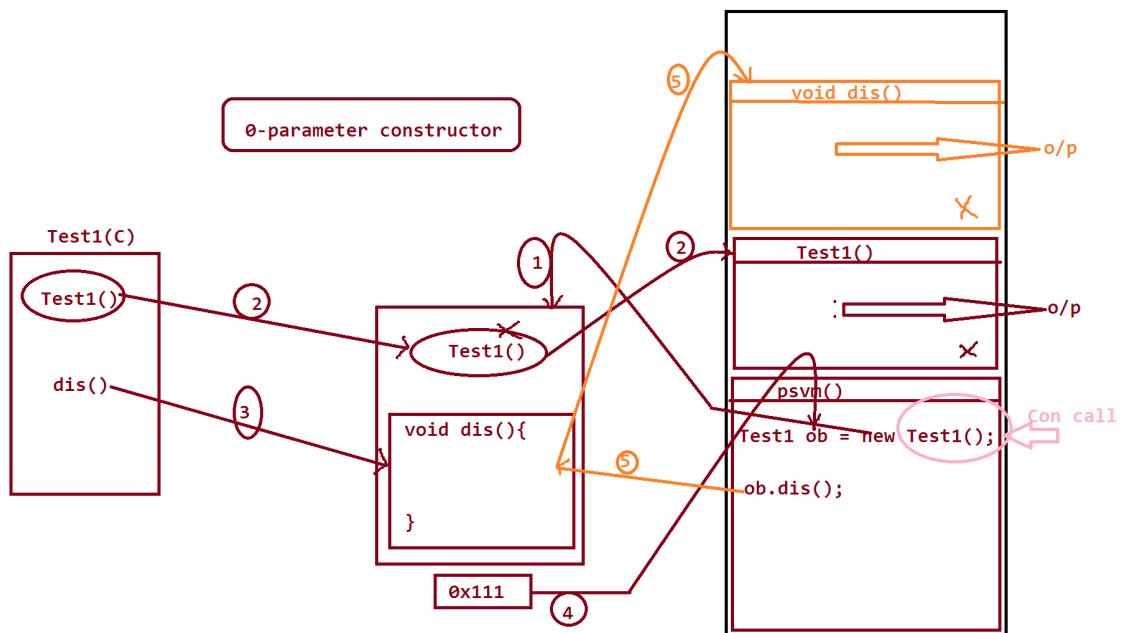
Dt : 1/12/2020

Execution flow of above program:(DCon1.java)

ClassFiles:

Test1.class

DCon1.class



faq:

wt is the diff b/w

(i)Constructor

(ii)Instance method

=>Constructor is executed while object creation, but Instance method is executed after object creation.

=>Constructor is executed only once while object creation, but the instance method can be called for execution any number of times after

**object creation.**

**faq:**

**wt is diff b/w**

**(i)Instance block**

**(ii)Construtor**

**=>Both components are executed while object creation,but Instance block will have highest priority in execution than Constructor,because constructor comes under method category.**

**faq:**

**wt is the diff b/w**

**(i)static block**

**(ii)Constructor**

**=>Both components are executed only once,but static block is executed while class loading and Constructor is executed while object creation.**

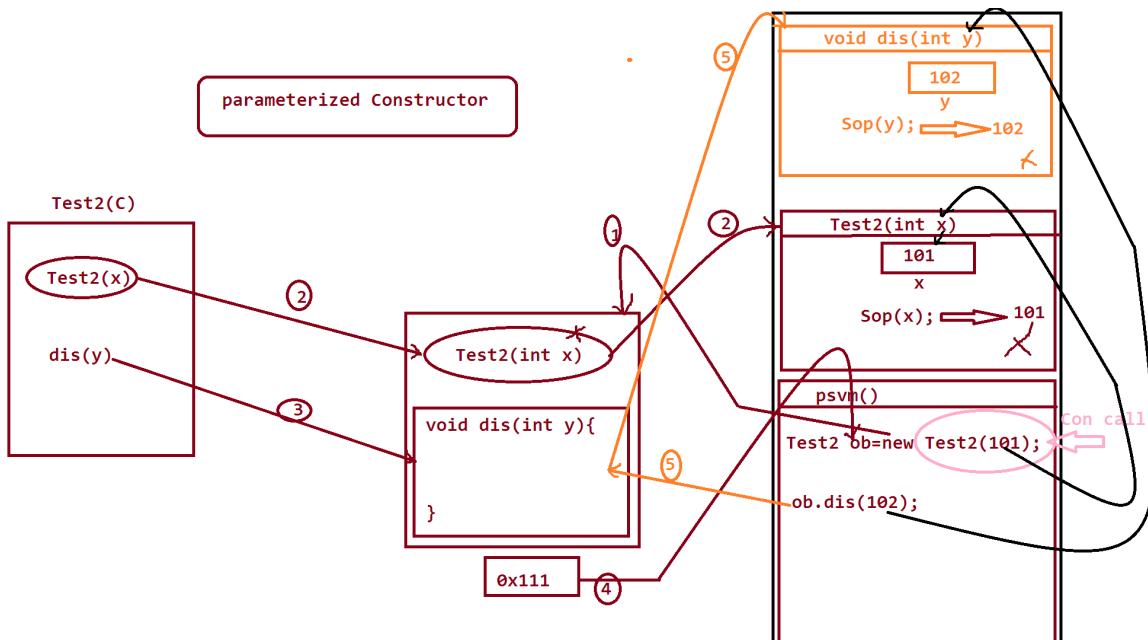
---

**Execution flow of above progra:(DCon2.java)**

**ClassFiles:**

**Test2.class**

**DCon2.class**



**Note:**

=>We pass parameters to the parameterized constructors while object creation.

---

**faq:**

**wt is the advantage of Constructor?**

=>Constructors are used to initialize instance variables while object creation and which saves the execution time and generates high performance of an application.

**Exp program:**

```
/*Program to read and display Product details*/
```

```
import java.lang.String;
import java.lang.System;
import java.util.Scanner;
```

```
class Product //SubClass
{
    String pCode,pName;
    float pPrice;
    int pQty;
    Product(String pCode,String pName,float pPrice,int pQty)
    {
        this.pCode=pCode;
        this.pName=pName;
        this.pPrice=pPrice;
        this.pQty=pQty;
    }
    void getProduct()
    {
        System.out.println("ProdCode:"+pCode);
        System.out.println("ProdName:"+pName);
        System.out.println("ProdPrice:"+pPrice);
        System.out.println("ProdQty:"+pQty);
    }
}
class DCon4 //MainClass
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);//Built-Class object
```

```
System.out.println("Enter the PCode:");
String pC = s.nextLine();
System.out.println("Enter the PName:");
String pN = s.nextLine();
System.out.println("Enter the PPrice:");
float pP = s.nextFloat();
System.out.println("Enter the PQty:");
int pQ = s.nextInt();
```

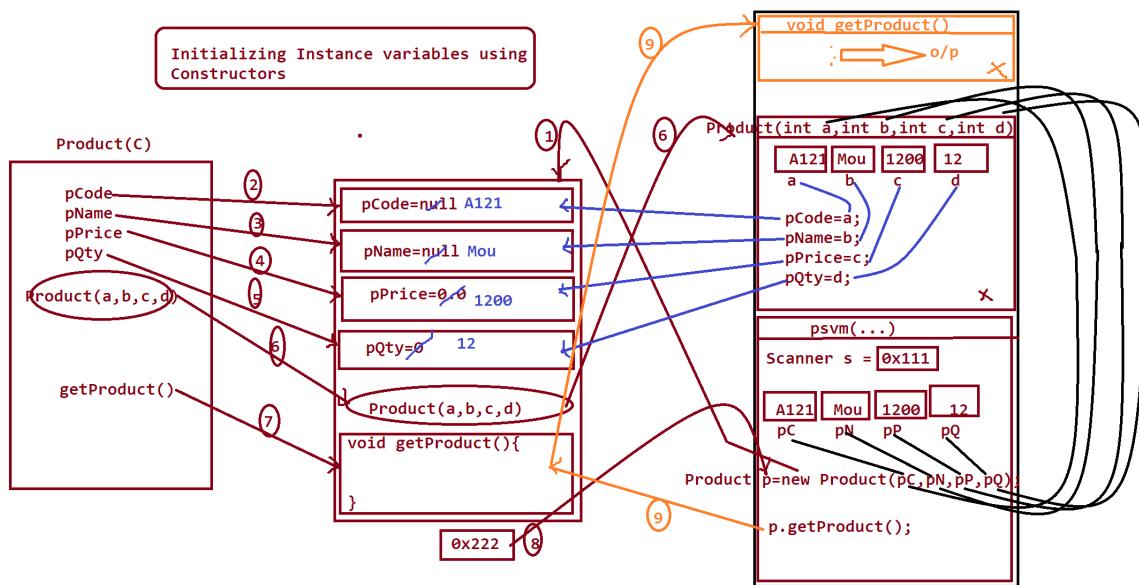
```
Product p = new Product(pC,pN,pP,pQ);
p.getProduct();//Method call
}
```

**Execution flow of above program:**

**ClassFiles:**

**Product.class**

**DCon4.class**



### Assignment1:

wap to display Book details?

### BookData

=>bCode,bName,bAuthor,bPrice,bQty

=>void getBookData()

### BMainClass

=>public static void main(String[] args)

### Assignment2:

### StuDetails

=>rollNo,stuName,stuBr  
=>void getStuDetails()

### **StuAddress**

=>hNo,sName,city,pinCode  
=>void getStuAddress()

### **SMainClass**

=>public static void main(String[] args)

---

#### **Note:**

=>Use Constructors to initialize instance variables

---

Dt : 2/12/2020

#### **define 'this' keyword?**

=>'this' keyword will hold the reference of object from where the current method or current constructor is executing.

#### **Note:**

=>'this' keyword is used when we have same variable names in Local variables and Instance variables.

---

#### **faq:**

**define default constructor?**

=>The constructor without parameters,which is added by the compiler at compilation stage is known as default constructor.

faq:

In what situation default constructor is added?

=>If the compiler finds any class declared without any constructors then the compiler will add default constructor.

faq:

define static Constructor?

=>There is no concept of static constructors in Java.Because constructor means executed while object creation and cannot be at class level.

---

\*imp

Relationship b/w Classes:

=>The process of establishing communication b/w classes is known as Relationship b/w Classes.

Relationship b/w classes are categorized into three types:

1.References

2.Inheritance

3.InnerClasses

## **1. References:**

=>The process of declaring NonPrimitive datatype variable part of class and this NonPrimitive datatype variable will hold the reference of object of another class is known as 'References Concept'.

### **Exp program1:**

```
class SubClass2
{
    int b=20;
    void m2()
    {
        System.out.println("==>m2()===");
        System.out.println("The value b:"+b);
    }
}

class SubClass1
{
    int a=10;
    SubClass2 ob2;//Loosly Coupled reference
    SubClass1(SubClass2 ob2)
    {
        this.ob2=ob2;
    }
    void m1()
    {
```

```
System.out.println("==m1()==");
System.out.println("The value a:"+a);
System.out.println("The value b:"+ob2.b);
    ob2.m2();
}
}

class DRef1 //MainClass
{
    public static void main(String[] args)
{
```

```
SubClass2 ob2 = new SubClass2();
SubClass1 ob1 = new SubClass1(ob2);
ob1.m1();
}
}
```

---

#### Ex program2:

wap to read and display Emp details?  
(Use references concept)

```
import java.util.Scanner;
class Address//SubClass
{
    String hNo,sName,city;
    int pinCode;
```

```
void getAddress()
{
System.out.println("HNo:"+hNo+"\nName:"+sName+"\nCity:"+city+
"\nPinCode:"+pinCode);
}

}

class Employee//SubClass

{
    String id,name;
    Address ad = new Address();//Tightly Coupled reference

    void getEmployee()
    {
        System.out.println("Id:"+id+"\nname:"+name);
    }
}

class Read//SubClass

{
    void read(Scanner s,Employee e)
    {
        System.out.println("Enter the id:");
        e.id = s.nextLine();
        System.out.println("Enter the name:");
        e.name = s.nextLine();
        System.out.println("Enter the hNo:");
        e.ad.hNo = s.nextLine();
    }
}
```

```
System.out.println("Enter the sName:");
e.ad.sName = s.nextLine();

System.out.println("Enter the city:");
e.ad.city = s.nextLine();

System.out.println("Enter the pinCode:");
e.ad.pinCode = s.nextInt();

}

}

class Display//SubClass

{
    void dis(Employee e)
    {
        e.getEmployee();
        e.ad.getAddress();
    }
}

class DRef2 //MainClass

{
    public static void main(String[] args)
    {
Scanner s = new Scanner(System.in);

Employee e = new Employee();

Read r = new Read();

r.read(s,e);

Display d = new Display();
```

```
d.dis(e);
```

```
}
```

```
}
```

Dt : 3/12/2020

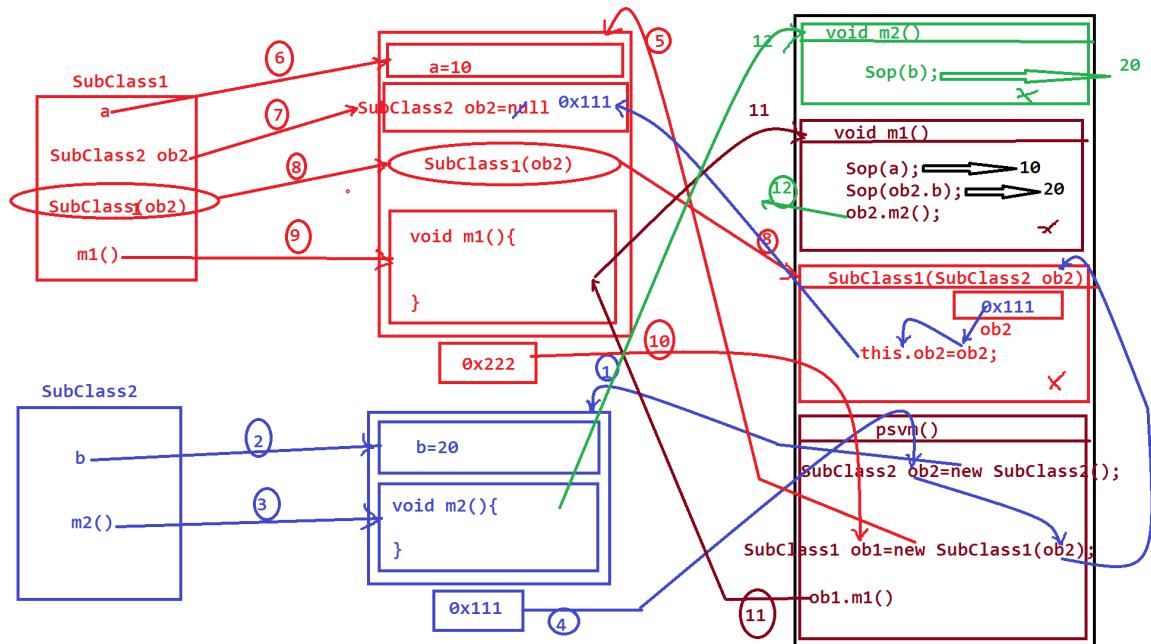
Execution flow of above program:(DRef1.java)

ClassFiles:

**SubClass2.class**

**SubClass1.class**

**DRef1.class(MainClass)**



Note:

=>In the above diagram the object of SubClass1 is holding the Object reference of SubClass2,in this process the methods of SubClass1 will access all the members of SubClass2 using this reference.

## Execution flow of above program:(DRef2.java)

**ClassFiles:**

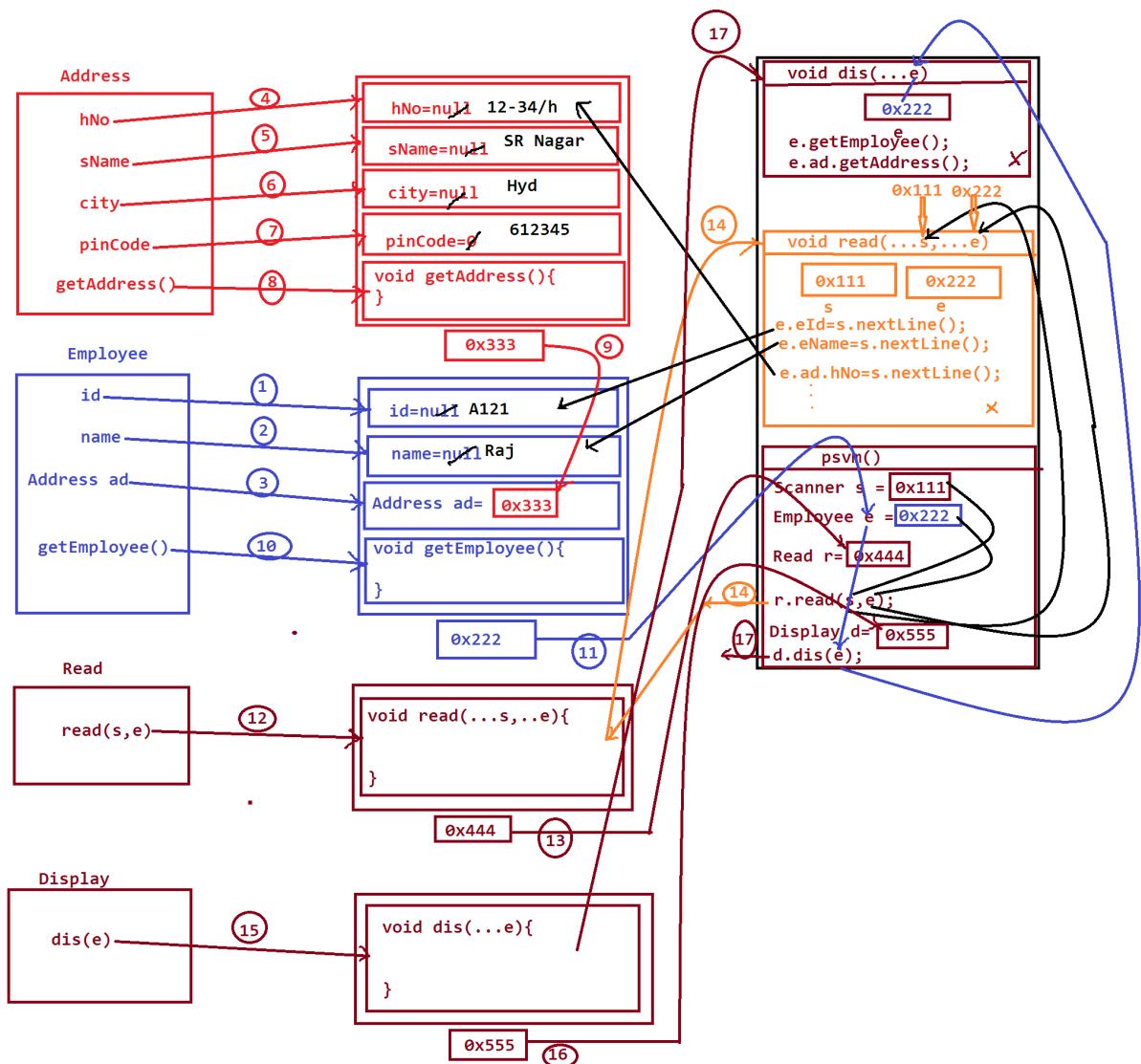
**Address.class**

**Employee.class**

**Read.class**

**Display.class**

**DRef2.class**



Dt : 4/12/2020

=>References in Java are categorized into two types:

- 1.Lososly Coupled References
- 2.Tightly Coupled References

1.Lososly Coupled References:

=>In Lososly coupled references the objects which are linked are independent objects.

Exp:

**DRef1.java**

Note:

=>SubClass2 object can be created without creating object for SubClass1.

---

2.Tightly Coupled References:

=>In Tightly Coupled references the objects which are linked are dependent objects.

Exp:

**DRef2.java**

Note:

=>In DRef2.java,Address class object is created while creating object for Employee class,in this process Address class object is depending on Employee class object.

---

Dt : 5/12/2020

**Note:**

=>when we pass NonPrimitive DataType variables as parameters to the methods,then the references are passed as parameters.

**Assignment:**

**Construct program to demonstrate "Bank Transaction Model".**

**step1 : read pinNo**

**step2 : pinNo must be in b/w**

**1111**

**2222**

**3333**

**step3 : If the pinNo is validated then display the choice**

**1.withDraw**

**2.Deposit**

**1.withDraw**

=>Enter int amt and the amt must be Greater than zero and must

**be multiples of 100**

=>If amt is validated then pass the amt as parameter to the

**method of WithDraw class.**

=>This method will check the amt is less than balance or not

=>If the amt is less than balance then perform transaction and

**display the following:**

**Amt withDrawn =**

**Balance =**

**Transaction completed**

## **2.Deposit**

**=>Enter int amt and the amt must be Greater than zero and must**

**be multiples of 100**

**=>If amt is validated then pass the amt as parameter to the  
method of Deposit class.**

**=>Display the following:**

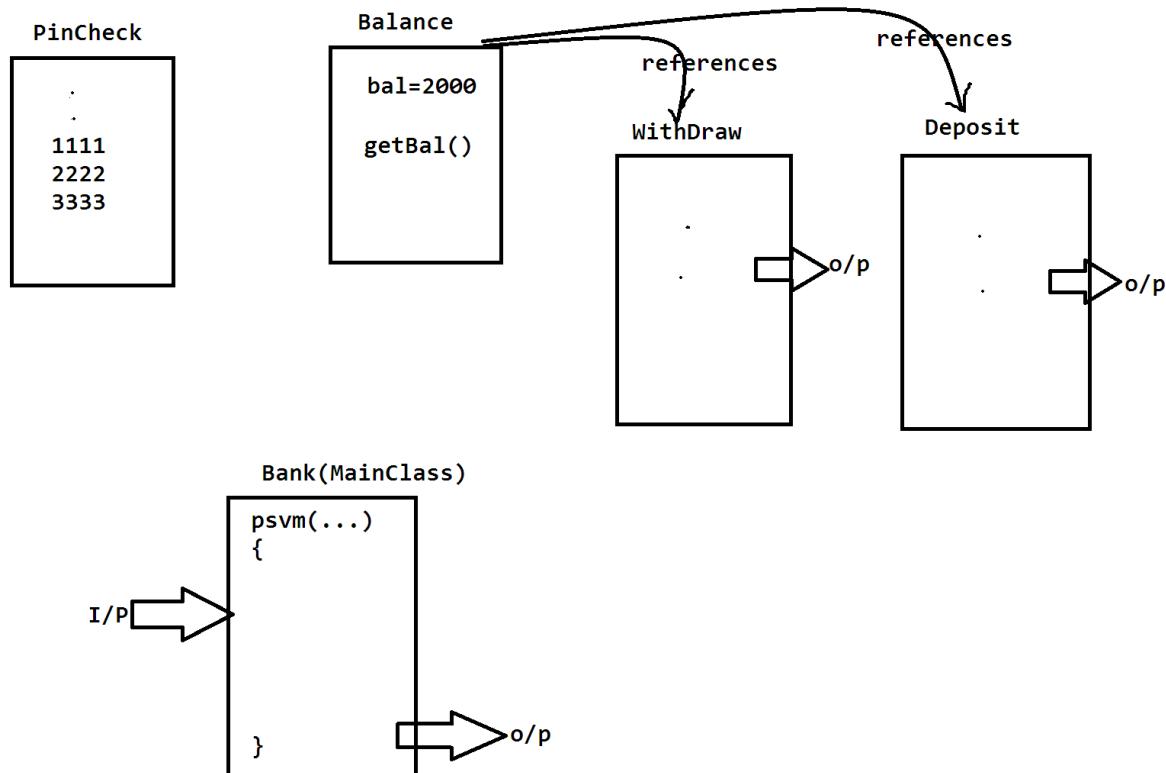
**Amt deposited =**

**Balance =**

**Transaction completed**

### **Note:**

**=>If the pinNo validation is failed for three times then display  
the msg as "Transaction blocked temporarily".**



Dt : 7/12/2020

\*imp

## 2.Inheritance:

=>The process of establishing communication b/w two classes using  
'extends' keyword is known as Inheritance.

**syntax:**

```

class SubClass2

{
    //members
}

```

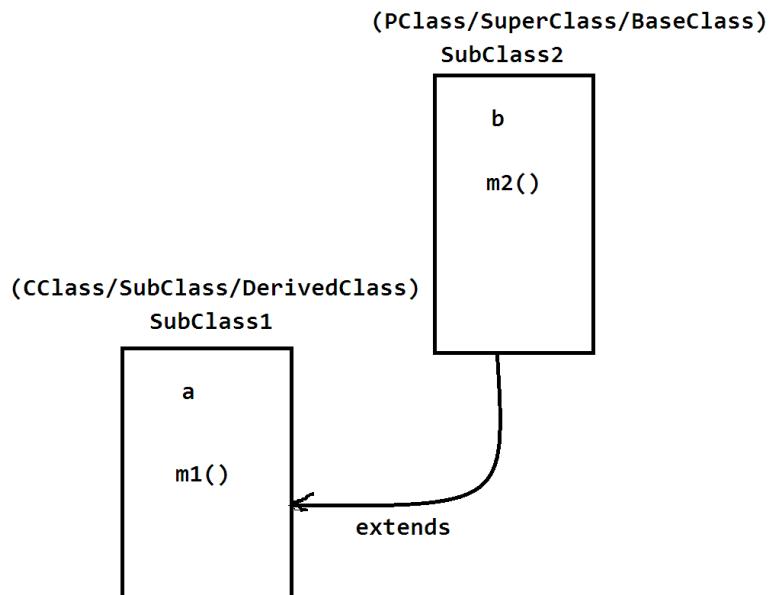
```

class SubClass1 extends SubClass2

```

```
{  
//members  
}
```

=>In Normal Inheritance process we always create object for CClass or SubClass.



---

Note:

=>In Normal Inheritance process all the members of PClass are available to CClass through 'extends' keyword.

---

Inheritance Case-I: NonStatic members of PClass or SuperClass

**Exp program:**

```
class SubClass2 //PClass
{
    int b;
    void m2()
    {
        System.out.println("====PClass====");
        System.out.println("The value b:"+b);
        //System.out.println("The value a:"+a);//Compilation
    }

    {
        System.out.println("==PClass NonStatic Block===");
    }
}

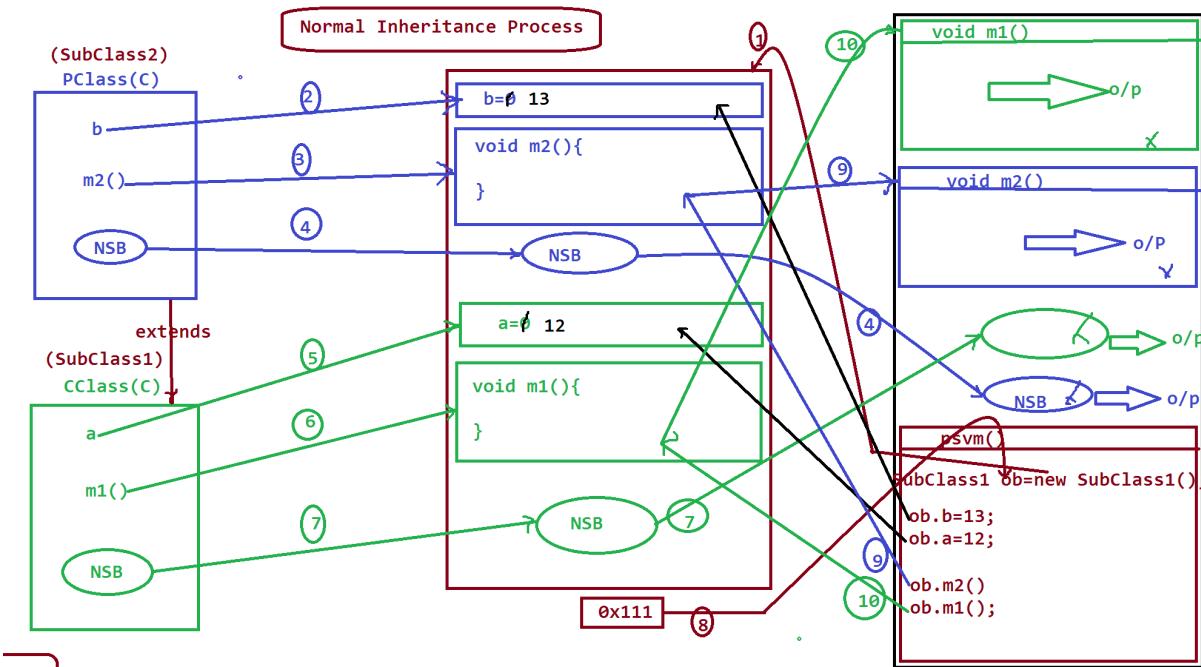
class SubClass1 extends SubClass2 //CClass
{
    int a;
    void m1()
    {
        System.out.println("==CClass===");
        System.out.println("The value a:"+a);
        System.out.println("The value b:"+b);
    }
}
```

```
{  
System.out.println("====CClass NonStatic Block====");  
}  
}  
class Inheritance1 //MainClass  
{  
    public static void main(String[] args)  
    {  
SubClass1 ob = new SubClass1();//Normal Inheritance process  
ob.b=13;  
ob.a=12;  
ob.m2();  
ob.m1();  
    }  
}
```

**Execution flow of above program:**

**ClassFiles:**

**SubClass2.class(PClass)**  
**SubClass1.class(CClass)**  
**Inheritance1.class(MainClass)**



**Note:**

=>In Normal Inheritance process one reference is created and the reference is binded with all the NonStatic members of PClass and all the NonStatic members of CClass.

=>In Inheritance process the PClass is loaded first and then the CClass is loaded.

=>In Inheritance process while object creation PClass members are binded first and then the CClass members are binded.

=>In Inheritance process through 'extends' keyword the PClass members are available to CClass,in this process the members of CClass can access all the members of PClass but PClass members cannot access CClass members.

---

**faq:**

**define Method Overriding process?**

=>When we have same method signature in PClass and CClass,then PClass method is replaced by CClass method while object creation is known as Method Overriding process.

**Same Method Signature means.**

- same return\_type**
- same method\_name**
- same para\_list**
- same para\_type**

**Exp program:**

```
class PClass
{
    void m(int x)//Overrided method
    {
        System.out.println("==PClass===");
        System.out.println("The value x:"+x);
    }
}

class CClass extends PClass
{
    void m(int x)//Overriding method
    {
        System.out.println("==CClass===");
    }
}
```

```
System.out.println("The value x:"+x);

    }

}

class Inheritance2 //MainClass

{

    public static void main(String[] args)

    {
```

```
    CClass ob = new CClass();

    ob.m(102);

}
```

---

faq:

**defien Method OverLoading process?**

**=>More than one method with same method\_name but differentiated by their para\_list or para\_type is known as Method OverLoading process.**

**Exp program:**

```
class PClass

{

    void m(int x)

    {

System.out.println("==PClass==");

System.out.println("The value x:"+x);

    }
```

```
}

class CClass extends PClass

{

    void m(int x,int y)

    {

System.out.println("====CClass====");

System.out.println("The value x:"+x);

System.out.println("The value y:"+y);

    }

}

class Inheritance3 //MainClass

{
```

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

    {

CClass ob = new CClass();

ob.m(12);

ob.m(13,14);

    }

}
```

---

Dt : 8/12/2020

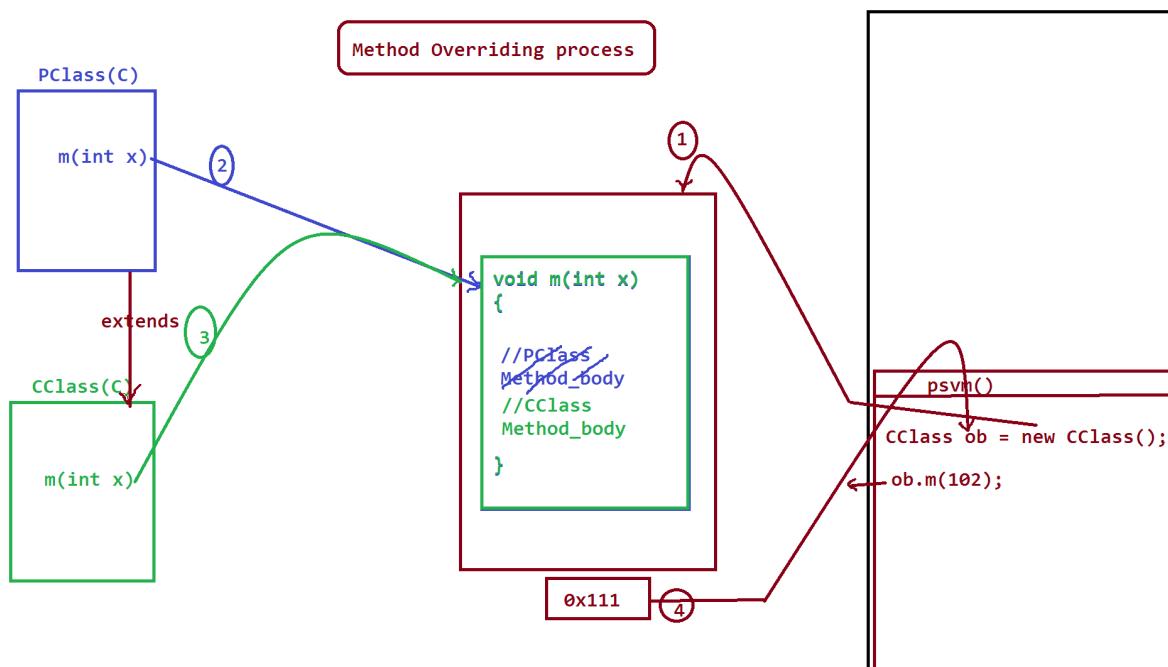
**Execution flow of above program:(Method Overriding process)**

**ClassFiles:**

**PClass.class**

**CClass.class**

## Inheritance2.class



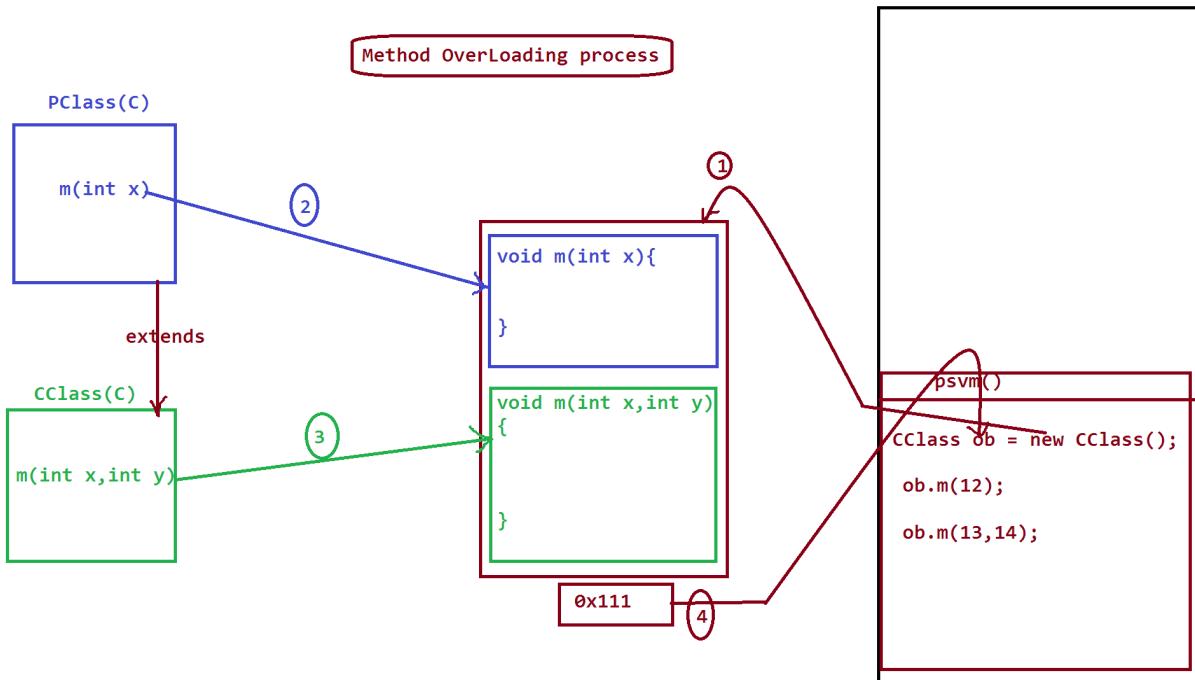
Execution flow of above program:(Method OverLoading process)

ClassFiles:

**PClass.class**

**CClass.class**

**Inheritance3.class**



\*imp

**define Generalization process?**

=>The process in which one reference is created and the reference is binded with all the members of PClass and only Overriding members from the CClass, is known as Generalization process.

=>The following syntax is used to achieve Generalization process:

PClass ob = new CClass();

**Exp program:**

```

class PClass
{

```

```
void m1()
{
System.out.println("==PClass m1()===");
}

void m2()
{
System.out.println("==PClass m2()===");
}

}

class CClass extends PClass
{

void m1()
{
System.out.println("==CClass m1()===");
}

void m3()
{
System.out.println("==Class m3()===");
}

}

class Inheritance4 //MainClass
{
public static void main(String[] args)
{
PClass ob = new CClass(); //Generalization process
```

```

ob.m1();
ob.m2();
//ob.m3(); //Compilation Error
}

}

```

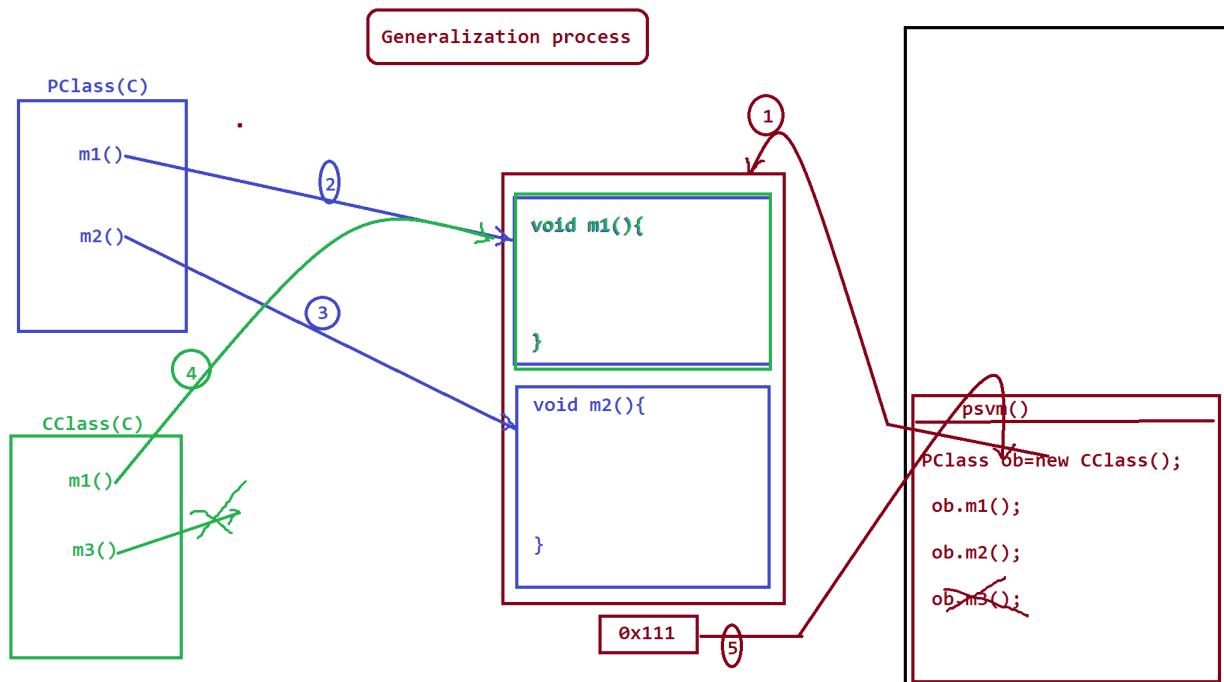
**Execution flow of above program:**

**ClassFiles:**

**PClass.class**

**CClass.class**

**Inheritance4.class**



**Note:**

=>This Generalization process can be achieved by declaring the CClass

without name known as 'Anonymous InnerClass as Class extention'.

**syntax;**

```
class PClass
{
    //PClass_body
}
```

```
PClass ob = new PClass()
{
    //CClass_body
};
```

**Exp program:**

```
class PClass
{
    void m1()
    {
        System.out.println("==PClass m1()==");
    }
    void m2()
    {
        System.out.println("==PClass m2()==");
    }
}
```

```
}

class Inheritance5 //MainClass
{
    public static void main(String[] args)
    {
        PClass ob = new PClass()

        {
            void m1()
            {

System.out.println("====CClass m1()====");

            }

            void m3()
        }

System.out.println("====Class m3()====");

        }

    };

    ob.m1();

    ob.m2();

//ob.m3(); //Compilation Error

    }

}
```

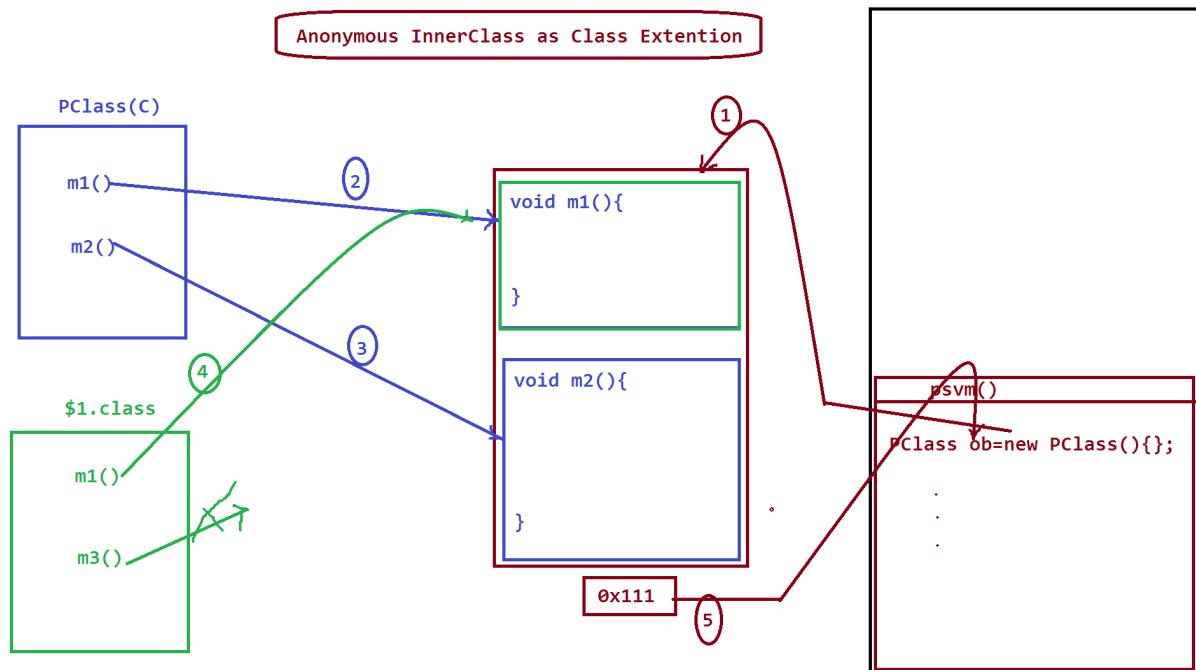
**Execution flow of above program:**

## ClassFiles:

PClass.class

Inheritance5.class(MainClass)

Inheritance5\$1.class(CClass without name)



Dt : 9/12/2020

## Summary of Inheritance:

**syntax1 :** CClass ob = new CClass();

=>In this Normal Inheritance process one reference is created and the reference is binded with all the members of PClass and all the members of CClass.

**syntax2 : PClass ob = new CClass();**

=>In this Generalization process one reference is created and the reference is binded with all the members of PClass and only Overriding members from the CClass.

**syntax3 : PClass ob = new PClass(){};**

=>This also generalization process only but in which the CClass is decalred without name.

=====

### **Inheritance Case-II: Constructors from the PClass or SuperClass**

#### **(i)0-parameter constructor from the PClass**

=>when we have 0-parameter constructor from the PClass then the Compiler at compilation stage will add 'super()' to the constructor of CClass to call 0-parameter constructor from the PClass.

**Exp program:**

```
class PClass
{
    PClass()
    {
        System.out.println("PClass 0-parameter Constructor...");
```

```
    }

}

class CClass extends PClass

{

    CClass()

    {

        System.out.println("CClass 0-parameter Constructor...");

    }

}

class Inheritance6 //MainClass

{
```

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

    {

        CClass ob = new CClass();

    }

}
```

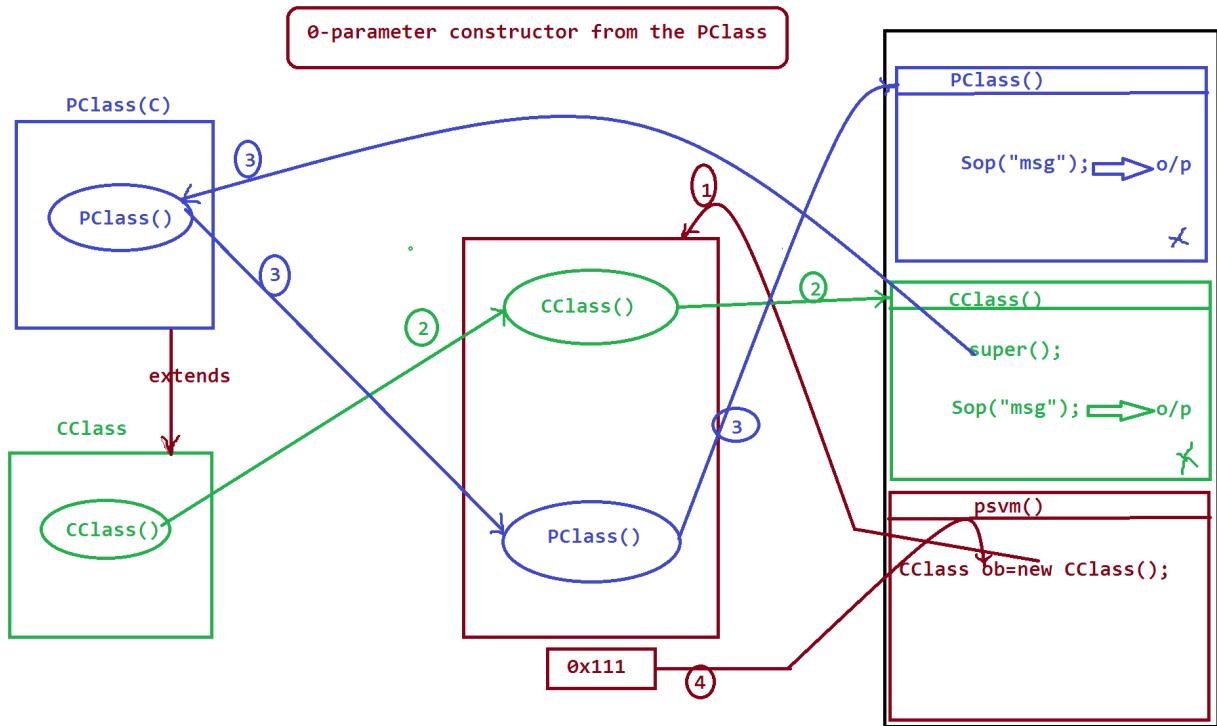
**Execution flow of above program:**

**ClassFiles:**

**PClass.class**

**CClass.class**

**Inheritance6.class**



**Note:**

=>In inheritance process the CClass constructor will call PClass

**Constructor for execution.**

---

### (ii)Parameterized Constructor from the PClass

=>when we have parameterized constructor in the PClass then we must add `super()` to the Constructor of CClass to call PClass class Constructor.

**Ex program:**

```
class PClass
{
    PClass(int x)
```

```
{  
System.out.println("==PClass==");  
System.out.println("The value x:"+x);  
}  
}  
  
class CClass extends PClass  
{  
    CClass(int x)  
    {  
        super(x);  
    }  
}
```

```
class Inheritance7 //MainClass  
{  
    public static void main(String[] args)  
    {  
        CClass ob = new CClass(123);  
    }  
}
```

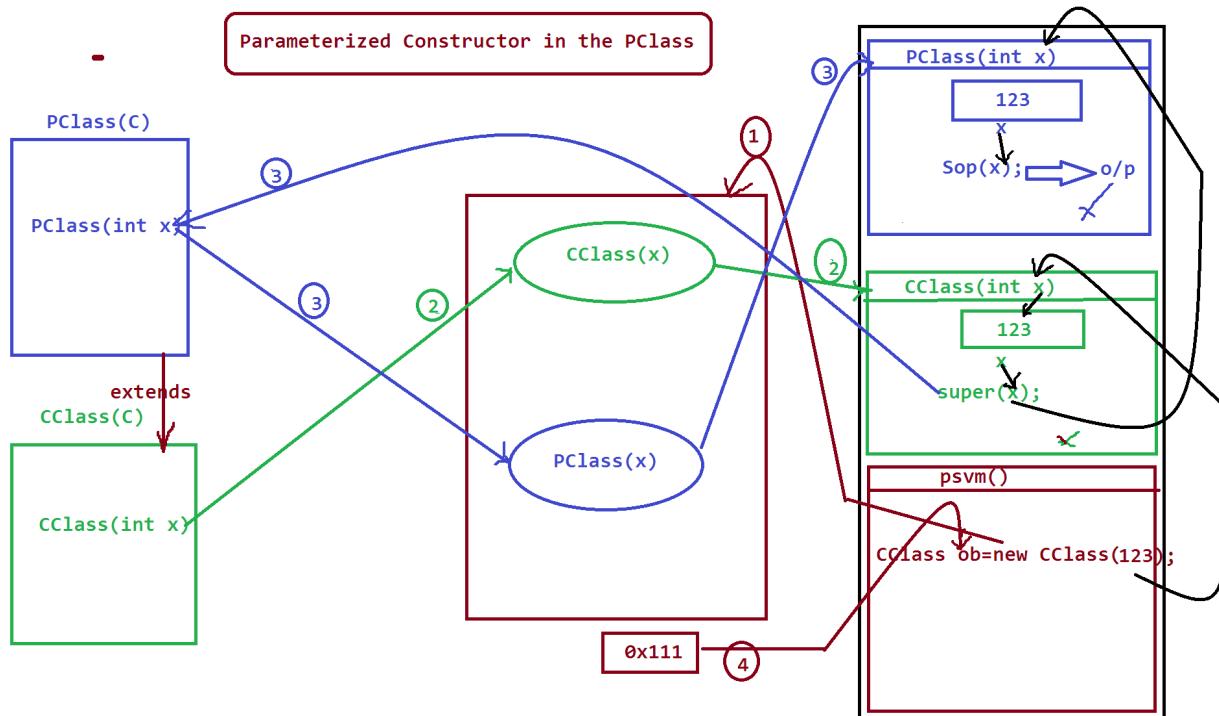
**Execution flow of above program:**

**ClassFiles:**

**PClass.class**

**CClass.class**

**Inheritance7.class**



Dt : 11/12/2020

faq:

wt is the diff b/w

(i)super()

(ii)this()

(i)super():

=>super() is used to access the constructors from the PClass or  
SuperClass.

(ii)this():

=>this() is used to access the constructors from the same class or  
Current running class.

Exp program:

wap to demonstrate "super()" and "this()"?

```
class PClass
{
    PClass(int a,int b)
    {
        this(a);
        System.out.println("The value b:"+b);
    }
    PClass(int a)
    {
        System.out.println("The value a:"+a);
    }
}
```

```
}

class CClass extends PClass

{

    CClass(int a,int b,int c,int d)
    {

        this(a,b,c);

System.out.println("The value d:"+d);

    }

    CClass(int a,int b,int c)
    {

        super(a,b);

System.out.println("The value c:"+c);

    }
}
```

```
}

class Inheritance8 //MainClass

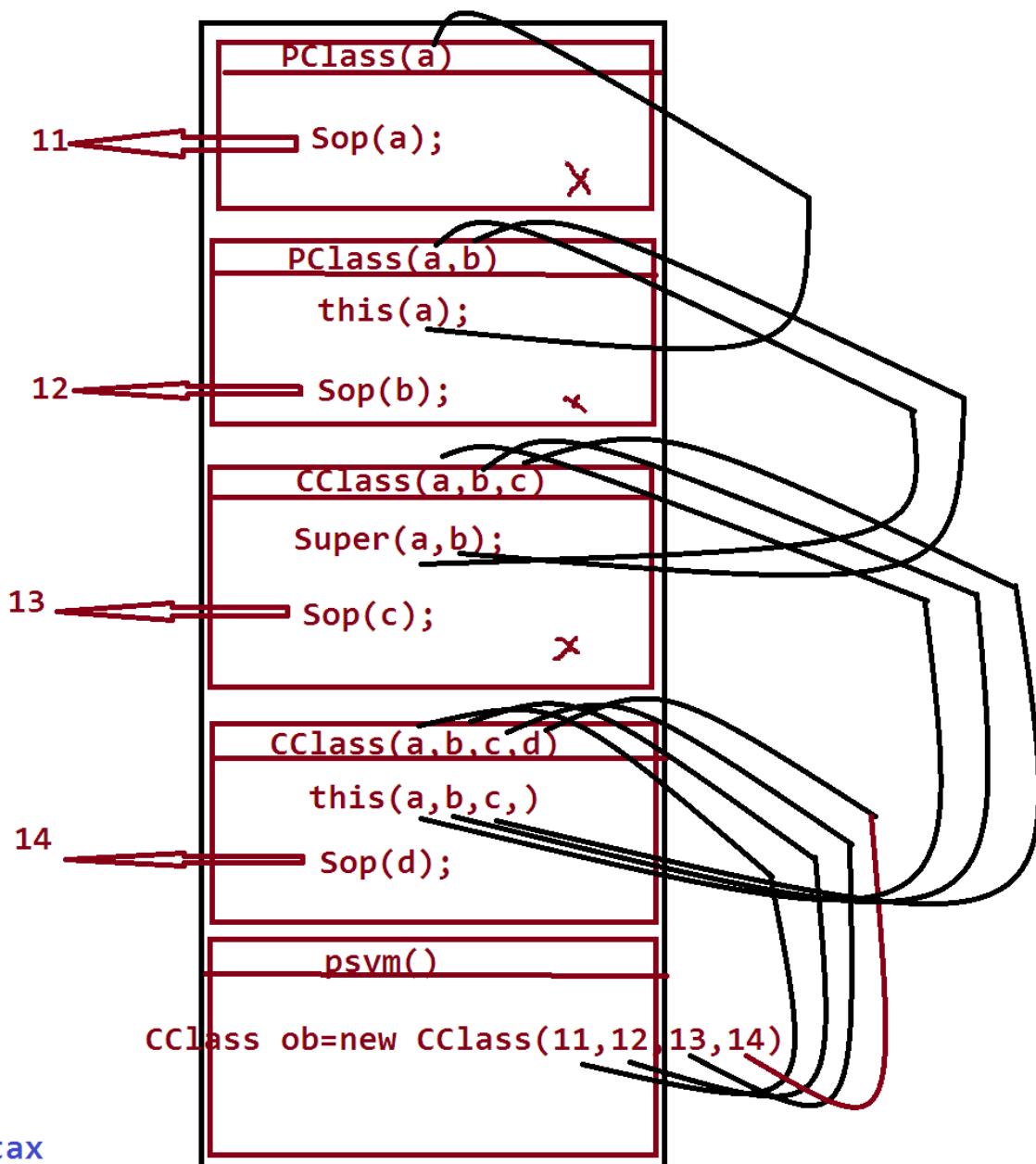
{

    public static void main(String[] args)
    {

CClass ob = new CClass(11,12,13,14);

    }

}
```



## Syntax

faq:

**define Constructor Chaining process?**

=>The process of calling the Constructor from the Constructor is known as Constructor Chaining process.

**Exp program:**

**above program**

**faq:**

**define Constructor OverLoading process?**

**=> More than one constructor differentiated bt their Para\_list or  
Para\_type is known as Constructor OverLoading process.**

**Exp program:**

**above program**

**faq:**

**define Constructor Overriding process?**

**=>There is no concept of Constructor Overriding process,because we  
cannot have same Constructor names in PClass and CClass.**

---

**Inheritance case-III: Static members from the PClass or SuperClass.**

**=>In heritance process all the static members of PClass are available  
to CClass through extends keyword and can be accessed with the  
CClass\_name.**

**Exp program:**

```
class PClass
```

```
{
```

```
    static int a;
```

```
static void m1()
{
System.out.println("==PClass==");
System.out.println("The value a:"+a);
}

static

{
System.out.println("PClass static block...");

}

class CClass extends PClass

{
    static int b;

    static void m2()
    {

System.out.println("==CClass==");
System.out.println("The value b:"+b);

    }

    static

    {
System.out.println("CClass static block...");

    }

class Inheritance9 //MainClass

{
```

```
public static void main(String[] args)
{
    CClass ob = new CClass();
    CClass.a=12;
    CClass.b=13;
    CClass.m1();
    CClass.m2();
}
```

Dt : 12/12/2020

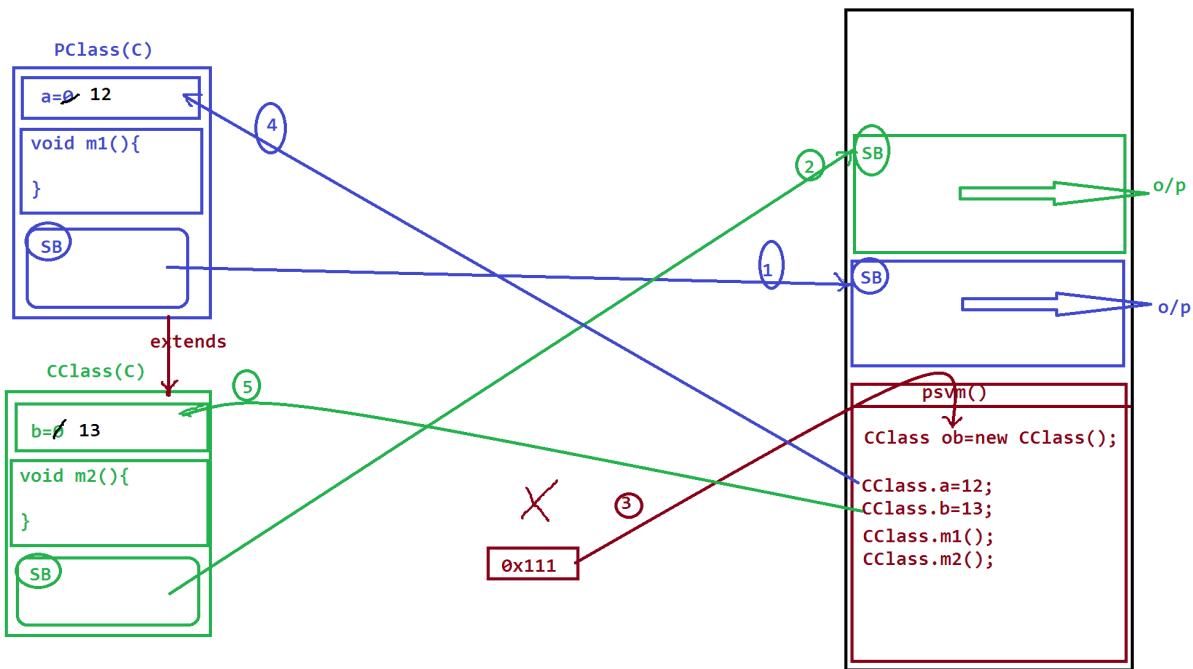
**Execution flow of above program:**

**ClassFiles:**

**PClass.class**

**CClass.class**

**Inheritance9.class**



**define Static Method Overloading process?**

=>More than one static method with same method name but differentiated

by their para\_list or para\_type is known as Static Method OverLoading process.

**Exp program:**

```
class PClass
{
    static void m(int x,int y)
    {
        System.out.println("==PClass===");
        System.out.println("The value x:"+x);
        System.out.println("The value y:"+y);
    }
}
```

```
}

class CClass extends PClass

{

    static void m(int x,int y,int z)

    {

System.out.println("==CClass==");

System.out.println("The value x:"+x);

System.out.println("The value y:"+y);

System.out.println("The value z:"+z);

    }

    static void m(int x,float y)

    {

System.out.println("==CClass==");

System.out.println("The value x:"+x);

System.out.println("The value y:"+y);

    }

}

class Inheritance10 //MainClass

{

    public static void main(String[] args)

    {

CClass.m(1,2);

CClass.m(1,2,3);

CClass.m(1,2.3F);

    }

}
```

}

---

**faq:**

**can we perform OverLoading process for Standard main() method?**

**=>Yes,we can perform OverLoading process for standard main() method because it also static method.**

**faq:**

**Can we pass parameters to the Standard main() method?**

**=>Yes,we can pass parameters to the Standard main() method while execution command because main() method call is available within the execution command.**

**syntax:**

**java Class\_name para1 para2 para3...**

**Exp:**

**java Inheritance11 Java8 by 2014**

**Note:**

**=>The parameters which are passed while execution command are holded by String Array.(args)**

**faq:**

**define CommandLine Argument program?**

**=>The process of passing parameters to the standard main() method is known as CommandLine Argument program.**

**Exp program:**

```
class Inheritance11 //MainClass
{
    public static void main(String[] p)
    {
        Inheritance11.main(12.34F);
        Inheritance11.main(12);
        System.out.println("==Standdard main() method===");
        for(int i=0;i<p.length;i++)
        {
            System.out.println(p[i]);
        }
    }
    public static void main(float x)
    {
        System.out.println("==OverLoaded main() method===");
        System.out.println("The value x:"+x);
    }
    public static void main(int y)
    {
        System.out.println("==OverLoaded main() method===");
    }
}
```

```
System.out.println("The value y:"+y);  
}  
}  
-----
```

faq:

**define static method Overriding process?**

=>There is no concept of static method Overriding process,because the methods are binded to the classes and available in the classLevel.

Dt : 14/12/20202

faq:

**define Method Hiding process?**

=>when we have same static method signature in PClass and CClass then the CClass method is executed and the PClass method is not identified for execution,in this process PClass method is hided by CClass method known as Method Hiding process.

**Exp program:**

```
class PClass  
{  
    static void m(int x)//Method Hiding  
    {  
        System.out.println("==PClass m()==");  
        System.out.println("The value x:"+x);  
    }  
}
```

```
}

class CClass extends PClass

{

    static void m(int x)

    {

System.out.println("====CClass m()====");

System.out.println("The value x:"+x);

    }

}

class Inheritance12 //MainClass

{

    public static void main(String[] args)

    {

CClass.m(102);

    }

}

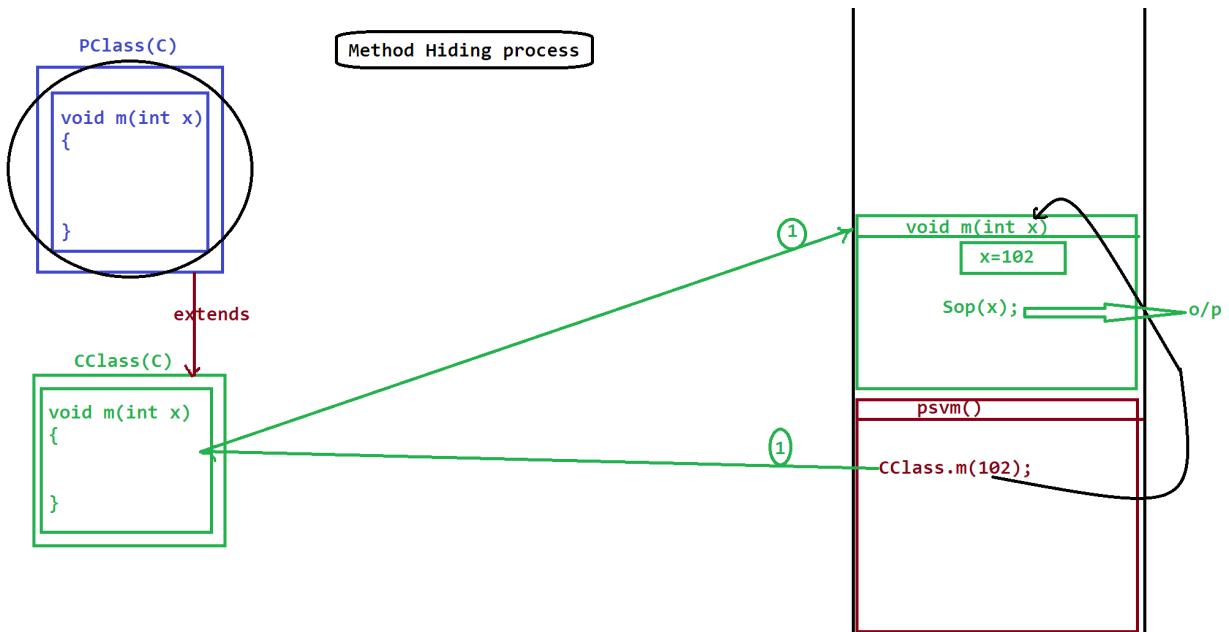
Execution flow of above program:

ClassFiles:

PClass.class

CClass.class

Inheritance12.class
```



**faq:**

**can we use super keyword to access the static members from the PClass?**

=>Yes,we can use super keyword to access static members of PClass

**but the super keyword must be used within the NonStatic members of CClass.**

**Exp program:**

```
class PClass
{
    static void m(int x)
    {
        System.out.println("==PClass m()==");
        System.out.println("The value x:"+x);
    }
}
```

```
class CClass extends PClass
{
    static void m(int y)
    {
        System.out.println("====CClass m()====");
        System.out.println("The value y:" + y);
    }

    void dis(int x,int y)
    {
        super.m(x); //PClass method call
        this.m(y); //same class method call
    }
}
```

```
class Inheritance13 //MainClass
{
    public static void main(String[] args)
    {
        CClass ob = new CClass();
        ob.dis(101,102);
    }
}
```

faq:

wt is the diff b/w

**(i)super**

**(ii)super()**

**(i)super:**

**=>super keyword is used to access the variables and methods from the PClass.**

**(ii)super():**

**=>super() is used to access the constructors from the PClass.**

**faq:**

**wt is the diff b/w**

**(i)this**

**(ii)this()**

**(i)this:**

**=>this is used to access the variables and methods from the same class.**

**(ii)this():**

**=>this() is used to access the constructors from the same class.**

---

**Types of Inheritances:**

**=>Inheritances are categorized into the following:**

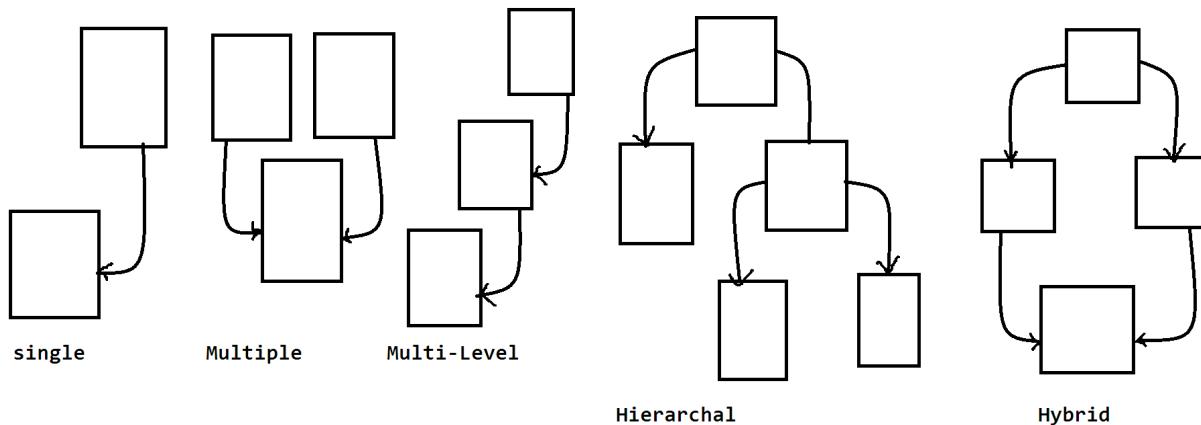
**1.Single Inheritance**

**2.Multiple Inheritance**

### **3. Multi-Level Inheritance**

### **4. Hierarchical Inheritance**

### **5. Hybrid Inheritance.**



=>In realtime these Inheritances are categorized into two types:

**(a) Single Inheritance**

**(b) Multiple Inheritance**

**(a) Single Inheritance:**

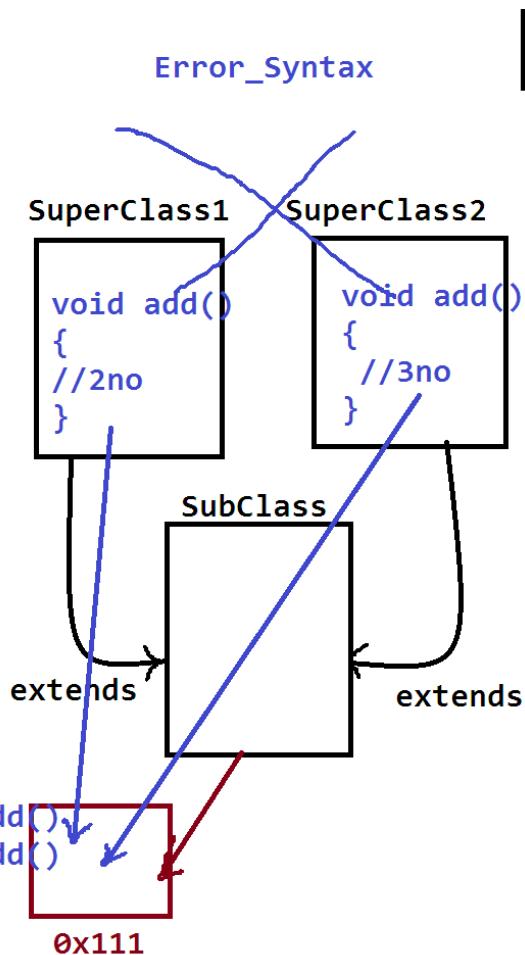
=>The process of extracting the features from one class at-a-time  
is known as **Single Inheritance**.

**Exp:**

**above programs**

**(b) Multiple Inheritance:**

=>The process of extracting the features from more than one class  
at a time is known as **Multiple Inheritance**.



**Note:**

=>Multiple Inheritance using Classes concept is removed from JavaLang and not available in Java,because which leads to replication of components and raised ambiguity(confusion).The ambiguity state applications will generate Wrong results.

**Note:**

=>In JavaLang the Multiple Inheritance process can be achieved using Interfaces.

---

Dt : 15/12/2020

\*imp

**Interfaces in Java:**

=>Interface is a collection of variables and abstract methods upto Java7 version.

**define abstract methods:**

=>The methods which are declared without method\_body are known as abstract methods.

**Structure of abstract method:**

```
return_type method_name(para_list);
```

**define Concrete methods:**

=>The methods which are declared with method\_body are known as Concrete methods

**Structure of Concrete method:**

```
return_type method_name(para_list)
{
    //method_body
}
```

**Coding Rules of Interfaces:**

1.we use 'interface' keyword to declare interfaces.

**syntax:**

```
interface Interface_name  
{  
//members  
}
```

**2.The members which are declared within the interface are automatically 'public'.**

**Note:**

=>The members which are declared within the class are automatically 'default'.

**3.Interface can be declared with both Primitive datatype variables and NonPrimitive datatype variables.**

**4.The variables which are declared part of Interface are automatically 'static and final' variables.**

**Note:**

=>'static' variables are binded to interface and access with interface name.

=>'final' variables must be initialized with values,once initialized cannot be modified.

**Exp program:**

```
class Display
```

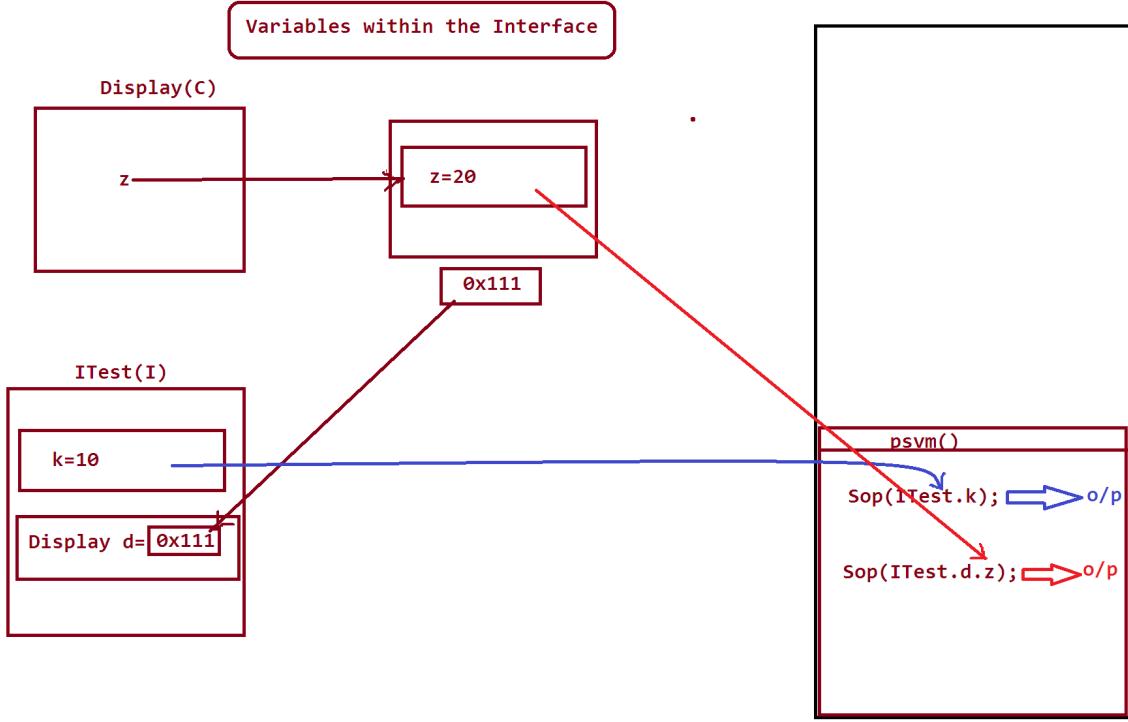
```
{  
    int z=20;  
}  
interface ITest  
{  
    public static final int k=10;//PDTV  
    public static final Display d = new Display();//NPDV  
}  
class Interface1 //MainClass  
{
```

```
    public static void main(String[] args)  
    {  
        System.out.println("The value k:"+ITest.k);  
        System.out.println("The value z:"+ITest.d.z);  
    }  
}
```

**Execution flow of above program:**

**ClassFiles:**

**Display.class**  
**ITest.class(I)**  
**Interface1.class(MainClass)**



**faq:**

**Can we declare NonStatic variables within the Interface?**

=>we cannot declare NonStatic variables within the Interafce,because  
the variables which are declared within the interface are automatically  
static variables.

---

**5.The methods which are declared within the interface are automatically  
NonStatic abstract methods.**

**Note:**

=>The methods which are declared within the class are automatically  
Concrete methods.

**6.There is no concept of Static abstract methods in Java.**

**7.We cannot instantiate Interface in Java because it is an abstract component in java.**

**Dt : 16/12/2020**

**8.Interfaces are implemented to implementation classes using 'implements' keyword.**

**9.These implementation classes will take the abstract methods and construct the body**

**Exp program:**

```
import java.util.Scanner;

interface IComparable
{
    public abstract int compareTo(int x,int y);
}

class Greater implements IComparable
{
    public int compareTo(int x,int y)
    {
        if(x>y) return x;
        else return y;
    }
}
```

```
}

class Smaller implements IComparable

{

    public int compareTo(int x,int y)

    {

        if(x<y) return x;

        else return y;

    }

}

class Interface2 //MainClass

{

    public static void main(String[] args)

    {

//IComparable ob = new IComparable();//Error

Scanner s = new Scanner(System.in);

System.out.println("Enter the value1:");

int v1 = s.nextInt();

System.out.println("Enter the value2:");

int v2 = s.nextInt();

System.out.println("====Choice====");

System.out.println("1.Greater\n2.Smaller");

System.out.println("Enter the choice:");

int choice = s.nextInt();

switch(choice)

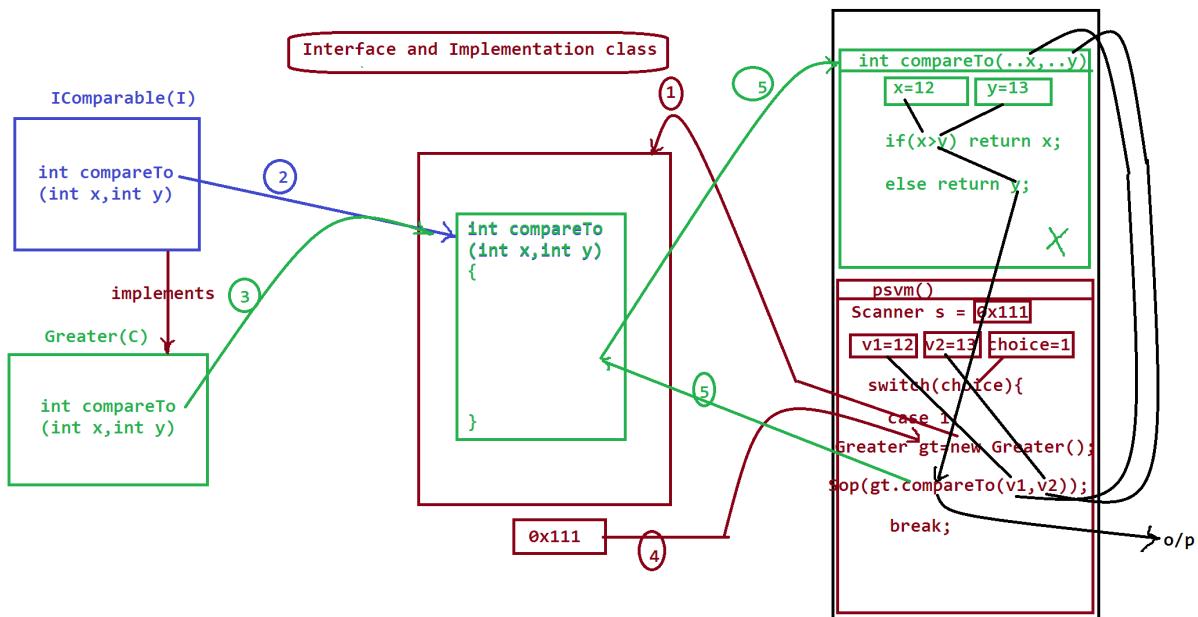
{
```

```
    case 1: //Greater  
  
Greater gt = new Greater();  
  
System.out.println("Greater Value:"+gt.compareTo(v1,v2));  
        break;  
  
    case 2: //Smaller  
  
Smaller sm = new Smaller();  
  
System.out.println("Smaller Value:"+sm.compareTo(v1,v2));  
        break;  
  
    default:  
  
System.out.println("Invalid choice...");  
  
}//end of switch  
  
}  
  
}
```

**Execution flow of above program:**

**ClassFiles:**

**IComparable.class**  
**Greater.class**  
**Smaller.class**  
**Interface2.class(MainClass)**



Dt : 17/12/2020

faq:

**define Runnable?**

=>'Runnable' is an built-in interface from java.lang package and which is used in constructing threads.

**structure of Runnable:**

```

public interface java.lang.Runnable
{
    public abstract void run();
}
    
```

**Steps used to construct threads:**

**step1 : User defined class must be implemented from 'Runnable'**

**interface**

**step2 : This user defined class will construct the body for run()**

**method**

**Note:**

**=>The required logic is constructed within the run() method.**

**step3 : Create object of User defined class**

**step4 : Create object for 'java.lang.Thread' class and while object**

**creation pass the User defined object reference as parameter.**

**Note:**

**=>Thread class object will hold the reference of User defined class**

**Object.**

**step5 : Call run() method using start() method**

**Exp program:**

```
class Display1 implements Runnable//step1
{
    public void run()//Step2
    {
        for(int i=1;i<=10;i++)
    }
```

```
{  
System.out.println("Val:"+i);  
try{  
Thread.sleep(2000);  
}catch(Exception e){}
//end-of-loop  
}  
}  
  
class Display2 implements Runnable  
{  
    public void run()  
    {  
        for(int i=101;i<=110;i++)  
        {  
System.out.println("Val:"+i);  
try{  
    Thread.sleep(2000);  
}catch(Exception e){}
        }
    }
}  
  
class Interface3 //MainClass  
{  
    public static void main(String[] args)  
    {
```

```
Display1 d1 = new Display1();//step3
```

```
Display2 d2 = new Display2();
```

```
Thread t1 = new Thread(d1);//step4
```

```
Thread t2 = new Thread(d2);
```

```
t1.start();//step5
```

```
t2.start();
```

```
}
```

```
}
```

---

**define Application?**

=>'Set-of-programs' collected together to perform defined action is known as application.

**define process?**

=>Application under execution is known as process.

**define Task?**

=>The part of process is known as Task.

**Note:**

=>In the above application each program is one Task.

**define MultiTasking?**

=>Executing Multiple Tasks Simultaneously is known as MultiTasking.

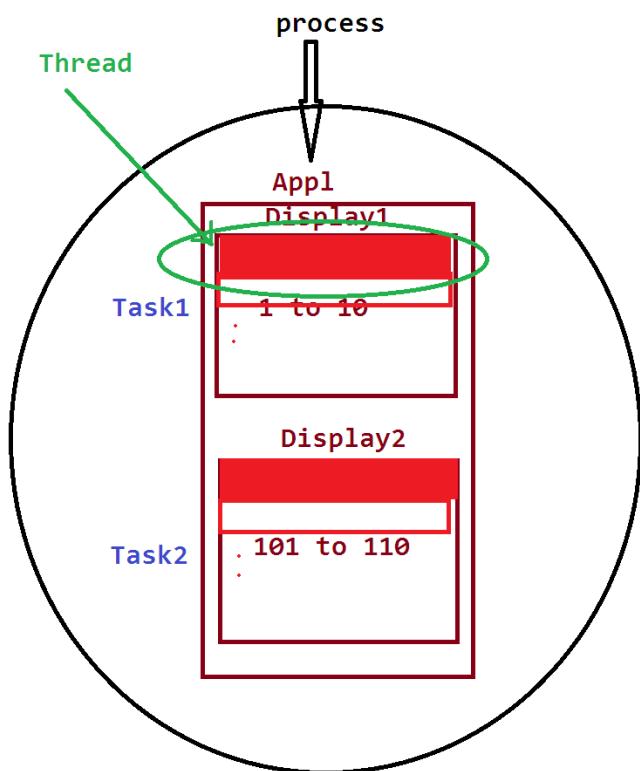
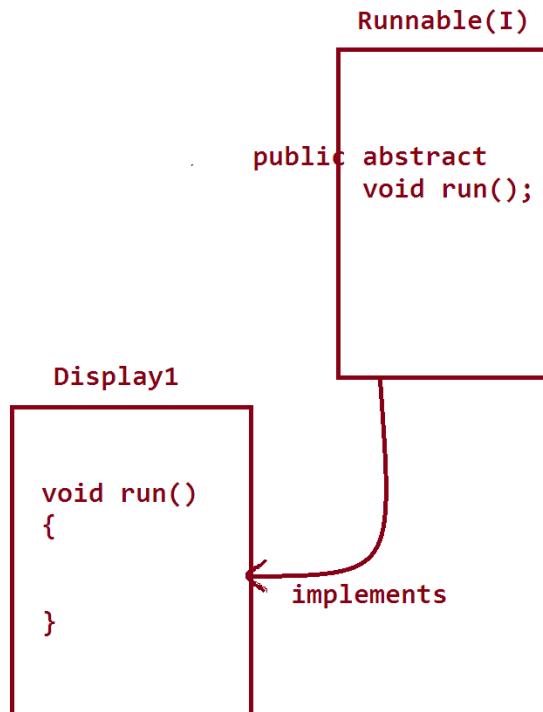
(Simultaneously means at-a-time but not parallel)

**Note:**

=>In the process of executing Multiple Tasks,some part of task is executed known as 'Thread'.

**define Thread?**

=>The part of Task is known as Thread.



Dt : 18/12/2020

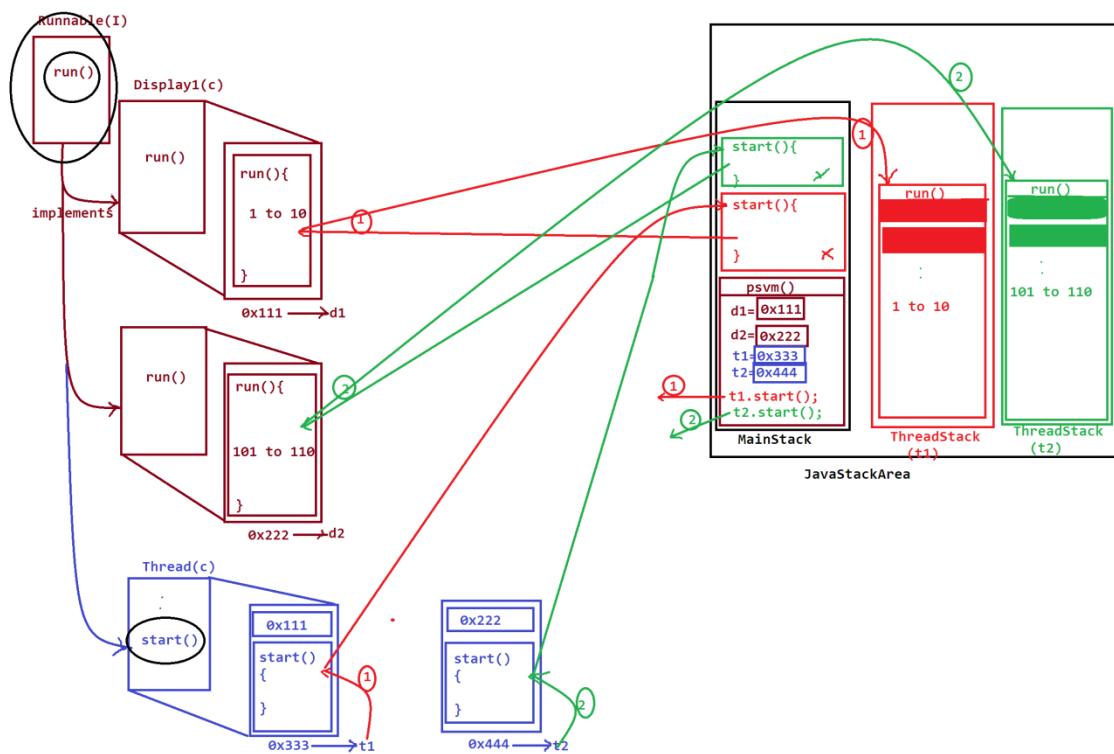
**Execution flow of above program:**

**ClassFiles:**

**Display1.class**

**Display2.Class**

**Interface3.class**



**Execution behaviour of start() method:**

=>start() method activates 'Thread Scheduler' which joins the execution control to execute threads.

=>In this process one separate thread stack is created and loaded with run() method.

=>In this way multiple thread stacks are created and all these multiple

**thread stacks are executed by 'Thread Scheduler' using TimeSlicing algorithm.**

**define TimeSlicing algorithm?**

**=>In TimeSlicing algorithm all the multiple thread stacks are executed based on define time slice.**

=====

**define Thread class?**

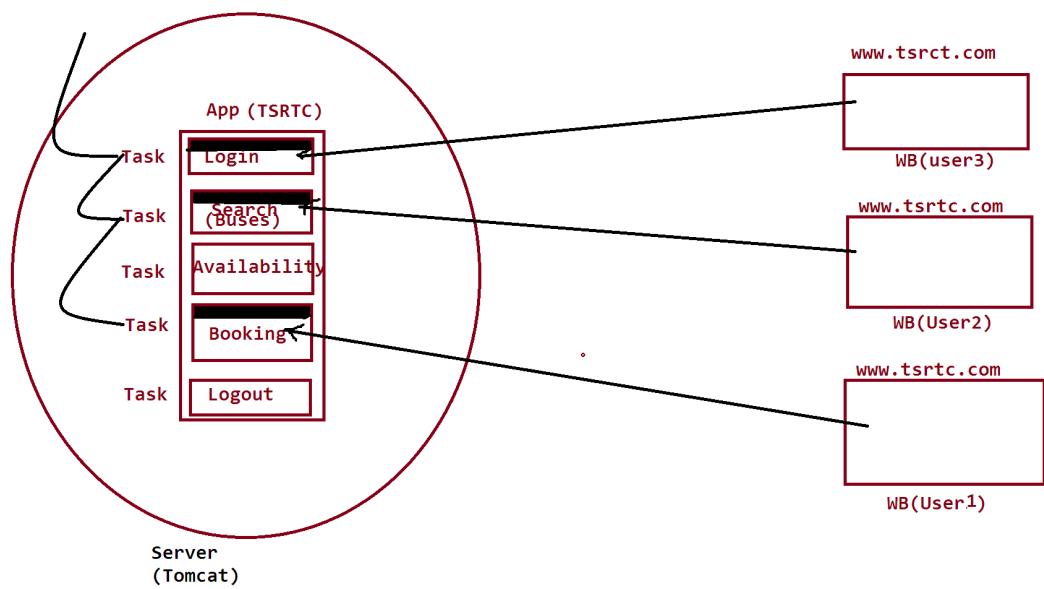
**=>Thread class is from java.lang package and which provides start() method used in thread creation.**

=====

**Note:**

**=>JavaStackArea is a collection of main() stack and thread Stacks in MultiThreading applications.**

=====



Dt : 19/12/2020

## 10.Interfaces will support Generalization process.

syntax:

```
Interface ob = new ImpClass();
```

=>In this Generalization process one reference is created and the reference is binded with all the members of Interface and only Overriding members from the ImplClass.

Exp Program:(Interface4.java)

```
import java.util.Scanner;

interface IComparable
{
    public abstract int compareTo(int x,int y);
}

class Greater implements IComparable
{
    public int compareTo(int x,int y)
    {
        if(x>y) return x;
        else return y;
    }
}

class Smaller implements IComparable
{
    public int compareTo(int x,int y)
    {
```

```
    if(x<y) return x;  
    else return y;  
}  
}  
class Interface4 //MainClass  
{  
    public static void main(String[] args)  
    {  
        Scanner s = new Scanner(System.in);  
        System.out.println("Enter the value1:");  
        int v1 = s.nextInt();  
        System.out.println("Enter the value2:");  
        int v2 = s.nextInt();  
        System.out.println("====Choice====");  
        System.out.println("1.Greater\n2.Smaller");  
        System.out.println("Enter the choice:");  
        int choice = s.nextInt();  
        switch(choice)  
        {  
            case 1: //Greater  
                IComparable gt = new Greater();//Generalization process  
                System.out.println("Greater Value:"+gt.compareTo(v1,v2));  
                break;  
            case 2: //Smaller  
                IComparable sm = new Smaller();//Generalization process  
        }
```

```
System.out.println("Smaller Value:"+sm.compareTo(v1,v2));  
        break;  
    default:  
        System.out.println("Invalid choice...");  
    } //end of switch  
}  
}
```

Note:

=>This Generalization process can be achieved by declaring implClass without name known 'Anonymous InnerClass as Implementation class'.

syntax:

```
interface Interface_name  
{  
    //Interface_body  
}  
  
Interface_name ob = new Interface_name()  
{  
    //ImplClass_body  
};
```

Ex Program:(Interface5.java)

```
import java.util.Scanner;  
interface IComparable
```

```
{  
    public abstract int compareTo(int x,int y);  
}  
  
class Interface5 //MainClass  
{  
    public static void main(String[] args)  
    {  
        Scanner s = new Scanner(System.in);  
        System.out.println("Enter the value1:");  
        int v1 = s.nextInt();  
        System.out.println("Enter the value2:");  
        int v2 = s.nextInt();  
        System.out.println("====Choice====");  
        System.out.println("1.Greater\n2.Smaller");  
        System.out.println("Enter the choice:");  
        int choice = s.nextInt();  
        switch(choice)  
        {  
            case 1: //Greater  
                IComparable gt = new IComparable()  
                {  
                    public int compareTo(int x,int y)  
                    {  
                        if(x>y) return x;  
                        else return y;  
                    }  
                };  
                System.out.println(gt.compareTo(v1,v2));  
        }  
    }  
}
```

```
    }

};

System.out.println("Greater Value:"+gt.compareTo(v1,v2));

        break;

    case 2: //Smaller

IComparable sm = new IComparable()

{

    public int compareTo(int x,int y)

    {

        if(x<y) return x;

        else return y;

    }

};


```

```
System.out.println("Smaller Value:"+sm.compareTo(v1,v2));

        break;

    default:

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

    } //end of switch

}

}
```

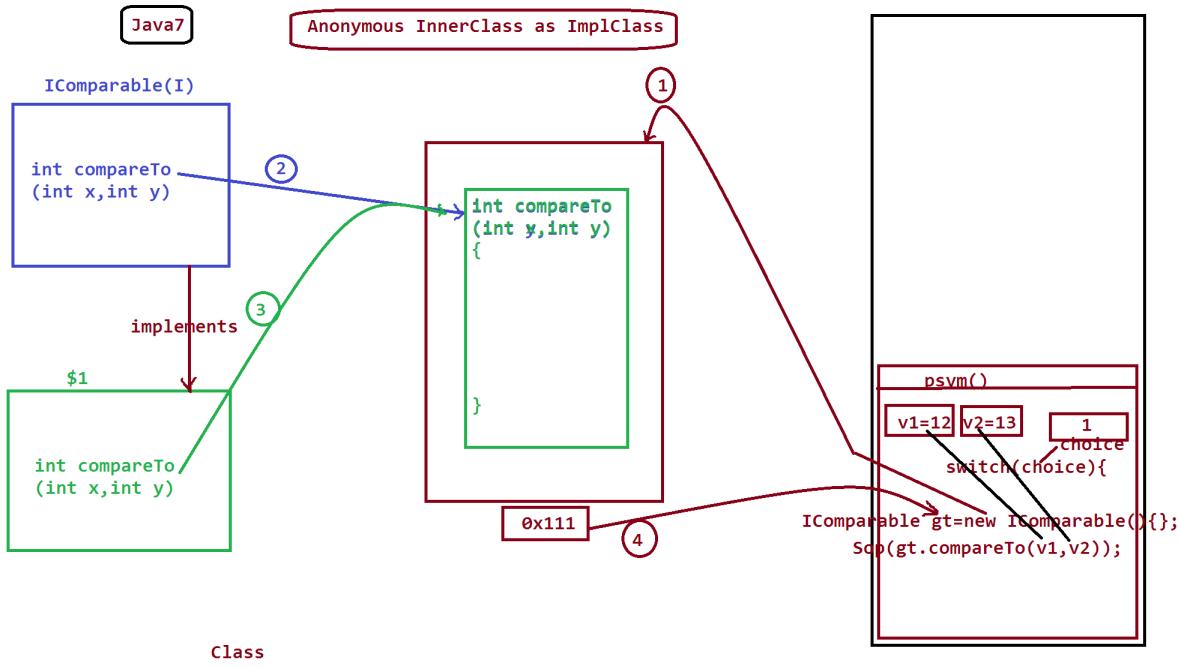
**Execution flow of above program:**

**ClassFiles:**

**IComparable.class(I)**  
**Interface5.class(MainClass)**

**Interface5\$1.class**

**Interface5\$2.class**



**Note:**

=>This 'Anonymous InnerClass as ImplClass' model is popularly used  
in MultiThreading application to create threads.

**Exp program:(Interface6.java)**

```
class Interface6 //MainClass
{
    public static void main(String[] args)
    {
        Runnable d1 = new Runnable()
```

```
{  
    public void run()  
    {  
        for(int i=1;i<=10;i++)  
        {  
            System.out.println("Val:"+i);  
            try{  
                Thread.sleep(2000);  
            }catch(Exception e){}  
            //end-of-loop  
        }  
    };
```

```
Runnable d2 = new Runnable()  
{  
    public void run()  
    {  
        for(int i=101;i<=110;i++)  
        {  
            System.out.println("Val:"+i);  
            try{  
                Thread.sleep(2000);  
            }catch(Exception e){}  
        }  
    }  
}
```

```
};

Thread t1 = new Thread(d1);
Thread t2 = new Thread(d2);

t1.start();//step5
t2.start();
```

```
}
```

**Creating thread using Anonymous InnerClass as ImpleClass of Runnable interface.**

---

**Note:**

=>By Java8 version this 'Anonymous InnerClasse as ImplClass' model is modified as 'LambdaExpression'.

**faq:**

**define LambdaExpression?**

=>The process of declaring the method without method\_name is known as LambdaExpression.which is also known as 'Anonymous method'.

**syntax:**

```
interface Interface_name
{
    public abstract return_type method_name(para_list);
}
```

```
Interface_name ob = (para_list)->
{
    //method_body
};
```

**Exp program:(Interface7.java)**

```
import java.util.Scanner;

interface IComparable
{
    public abstract int compareTo(int x,int y);
}

class Interface7 //MainClass
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        System.out.println("Enter the value1:");
        int v1 = s.nextInt();
        System.out.println("Enter the value2:");
    }
}
```

```
int v2 = s.nextInt();

System.out.println("==Choice==");

System.out.println("1.Greater\n2.Smaller");

System.out.println("Enter the choice:");

int choice = s.nextInt();

switch(choice)

{

    case 1: //Greater

IComparable gt = (int x,int y)->

{

    if(x>y) return x;

    else return y;

};

System.out.println("Greater Value:"+gt.compareTo(v1,v2));

    break;

    case 2: //Smaller

IComparable sm = (int x,int y)->

{

    if(x<y) return x;

    else return y;

};

System.out.println("Smaller Value:"+sm.compareTo(v1,v2));

    break;

    default:

System.out.println("Invalid choice...");
```

```

    } //end of switch
}

}

```

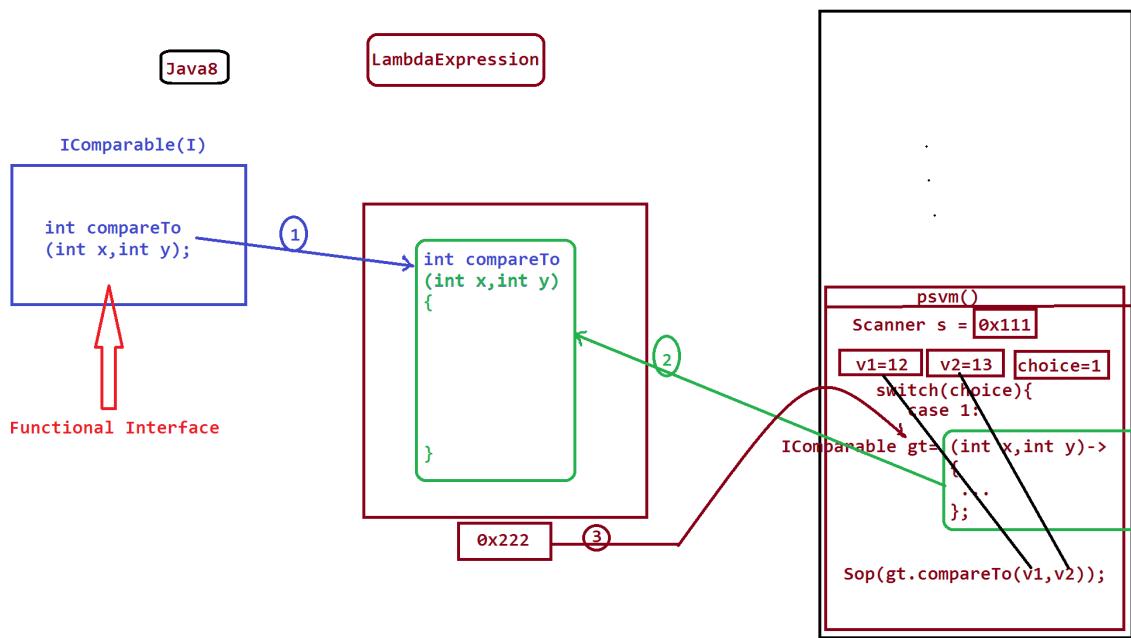
Dt : 20/12/2020(sunday 8:00 AM)

**Execution flow of above program:**

**ClassFiles:**

**IComparable.class(I)**

**Interface7.class(MainClass)**



**Execution behaviour of LambdaExpression:**

=>when execution control finds LambdaExpression one reference is created and the reference is binded with the abstract method signature and which is attached with LambdaExpression or Anonymous method body.

**faq:**

**define Functional Interface?**

=>The Interface which is declared with only one abstract method is known as Functional Interface.

**Note:**

=>These Functional Interfaces will provide abstract method signature to hold LambdaExpression.

**faq:**

**define Normal Interface?**

=>The interface which can be declared with any number of abstract methods is known as Normal Interface.

**Coding Rules:**

**Rule-1 : The Interface must be Functional Interface**

**Rule-2 : The LambdaExpressions will access the variables of Functional Interface using Interface\_name.**

**Rule-3 : We must not declare same variable names in LambdaExpression and within the same method\_scope.**

**Advantage of LambdaExpression:**

=>when we use LambdaExpression then separate (.class) files are not generated, when there are no separate class files then loading time of execution process is saved and generates HighPerformance.

---

**Note:**

=>Constructing threads using 'LambdaExpression'.

=>In this model the run() method is declared without name,which means as 'LambdaExpression'.

**Exp program:(Interface8.java)**

```
class Interface8 //MainClass
{
    public static void main(String[] args)
    {
        Runnable d1 = ()->
        {
            for(int i=1;i<=10;i++)
            {
                System.out.println("Val:"+i);
                try{
                    Thread.sleep(2000);
                }catch(Exception e){}
            }//end-of-loop
        };
    }
}
```

**Runnable d2 = ()->**

```
{
    for(int i=101;i<=110;i++)
```

```
{  
System.out.println("Val:"+i);  
try{  
    Thread.sleep(2000);  
}catch(Exception e){}  
};  
  
Thread t1 = new Thread(d1);  
Thread t2 = new Thread(d2);
```

t1.start(); //step5

t2.start();

```
}  
}
```

---

\*imp

**Method References in Java:**

=>The process in which abstract method signature of Functional interface will hold the body of method of a class,in which the class is not linked to Interface is known as 'Method References Concept'.

=>These Method references are categorized into three types:

- (a)Reference to Constructor
- (b)Reference to Instance method
- (c)Reference to static method

**(a)Reference to Constructor:**

=>The process in which the abstract method signature is binded with the body of Constructor is known as "Reference to constructor".

**syntax:**

`Interface_name ob = Class_name::new;`

**(b)Reference to Instance method:**

=>The process in which the abstract method signature is binded with the body of Instance method is known as "Reference to Instance method".

**syntax:**

`Interface_name ob = obj_name::method_name;`

**(c)Reference to static method:**

=>The process in which the abstract method signature is binded with the body of static method is known as "Reference to static method".

**syntax:**

**Interface\_name ob = Class\_name::method\_name;**

**Dt : 21/12/2020**

**Exp program:**

```
interface ITest //Functional Interface
{
    public abstract void m(int k);
}

class Display //SubClass
{
    Display(int x)
    {
        System.out.println("==>Constructor==>");
        System.out.println("The value x:" + x);
    }

    void dis1(int y)
    {
        System.out.println("==>Instance method==>");
        System.out.println("The value y:" + y);
    }

    static void dis2(int z)
    {
        System.out.println("==>Static method==>");
    }
}
```

```
System.out.println("The value z:"+z);
}

}

class Interface9 //MainClass
{
    public static void main(String[] args)
    {
        ITest ob1 = Display::new;//Reference to Constructor
        ob1.m(101);

        Display d = new Display(1010);

        ITest ob2 = d::dis1; //Reference to Instance method
        ob2.m(102);

        ITest ob3 = Display::dis2;//Reference to Static method
        ob3.m(103);
    }
}
```

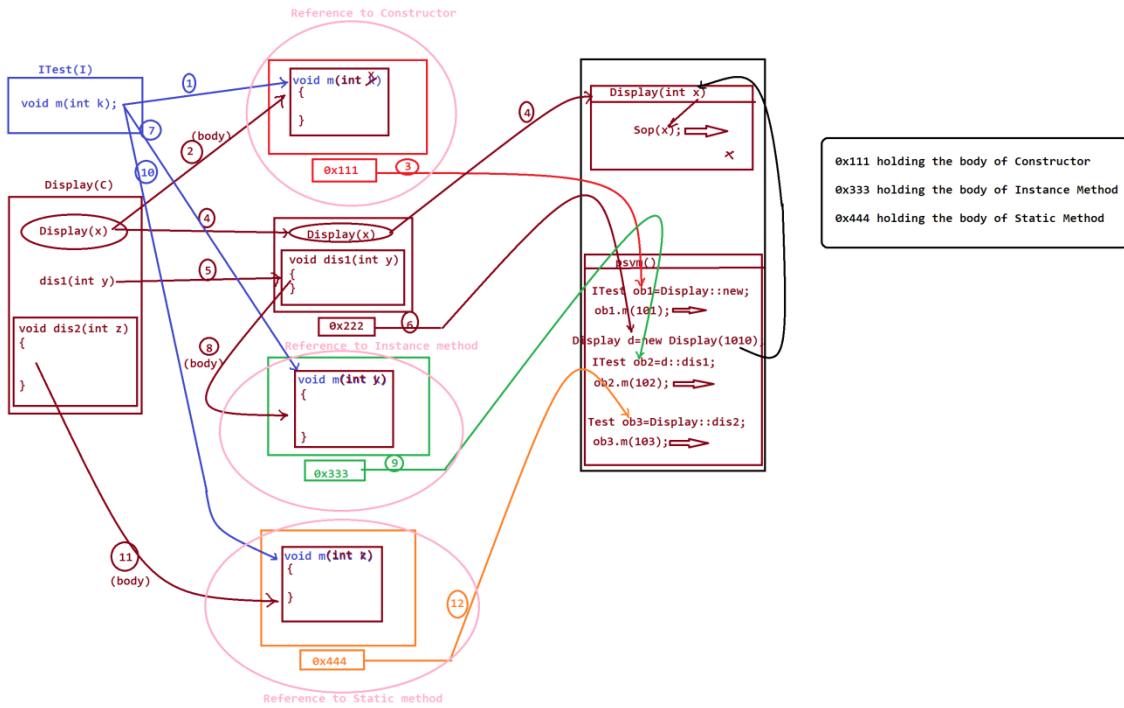
**Execution flow of above program:**

**ClassFiles:**

**ITest.class(I)**

**Display.class**

**Interface9.class(MainClass)**



## Summary:

=>The following are the ways to provide method\_body to abstract method signature of Interface:

### Model-1 : Using 'Interface and ImplClass names'

**Interface2.java**

**Interface3.java(Thread)**

### Model-2 : Using 'Interface and ImplClasses without name'

(Anonymous InnerClass as ImplClass)

**Interface5.java**

**Interface6.java(Thread)**

### Model-3 : Using 'Functional Interface and LambdaExpression'

**Interface7.java**

**Interface8.java(Thread)**

#### **Model-4 : Using 'Functional Interface and Method references'**

**Interface9.java**

---

#### **11.Interface can be declared with Concrete methods**

**=>The following are the concrete methods can be declared part of**

**Interfaces:**

**(i)default concrete methods(Java8)**

**(ii)static Concrete methods(Java8)**

**(iii)private Concrete methods(Java9)**

**Dt : 23/12/2020**

**(i)default concrete methods(Java8):**

**=>From Java8 version onwards the interface can be declared with  
defualt concrete methods.(The concrete methods declared with default  
keyword)**

**Coding rule:**

**=>These default concrete methods are available to ImplClasses and  
can be accessed with the ImplClass object references.**

**Exp program:**

```
interface ITest11
{
    default void m(int x)
    {
        System.out.println("==default method==");
        System.out.println("The value x:" + x);
    }
}
```

```
class Interface10 //MainClass
```

```
{
```

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

```
{
```

```
    ITest11 ob = new ITest11(){};
```

```
    ob.m(123);
```

```
}
```

```
}
```

**(ii)static Concrete methods(Java8):**

=>From Java8 version onwards the interface can be declared with static concrete methods.(The concrete methods declared with static keyword)

**Coding rule:**

=>These static concrete methods are binded to interfaces while interface loading and accessed with interface\_name.

=>These static concrete methods are not available to implClasses and and cannot be accessed with ImplClass names.

Exp program:

```
interface ITest22
{
    static void m(int x)
    {
        System.out.println("====static method====");
        System.out.println("The value x:"+x);
    }
}

class IClass implements ITest22
{
}

class Interface11 //MainClass
```

```
{
    public static void main(String[] args)
    {
        IClass.m(103);
        ITest22.m(102);
    }
}
```

(iii)private Concrete methods(Java9):

=>From Java9 version onwards the Interface can be declared with

**private concrete methods.**

=>These private concrete methods are categorized into two types:

(a)static private concrete methods

(b)NonStatic private concrete methods

**Coding Rule:**

=>**Private Concrete methods are accessed by the NonPrivate concrete methods of same Interface.**

**Exp program:**

```
interface ITest33
{
    private static void m1(int x)
    {
        System.out.println("==private static method==");
        System.out.println("The value x:" + x);
    }

    private void m2(int y)
    {
        System.out.println("==private NonStatic method==");
        System.out.println("The value y:" + y);
    }

    default void dis(int x,int y)
    {
        ITest33.m1(x);
        this.m2(y);
    }
}
```

```
}

class Interface12 //MainClass

{

    public static void main(String[] args)

    {

ITest33 ob = new ITest33(){};

ob.dis(101,102);

    }

}
```

**Note:**

=>Execute above program on Java9 version and above...

**Dt : 26/12/2020**

**12.There is no concept of declaring Blocks and Constructors part of Interfaces.(Compilation Error)**

**13.Interface can use the members of another interface using 'extends' keyword.**

**Exp program:**

```
interface ITest1

{

    public static final int k=20;

}
```

```
interface ITest2 extends ITest1
```

```
{  
    public static final int z=30;  
}  
  
class Interface13 //MainClass  
{  
    public static void main(String[] args)  
    {
```

```
        System.out.println("The value k:"+ITest2.k);  
        System.out.println("The value z:"+ITest2.z);  
    }  
}
```

=====

\*imp

### **Abstract Classes in Java:**

=>The class which is declared with abstract keyword is known as abstract class.

=>These abstract classes can be declared with both abstract methods and concrete methods.

=>The abstract methods part of abstract classes must be declared with abstract keyword.

=>Abstract classes in Java are abstract components and which cannot be instantiated.

=>These abstract classes are extended to normal classes to have method\_bodies to abstract method signatures,

**Exp program:**

```
import java.util.Scanner;

abstract class AClass

{
    double z,x;

    AClass(double x)
    {
        this.x=x;
    }

    void cal()
    {
        z = Math.sqrt(x);
    }

    abstract void dis();
}

class Display extends AClass

{
    Display(double x)
    {
        super(x);
    }

    void dis()
    {

        System.out.println("Sqrt of "+x+" is "+z);
    }
}
```

```
}
```

```
class ABClass //MainClass
```

```
{
```

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

```
    {
```

```
//AClass ob = new AClass();//Compilation Error
```

```
Scanner s = new Scanner(System.in);
```

```
System.out.println("Enter the value of x:");
```

```
double x = s.nextDouble();
```

```
Display d = new Display(x);
```

```
d.cal();
```

```
d.dis();
```

```
}
```

```
}
```

---

**faq:**

**wt is the diff b/w**

**(i)class**

**(ii)abstract class**

**=>Class can be declared with only concrete methods, but abstract class  
can be declared with both abstract methods and concrete methods.**

**=>Classes can be instantiated, but abstract classes cannot be  
instantiated.**

**faq:**

**wt is the diff b/w**

**(i)Interface**

**(ii)abstract class**

**=>Both Components are abstract components and cannot be instantiated,  
but abstract classes can be declared with 'blocks and constructors'  
and Interfaces cannot be declared with 'blocks and constructors' .**

---

**Note:**

**=>In realtime abstract classes are less used when compared to  
Interfaces.**

---

Dt : 28/12/2020

**Packages in Java:**

=>Package is a collection of classes and Interfaces.

=>The packages are categorized into two types:

1.Built-In packages

2.User Defined packages

**1.Built-In packages:**

=>The packages which are available from JavaLib are known as BiultIn packages.

The following are some important Built-In packages from JavaLib:

**java.lang - Language package(default package)**

**java.io - IO Stream and Files package**

**java.net - Networking package**

**java.util - Utility package**

**java.awt - AbstractWindowToolkit package(GUI)**

**javax.swing - Swing package(GUI)**

**java.applet - Applet programming package(GUI)**

**java.sql - DB Connection package**

**javax.servlet - Servlet programming package**

**javax.servlet.jsp - JSP Programming package**

---

\*imp

## 2.User Defined packages:

=>The package which is defined by the programmer is known as User defined package.

=>we use 'package' keyword to define package

**syntax:**

```
package package_name;
```

**Note:**

=>The classes,Interfaces and its members which are declared within the package must be 'public'.

**Exp program:**

```
package p1;  
public class Salary  
{  
    public float totSal;  
    public void cal(int bSal)  
    {  
        totSal = bSal+(0.96F*bSal)+(0.66F*bSal);  
    }  
}
```

```
public void getTotSal()
{
    System.out.println("TotSal"+totSal);
}

}
```

**step1 : Save the program in Project Folder as**

**Salary.java**

**step2 : Compile the program as**

**javac -d . Salary.java**

**Note:**

**(-d .) indicates create package and load the package with class file.**

---

```
-----
package p2;

import java.util.Scanner;

import p1.Salary;

public class SMainClass
{
    public static void main(String[] args)
    {
        Scanner s = new Scanner(System.in);
        Salary s1 = new Salary();
        System.out.println("Enter the bSal:");
    }
}
```

```

int bSal = s.nextInt();

s1.cal(bSal);

s1.getTotSal();

}

}

```

**step1 : Save the program in Program in Project folder as**

**SMainClass.java**

**step2 : Compile the program as**

**javac -d . SMainClass.java**

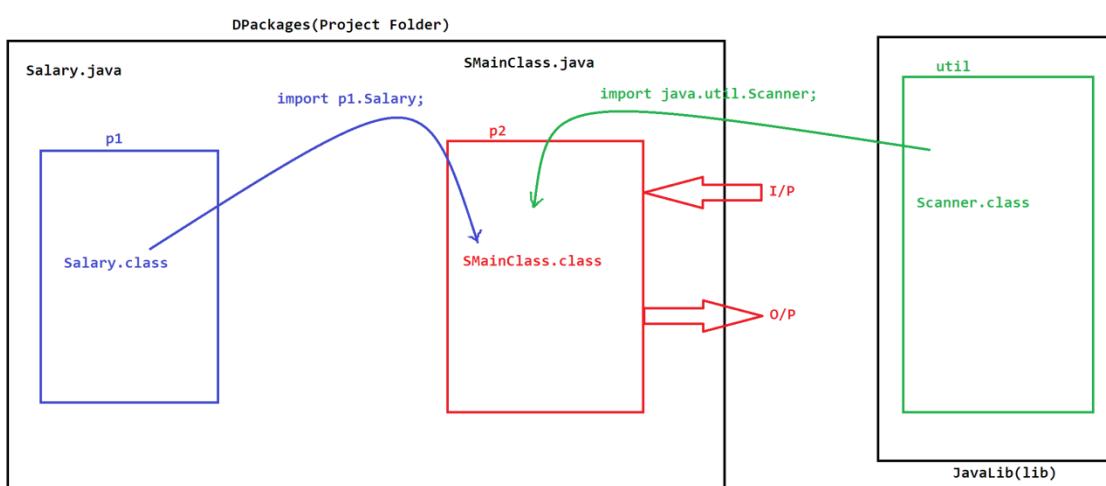
**step3 : Execute the program as**

**java p2.SMainClass**

---



---



Dt : 29/12/2020

**define 'import' statement?**

=>'import' statement is used when we want to make one class/interface available to another class/interface,in which the classes and interfaces are available in different packages.

=>This importing process is categorized into three types:

**(i)Using 'import pack\_name.CName/IName;'**

=>In this importing process we specify the CName or IName to be imported,which is also known as 'Explicit importing process'.

**Exp:**

```
import java.util.Scanner;
```

```
import p1.Salary;
```

**(ii)Using 'import pack\_name.\*;'**

=>In this importing process all the classes and Interfaces of the package are imported,which is also known as 'Implicit importing process'

**Exp:**

```
import java.util.*;
```

```
import p1.*;
```

**Note:**

=>This importing process is used when we want to import more number of classes or interfaces from the same package.

### **(iii)Using 'Fully Qualified names'**

**=>'Fully qualified names' means the classes and Interfaces are declared with package names part of programming code.**

**Exp:**

```
java.util.Scanner s = new java.util.Scanner(System.in);  
p1.Salary s1 = new p1.Salary();
```

---

**faq:**

**define 'static' import?**

**=>The 'import' statement which is declared with 'static' keyword is known as 'static' import.**

**syntax:**

```
import static pack_name.CName/IName.*;
```

**Note:**

**=>when we use 'static import' then all the static members of Class/Interface can be declared part of program without class\_name or Interface\_name.**

**define sqrt() method?**

**=>sqrt() method is used to calculate the sqrt of given number.**

**=>sqrt() method is a static method from 'java.lang.Math' class.**

**Method Signature of sqrt():**

```
public static double sqrt(double);
```

**syntax:**

```
double d = Math.sqrt(num);
```

**Exp program:**

```
package p2;  
import java.util.Scanner;  
import static java.lang.Math.*;  
public class DImport //MainClass  
{  
    public static void main(String[] args)  
    {  
        Scanner s = new Scanner(System.in);  
        System.out.println("Enter the value:");  
  
        double x = s.nextDouble();  
        double d = sqrt(x);  
        System.out.println("SQRT of "+x+" is "+d);  
    }  
}
```

**Note:**

**=>This 'static import' is introduced by Java5 version.**

---

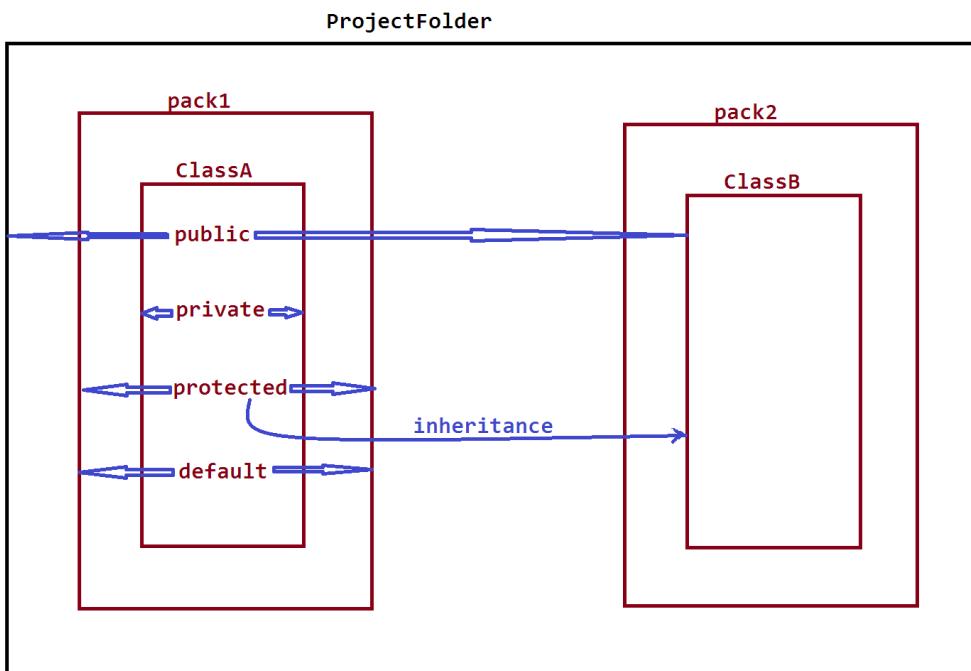
**faq:**

**define Access Modifiers?**

=>Access Modifiers specify the visibility of programming components within the project folder.

=>The following are some important Access Modifiers:

- 1.**public**
- 2.**private**
- 3.**protected**
- 4.**default**



**1.public:**

=>**public** accessed within the project folder.

**2.private:**

=>private accessed within the class.

**3.protected:**

=>protected accessed within the package, but protected can also be accessed by the CClass outside the package.

**4.default:**

=>default accessed within the package.

**Note:**

=>The programming components which are declared within the class without any access modifier are known as default programming components.

---

\*imp

**Constructing Java Project using IDE Eclipse:**  
**(IDE-Integrated Development Environment)**

**step1 :Open IDE Eclipse,while opening name the WorkSpace(folder) and click ok.**

**step2 : create Java Project**

**Click in File->new->Project->Java->select JavaProject and click next-> name the project and click 'finish'.**

### **step3 : Create packages**

**Right click on 'src'->new->package, name the package and click 'finish'**

### **step4 : Create Classes and Interfaces**

**Right Click on package->new->class, name the class and click 'finish'**

**Right Click on package->new->Interface, name the interface and click 'finish'.**

#### **Note:**

**To increase TextFont, click on Window->Preferences->General->Appearance->Colors and Fonts->Java->Java Editor Text font...**

### **Step5 : Execute the program**

**press 'ctrl+f11'**

---

#### **Note:**

**press 'ctrl+m' to Max and Min**

---

---

**==**

Dt : 30/12/2020

### **InnerClasses in Java:**

=>The process of declaring the class within the class is known as **InnerClass** or **NestedClass**.

=>These InnerClasses are categorized into the following:

- 1.Member InnerClasses**
- 2.Local InnerClasses**
- 3.Anonymous InnerClasses**

#### **1.Member InnerClasses:**

=>The InnerClasses which are declared as members of the class are known as **MemberInnerClasses**.

=>These MemberInnerClasses are categorized into the following:

- (a) **NonStatic Member InnerClasses**
- (b) **Static Member InnerClasses**

#### **(a)NonStatic Member InnerClasses:**

=>The member InnerClasses which are declared without static keyword are known as **NonStatic Member InnerClasses**.

### **Coding Rules:**

=>**NonStatic member InnerClasses can be declared with only NonStatic members.**

=>**The methods of NonStatic member InnerClasses can access all the members of OuterClass directly.**

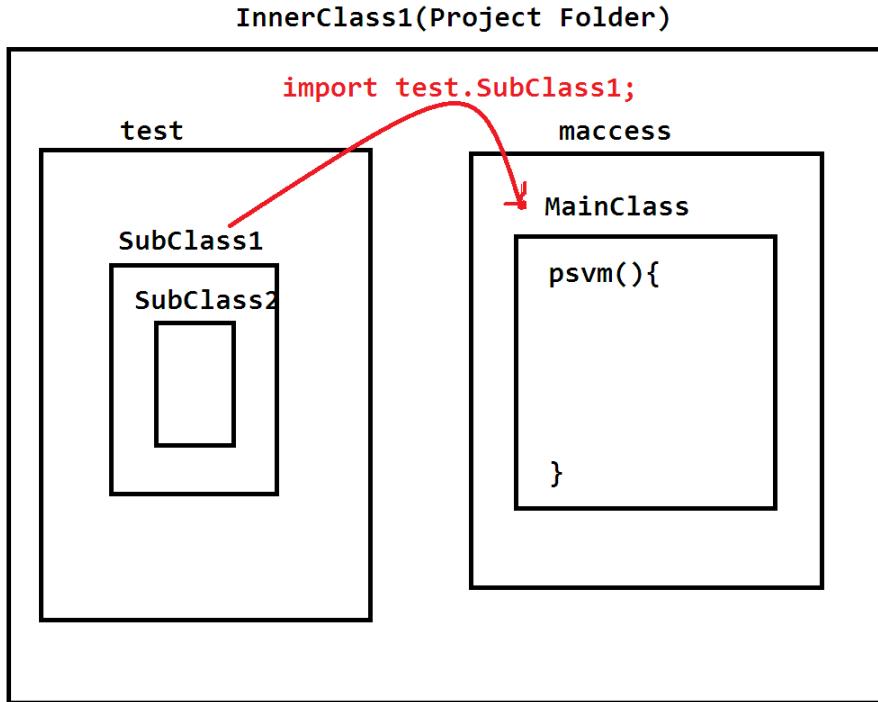
### **Exp program:**

### SubClass1.java

```
package test;
public class SubClass1 {
    public int a=10;
    public static int b=20;
    public class SubClass2{
        public void m2(){
System.out.println("====InnerClass m2()====");
System.out.println("The value a:"+a);
System.out.println("The value b:"+b);
    }
    } //InnerClass
} //OuterClass
```

### MainClass.java

```
package maccess;
import test.SubClass1;
public class MainClass {
    public static void main(String[] args) {
SubClass1 ob1 = new SubClass1(); //OuterClass object
SubClass1.SubClass2 ob2 = ob1.new SubClass2(); //InnerClass
Object
ob2.m2();
    }
}
```



### (b) Static Member InnerClasses:

=>The member InnerClasses which are declared with static keyword are known as **Static Member InnerClasses**.

#### Coding Rules:

=>Static member InnerClasses can be declared with both static and NonStatic members.

=>The methods of Static Member InnerClasses can access static members of OuterClass directly, but the NonStatic members of OuterClass are accessed with the OuterClass object reference.

#### Exp program:

### SubClass11.java

```
package test;
public class SubClass11 {
    public int a=10;
    public static int b=20;
    public static SubClass11 ob;
    public static class SubClass22{
        public void m1(){
            System.out.println("====InnerClass m1()====");
            System.out.println("The value a:"+ob.a);
            System.out.println("The value b:"+b);
        }
        public static void m2(){
            System.out.println("====InnerClass m2()====");
            System.out.println("The value a:"+ob.a);
            System.out.println("The value b:"+b);
        }
    }//InnerClass
}//OuterClass
```

### MainClass.java

```
package maccess;
import test.SubClass11;
public class MainClass {
    public static void main(String[] args) {
        SubClass11.ob = new SubClass11(); //OuterClass object
        SubClass11.SubClass22 ob2 = new
        SubClass11.SubClass22(); //InnerClass object
        ob2.m1();
        SubClass11.SubClass22.m2();
    }
}
```

Dt : 22/1/2021

## 2.Local InnerClasses:

=>The NonStatic member InnerClass which is declared inside the method of OuterClass is known as Local InnerClass.

Coding rules:

=>The Class declaration, Object creation and accessing methods must be declared inside the methods of OuterClass.

Exp program:

```
package test;
public class SubClass111 {
    public void m1(){
        class SubClass222{
            public void m2(){
                System.out.println("==Display m2()==");
            }
        }//LocalInnerClass
        SubClass222 ob2 = new SubClass222(); //InnerClass object
        ob2.m2();
    }//OuterClass methods
}//OuterClass
```

```
package maccess;
import test.SubClass111;
public class MainClass {
    public static void main(String[] args) {
        SubClass111 ob1 = new SubClass111(); //OuterClass object
        ob1.m1();}}
```

Note:

=>while OuterClass method execution, Local innerClass is loaded,  
Object created and method accessed.

---

### 3. Anonymous InnerClasses:

=>The InnerClasses which are declared without name are known as  
Anonymous InnerClasses.

=>Anonymous InnerClasses are categorized into the following:

- (a)Anonymous InnerClass as Class extention
- (b)Anonymous InnerClass as Implementation class

#### (a)Anonymous InnerClass as Class extention:

=>The process of declaring the CClass without name is known as  
"Anonymous InnerClass as Class extention".

syntax:

```
class PClass
{
    //PClass_body
}

PClass ob = new PClass()
{
    //CClass_body
};
```

Exp program:

## **Inheritance5.java**

**(b)Anonymous InnerClass as Implementation class:**

=>The process of declaring the ImplClass without name is known as "Anonymous InnerClass as Implementation class".

**syntax:**

```
interface Interface_name  
{  
    //Interface_body  
}  
  
Interface_name ob = new Interface_name()  
{  
    //ImplClass_body  
};
```

**Exp program:**

## **Interface4.java**

```
=====
```

\*imp

**Anonymous InnerClass as Method Argument:**

=>The process of passing Anonymous InnerClass as parameter to the method is known as "Anonymous InnerClass as Method Argument".

**Exp program:**

## **IComparable.java**

```
package test;
public interface IComparable {
    public abstract int compareTo(int x,int y);
}
```

### Result.java

```
package test;
public class Result {
    public void getResult(int x,int y,IComparable ob){
        System.out.println("Result:"+ob.compareTo(x, y));
    }
}
```

### MainClass.java

```
package maccess;

import test.IComparable;

import test.Result;

import java.util.Scanner;

public class MainClass {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        System.out.println("Enter the value1:");

        int v1 = s.nextInt();

        System.out.println("Enter the value2:");

        int v2 = s.nextInt();

        System.out.println("====Choice====");

        System.out.println("1.Greater\n2.Smaller");
```

```
System.out.println("Enter the choice:");

int choice = s.nextInt();

switch(choice){

    case 1:

new Result().getResult(v1,v2,new IComparable()

    {

        public int compareTo(int x,int y){

            if(x>y) return x;

            else return y;

        }

    });

    break;

    case 2:

new Result().getResult(v1,v2, new IComparable()

    {

        public int compareTo(int x,int y)

        {

            if(x<y) return x;

            else return y;

        }

    });

    break;

default:

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

}

//end of switch
```

```
s.close();  
}  
}  
=====
```

**Note:**

=>From Java8 version onwards "Anonymous InnerClass as ImplClass" model is modified as "LambdaExpression".

\*imp

**LambdaExpression as Method Argument:**

=>The process of passing "LambdaExpression" as parameter to the method is known as "LambdaExpression as Method Argument".

**Exp program:**

IComparable.java

```
package test;  
public interface IComparable {  
    public abstract int compareTo(int x,int y);  
}
```

Result.java

```
package test;  
public class Result {  
    public void getResult(int x,int y,IComparable ob){  
        System.out.println("Result:"+ob.compareTo(x, y));  
    }  
}
```

```
}
```

### MainClass.java

```
package maccess;

import test.Result;

import java.util.Scanner;

public class MainClass {

    public static void main(String[] args) {

        Scanner s = new Scanner(System.in);

        System.out.println("Enter the value1:");

        int v1 = s.nextInt();

        System.out.println("Enter the value2:");

        int v2 = s.nextInt();

        System.out.println("====Choice====");

        System.out.println("1.Greater\n2.Smaller");

        System.out.println("Enter the choice:");

        int choice = s.nextInt();

        switch(choice){

            case 1:

                new Result().getResult(v1,v2,(int x,int y)->

                {

                    if(x>y) return x;

                    else return y;

                });

                break;
        }
    }
}
```

```
case 2:  
  
new Result().getResult(v1,v2, (int x,int y)->  
{  
    if(x<y) return x;  
    else return y;  
});  
break;  
  
default:  
  
System.out.println("Invalid choice...");  
}  
//end of switch  
s.close();  
}  
}
```

---

dt : 25/1/2021

Execution flow of above program:

(Anonymous InnerClass as method argument)

ClassFiles:

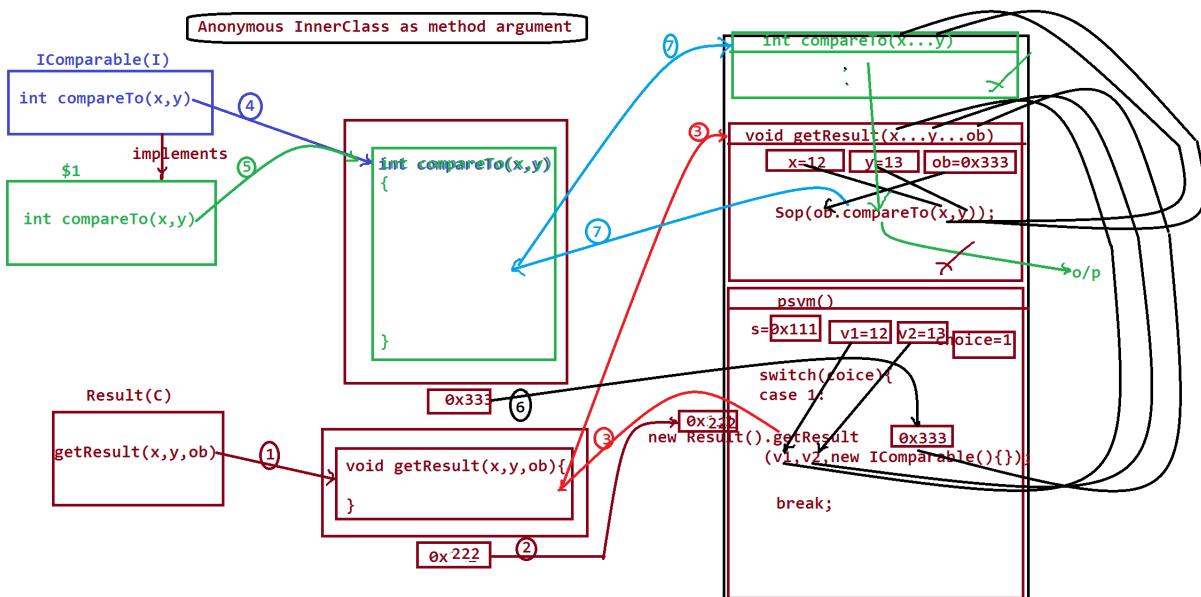
IComparable.class(I)

Result.class

MainClass.class(MainClass)

## MainClass\$1.class

## MainClass\$2.class



Note:

=>when we pass Anonymous InnerClass as Method argument,then one object is created and the reference of the object is passed as parameter to the method

---

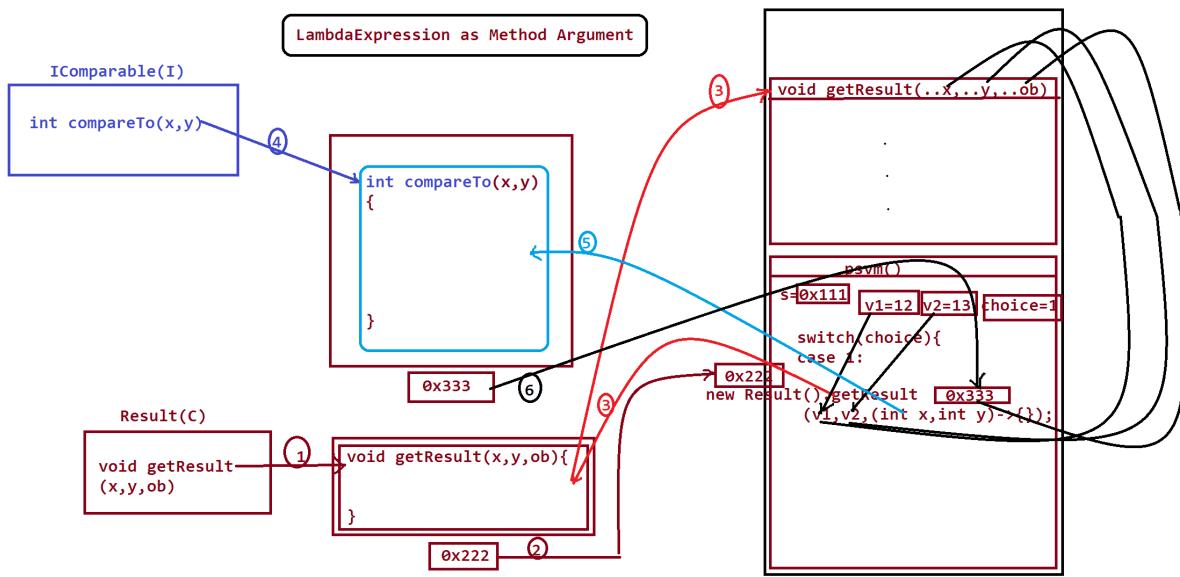
Execution flow of above program:(LambdaExpression as method argument)

ClassFiles:

**IComparable.class(I)**

**Result.class**

**MainClass.class(MainClass)**



**Note:**

=>when we pass LambdaExpression as Method Argument,the one object is created and the reference of an object is passed as parameter to the method.

---

=====

**InnerClass within the Interface:**

=>we can also declare InnerClass within the Interface and which is automatically static member InnerClass.

**InnerClass within the AbstractClass:**

=>We can also declare InnerClass within the Abstract class and which can be declared as static or NonStatic.

Exp program:

ITest.java

```
package test;
public interface ITest {
    public static class SubClass2{
        public void m2(){}
    System.out.println("==>Interface InnerClass
m2()====");
    }
}//InnerClass
}//OuterInterface
```

AClass.java

```
package test;
public abstract class AClass {
    public class SubClass1{
        public void m1(){}
    System.out.println("==>AClass InnerClass
m1()====");
    }
}//InnerClass
    public static class SubClass2{
        public void m2(){}
    System.out.println("==>AClass InnerClass m2()====");

    }
}//InnerClass
}//OuterClass
```

MainClass.java

```
package maccess;
```

```
import test.ITest;
import test.AClass;
public class MainClass {
    public static void main(String[] args) {
ITest.SubClass2 ob1 = new ITest.SubClass2();
ob1.m2();
AClass ob = new AClass() {};
AClass.SubClass1 ob2 = ob.new SubClass1();
ob2.m1();
AClass.SubClass2 ob3 = new AClass.SubClass2();
ob3.m2();
    }
}
=====
==
```



Dt:27/1/2021

Note:

=>static classes, static Interfaces and static Abstract classes can be declared as only inner components.

=>which means,

static class as InnerClass

static Interfaces as InnerInterface

static AbstractClass as InnerAbstractClass

=====

=

\*imp

**Exception Handling in Java:**

define Exception?

=>The disturbance which is occurred from the application is known as Exception.

define Exception Handling process?

=>The process which is used to handle the exception is known as Exception handling process.

=>The following are the blocks used in Exception handling process:

(a)try

(b)catch

(c)finally

(a)try :

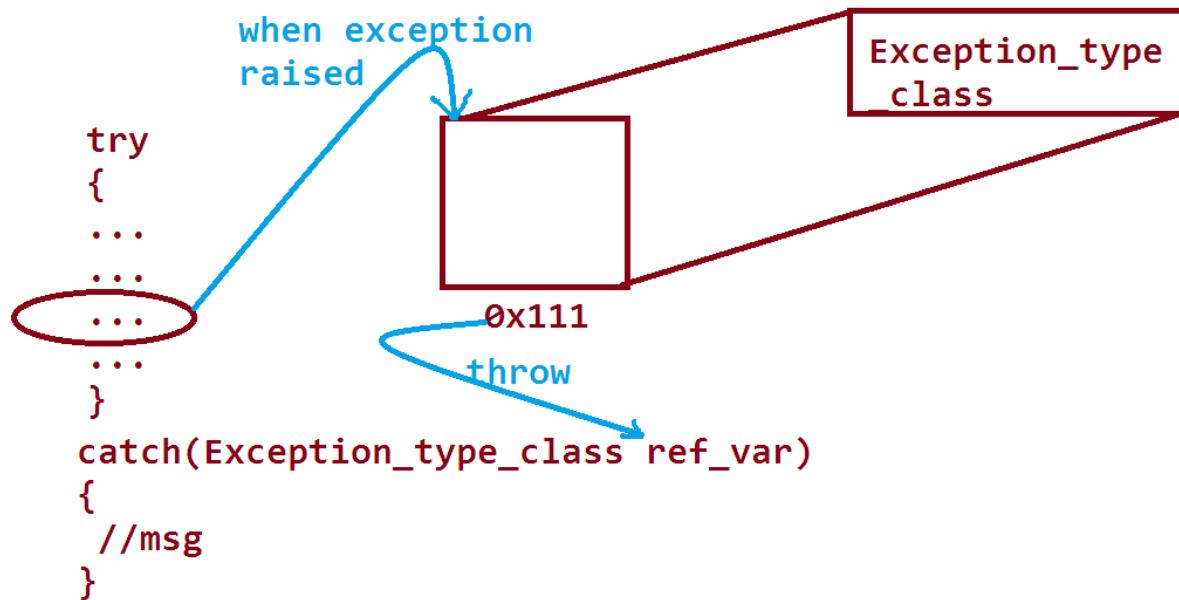
=>'try' block will hold the statements which are going to raise the exception.

**syntax:**

```
try
{
    //set-of-statements;
}
```

**Execution behaviour of try block:**

=>when the exception raises within the 'try' block,then one object is created for **Exception\_type\_class** and the reference of object is thrown onto **catch\_block**.



**(b) catch:**

=>The catch block will hold the reference of object thrown from the try block and the required msg is generated from the catch block.

**syntax:**

```
catch(Exception_type_class ref_var)
{
    //msg
}
```

**(c)finally :**

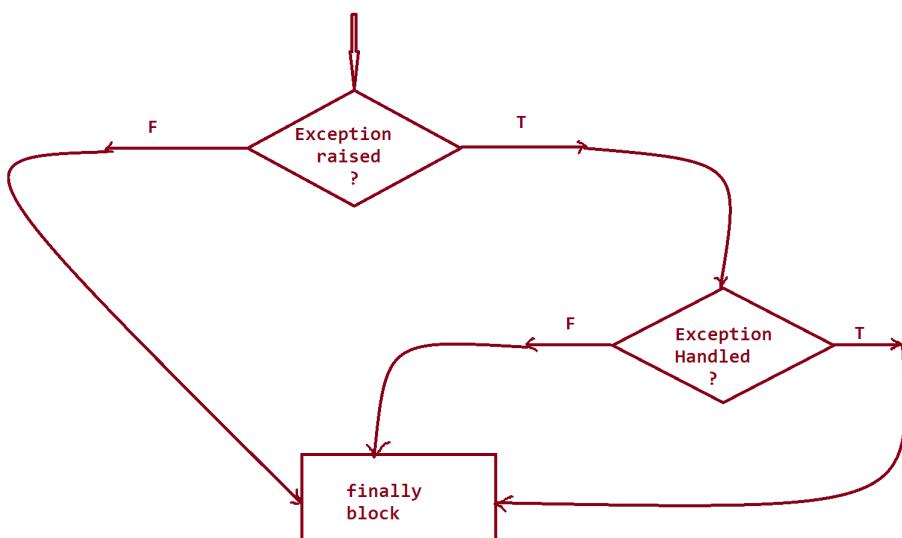
=>'finally' block is part of exception handling process, but executed independently without depending on exception.

**syntax:**

```
finally
{
    //set-of-statements;
}
```

**Note:**

=> In realtime finally block will hold resource closing operations like  
IO Close, File close, DB Close, ...



\*imp

define "Throwable"?

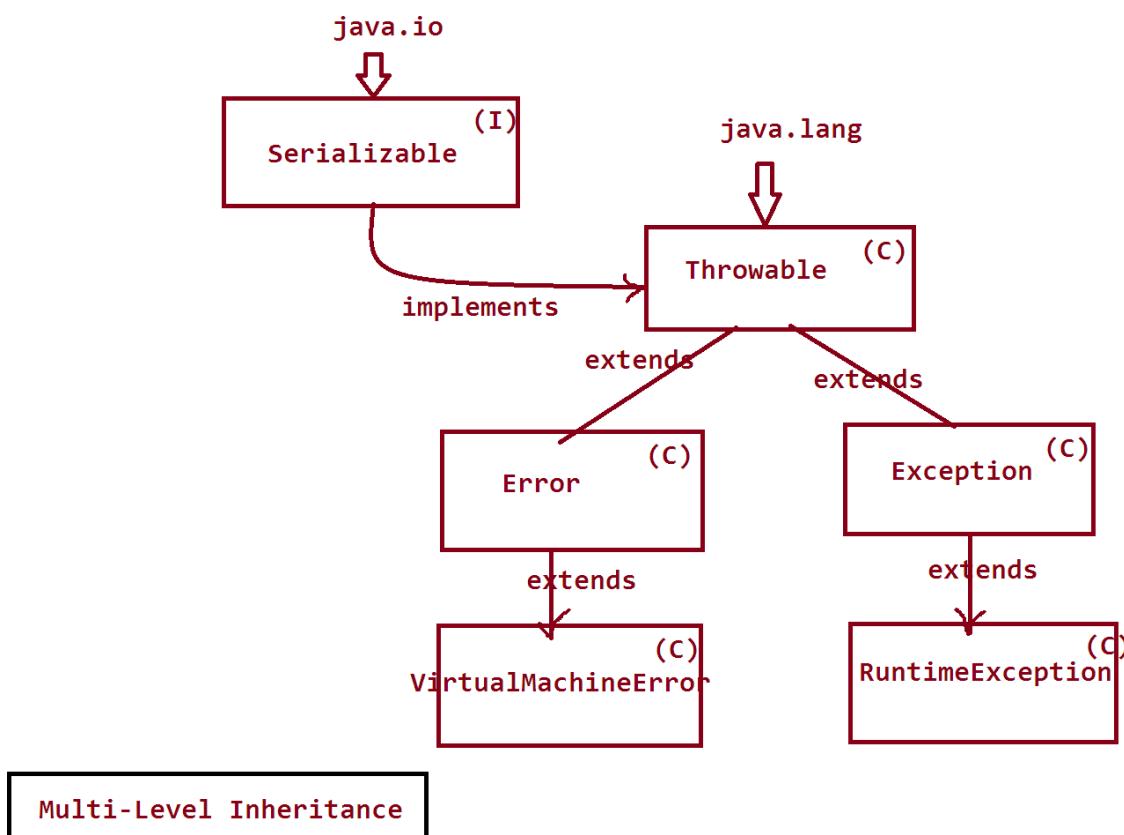
=>"Throwable" is a class from 'java.lang' package and which is root of Exception Handling Process.

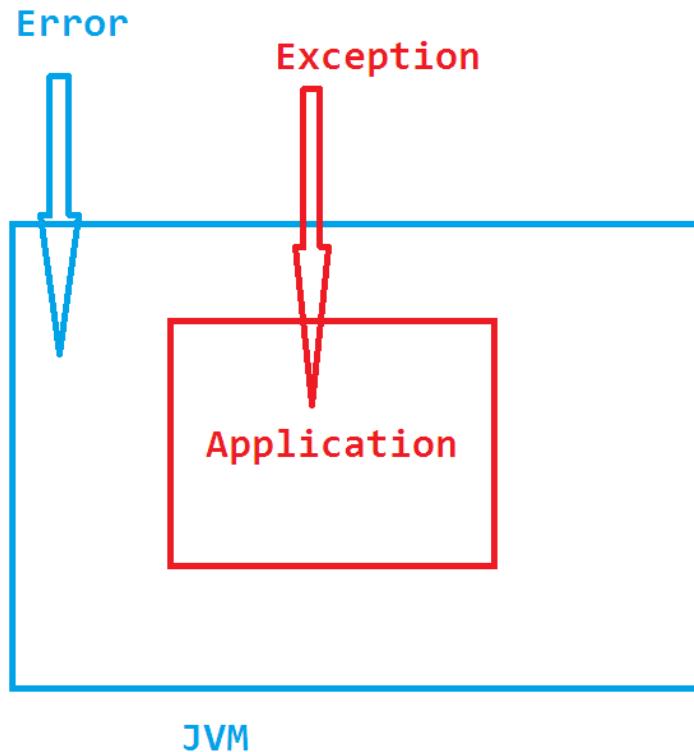
=>This "Throwable" class is extended into the following SubClasses:

1.Error class

2.Exception Class

Hierarchy of "Throwable":





### **1.Error class:**

=>The disturbance which is occurred from the environment(JVM) is known as 'error'.

=>'`java.lang.Error`' class is the PClass of all the errors raised from the Environment(JVM).

**Note:**

=>There is no concept of "Error handling process".

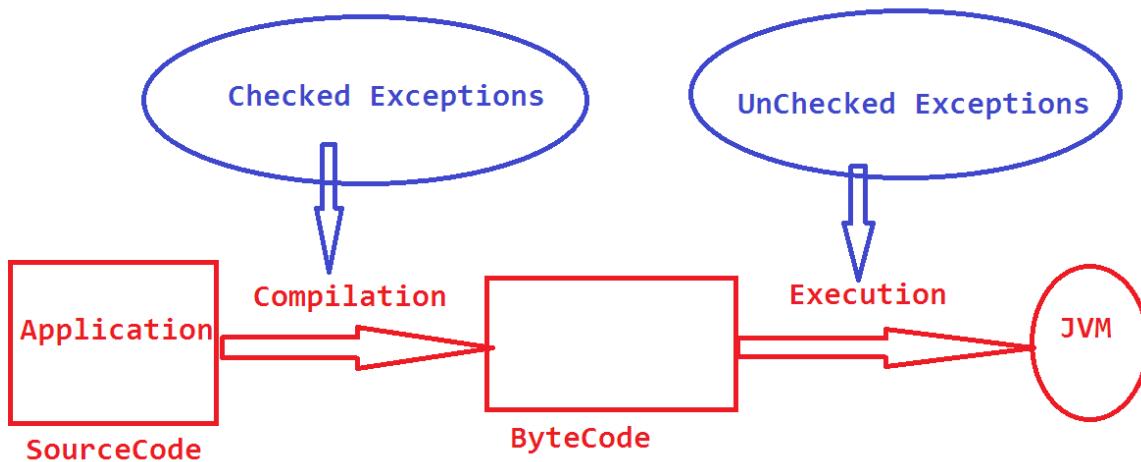
### **2.Exception Class:**

=>'`java.lang.Exception`' class is the PClass of all the exceptions raised from the application.

=>The exceptions which are raised from the application are categorized into two types:

(a)UnChecked Exceptions

(b)Checked Exceptions



(a)UnChecked Exceptions:

=>The exceptions which are not identified by the compiler at compilation stage,will be raised at execution stage are known as UnChecked Exceptions or Runtime exceptions.

=>These Unchecked exceptions are categorized into two types:

(i)Built-In Unchecked exceptions

(ii)User defined UnChecked exceptions

(i)Built-In Unchecked exceptions:

=>The UnChecked exceptions which are available from JavaLib are known as Built-In UnChecked exceptions.

**Exp:**

**java.lang.ArithmaticException**  
**java.lang.NumberFormatException**  
**java.util.InputMismatchException**  
...

**Dt:28/1/2021**

**Exp program:**

```
package maccess;

import java.util.Scanner;

import java.util.InputMismatchException;

public class DException1 {

    public static void main(String[] args) {

Scanner s = new Scanner(System.in);

try{

System.out.println("Enter the value:");

int v = s.nextInt(); //Exception for other than integer value

int k = 44/v; //Exception when v=0

System.out.println("The value k:"+k);

} //end of try

catch(InputMismatchException ime){

    System.out.println("Enter only Integer value...");

}

catch(ArithmaticException ae){

    System.out.println("Enter only NonZero value...");

}
```

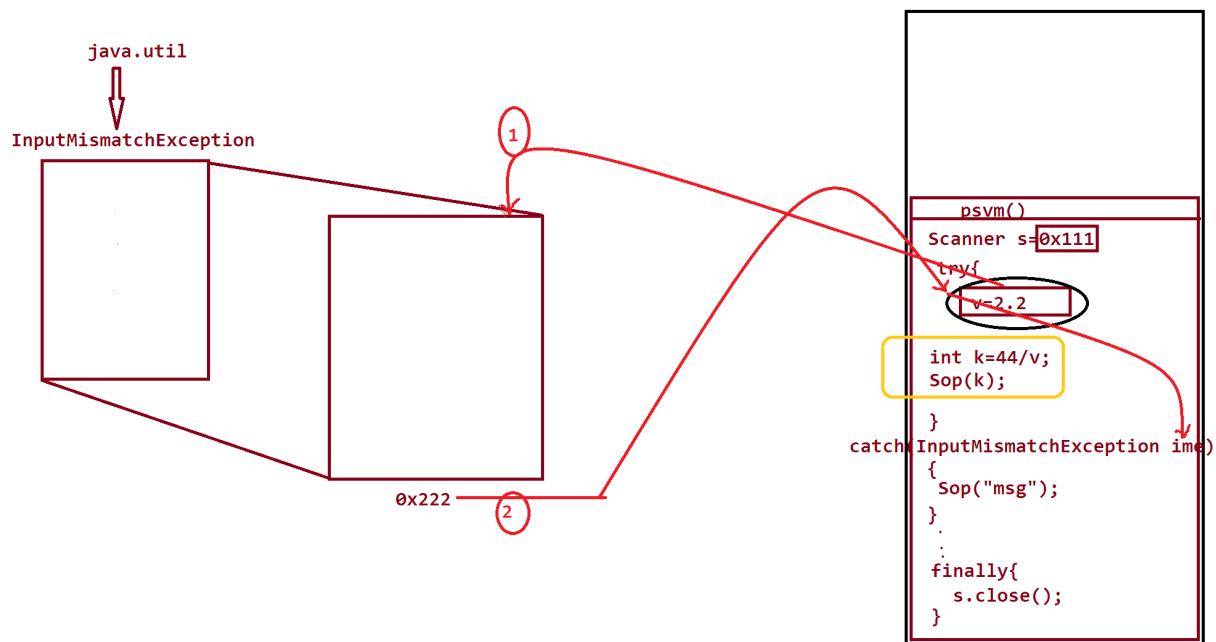
```

finally{
    s.close();
}

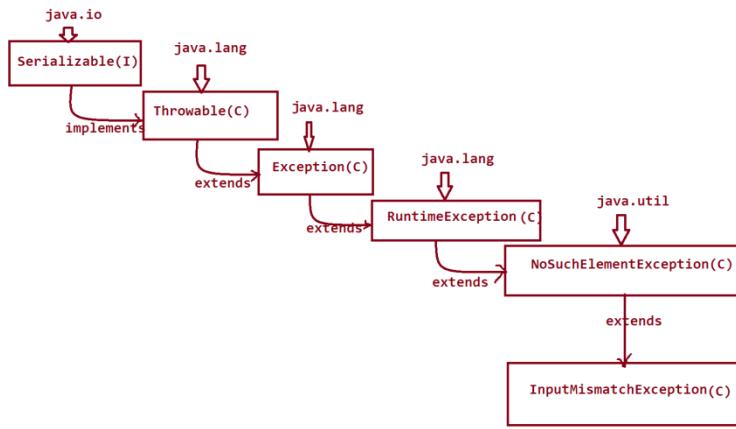
}

```

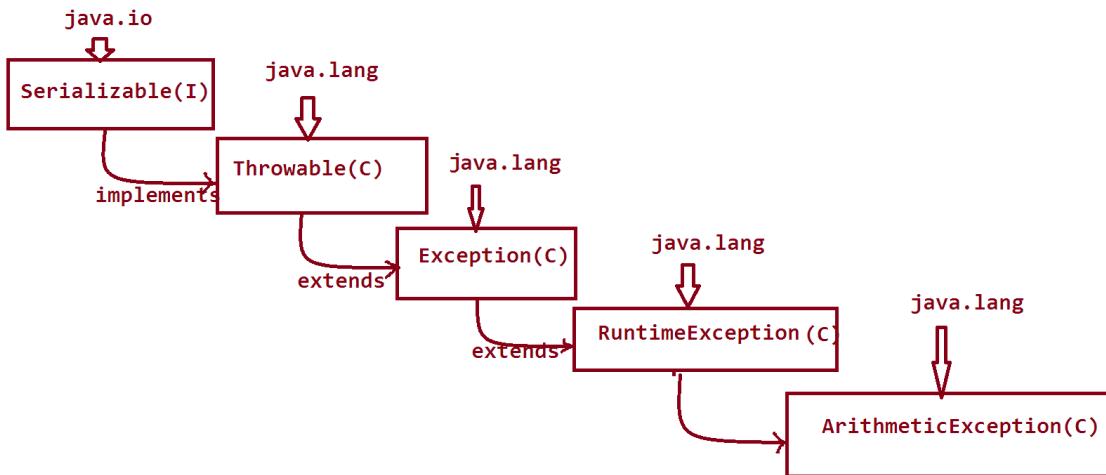
**Execution flow of above program:**



**Hierarchy of `java.util.InputMismatchException`:**



### Hierarchy of `java.lang.ArithmeticException`:



\*imp

### (ii) User defined UnChecked exceptions:

=>The Unchecked exceptions which are defined and raised by the programmer are known as User defined UnChecked exceptions.

**Steps to define and raise User defined Exception:**

**step1 : The class must be extended from java.lang.Exception**

**step2 : Define the condition to raise exception**

**step3 : when the condition is true,then create object for the class**

**where the condition is declared.**

**Step4 : Throw the reference onto catch block using 'throw' keyword.**

**step5 : Use catch block to hold reference and display the msg**

**from the catch block.**

**Exp program:**

**wap to read bSal and calculate totSal?**

**Note:**

**=>Min bSal is 5000/-**

```
package maccess;

import java.util.Scanner;

import java.util.InputMismatchException;

@SuppressWarnings("serial")

public class DException2 extends Exception

{

    public static void main(String[] args) {

Scanner s = new Scanner(System.in);

try{



System.out.println("Enter the bSal:");

int bSal = s.nextInt();

if(bSal<5000)//to raise Exception

{
```

```
DException2 ob = new DException2();

throw ob;

}

float totSal = bSal+(0.93F*bSal)+(0.63F*bSal);

System.out.println("TotSal:"+totSal);

}//end of try

catch(InputMismatchException ime){

System.out.println("Enter bSal in integer value...");

}

catch(DException2 ob){

System.out.println("Invalid bSal...(min 5000/-)");

}

finally{

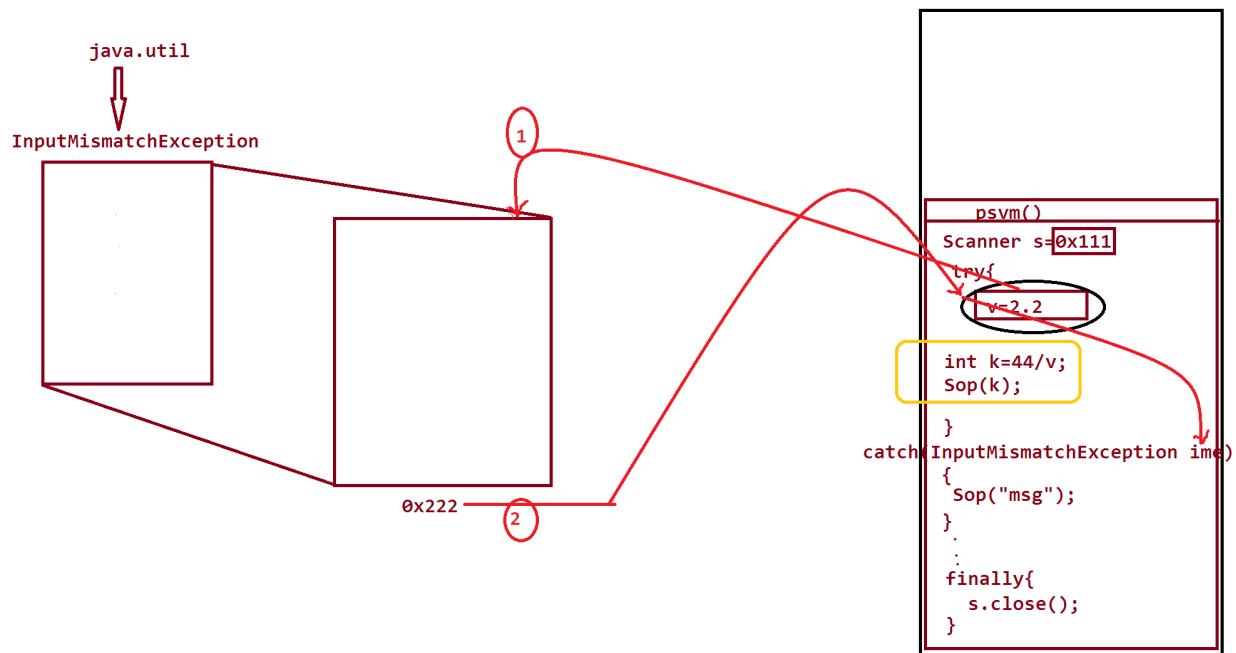
s.close();

}

}
```

Dt : 29/1/2021

### Execution flow of above program:



### Passing Exception message while object creation:

#### Exp program:(DEception3.java)

```
package maccess;
import java.util.Scanner;
import java.util.InputMismatchException;
@SuppressWarnings("serial")
public class DEception3 extends Exception
{
    public DEception3(String msg){
```

```
super(msg);

}

public static void main(String[] args) {

Scanner s = new Scanner(System.in);

try{

System.out.println("Enter the bSal:");

int bSal = s.nextInt();

if(bSal<5000)//to raise Exception

{

DException3 ob = new DException3("Invalid bSal...(min 5000/-)");

throw ob;

}

float totSal = bSal+(0.93F*bSal)+(0.63F*bSal);

System.out.println("TotSal:"+totSal);

}//end of try

catch(InputMismatchException ime){

System.out.println("Enter bSal in integer value...");

}

catch(DException3 ob){

//System.out.println(ob.getMessage());

ob.printStackTrace();

}

finally{

s.close();

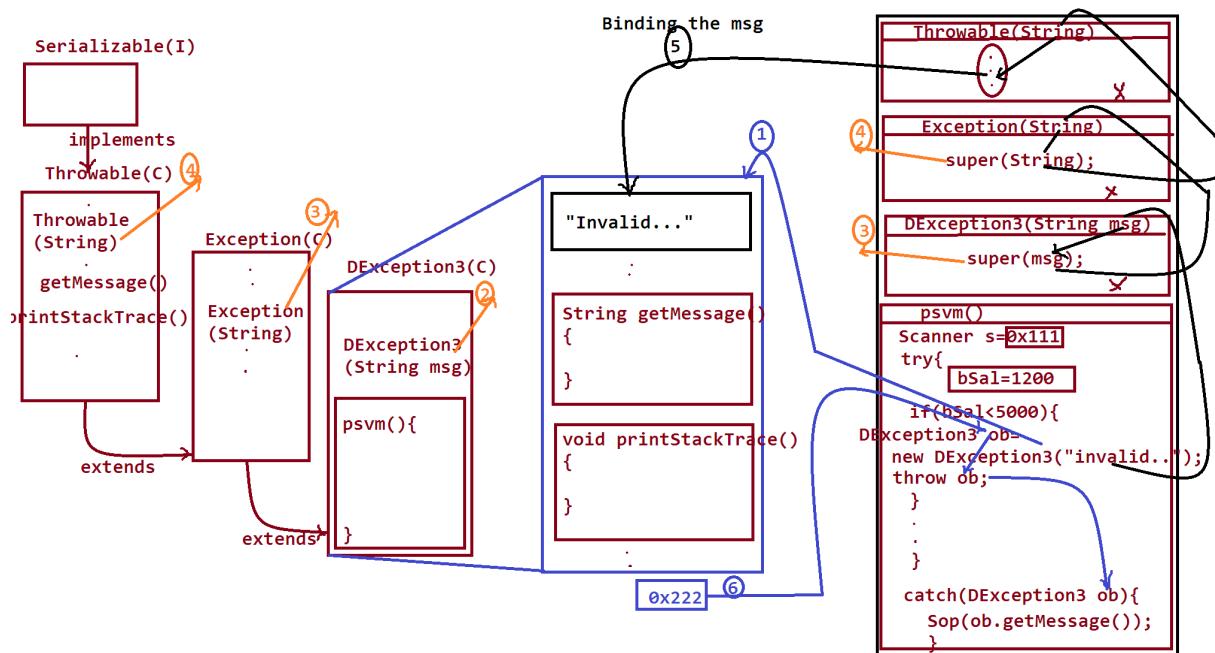
}
```

```

    }
}

```

### Execution flow of above program:



**define getMessage() method?**

=>`getMessage()` method is from "Throwable" class and which is used to display the msg from the object.

**Method Signature:**

`public java.lang.String getMessage();`

**syntax:**

`String msg = obj.getMessage();`

**define printStackTrace()?**

=>printStackTrace() method also from "Throwable" class and which display msg and exception details like Class\_name,method\_name and line\_number from where the exception is raised.

**Method Signature:**

**public void printStackTrace();**

**syntax:**

**obj.printStackTrace();**

---

**Assignment:**

**wap to validate branch and rollNo?**

**branch must be**

**CSE EEE ECE**

**rollNo must be**

**10 digits(Alphanumeric)**

---

**\*imp**

**(b)Checked Exceptions:**

=>The exceptions which are identified by the compiler at compilation stage are known Checked Exceptions or Compile time exceptions.

=>These Checked exceptions are categorized into two types:

**(i)Built-In checked exceptions**

**(ii)User defined Checked exceptions**

**(i)Built-In checked exceptions:**

=>The Checked exceptions which are available from JavaLib are known as Built-In Checked Exceptions.

**Exp:**

**java.lang.InterruptedException**

**java.lang.ClassNotFoundException**

...

Dt : 1/1/2021

faq:

**define sleep() method?**

=>sleep() method is used to stop the execution process temporarily on some time.

=>This method is available from 'java.lang.Thread' class.

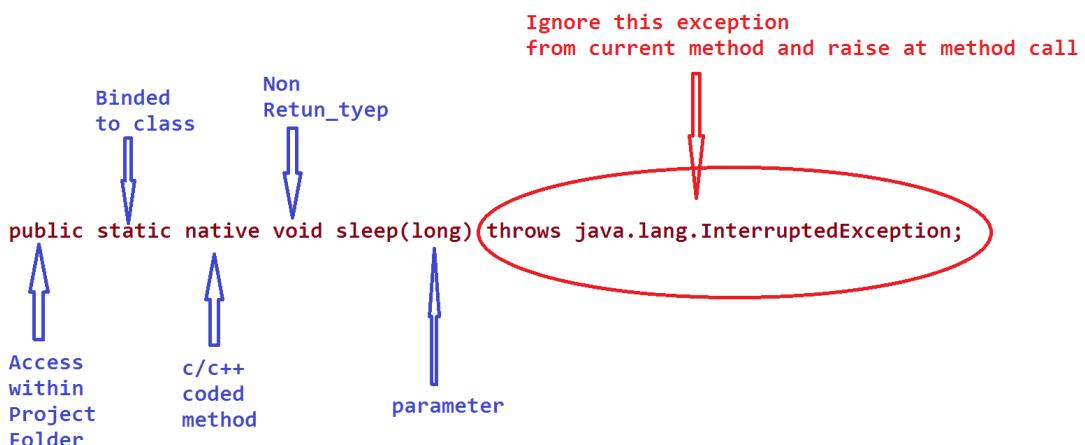
**Method Signature of sleep():**

```
public static native void sleep(long)
```

```
throws java.lang.InterruptedException;
```

**syntax:**

```
Thread.sleep(2000);
```



**Note:**

=>'throws' keyword specify to ignore the exception from current method and raise at method\_call.

=>when the exception is raised at method call then it is identified by the compiler at compilation stage and comes under checked exception or Compile time exception.

Exp program:

```
package maccess;
public class DException4 {
    public static void main(String[] args) {
        for(int i=1;i<=10;i++){
            try{
System.out.println("Val : "+i);
Thread.sleep(5000);
            }catch(InterruptedException ie){ie.printStackTrace();
            } //end of loop
        }
    }
}
```

---

(ii)User defined Checked exceptions:

=>The checked exceptions which are defined and raised by the programmer are known as User defined Checked exceptions.

Note:

=>To raise User defined checked exceptions we use 'throw' keyword part of catch block and perform re-throwing process,in this process we use 'throws' keyword added with method signature to ignore the exception in current method and raise at method call.

Exp program:

wap to read stuBranch and rollNo, and display the same?

Note:

=>If the branch is in CSE,ECE and EEE then read the roll of 10 digits.

BranchCheck.java

```
package test;
@SuppressWarnings("serial")
public class BranchCheck extends Exception{
    public BranchCheck(){}
    public BranchCheck(String msg){
        super(msg);
    }
    public void verify(String br) throws BranchCheck
    {
        try{
            switch(br){
                case "CSE":
                    break;
                case "EEE":
                    break;
                case "ECE":
                    break;
                default://Exception
                    BranchCheck bc = new BranchCheck("Invalid Branch..");
                    throw bc;
            }//end of switch
        }catch(BranchCheck bc){
            throw bc;//re-throwing
        }
    }
}
```

RollNoCheck.java

```
package test;
@SuppressWarnings("serial")
public class RollNoCheck extends Exception{
    public RollNoCheck(){}
    public RollNoCheck(String msg){
        super(msg);
    }
    String branch=null;
    public void verify(String br,String code) throws
RollNoCheck
    {
        try{
            switch(code){
                case "05":branch="CSE";
                break;
                case "02":branch="EEE";
                break;
                case "04":branch="ECE";
                break;
            }//end f switch
            if(branch!=null){
                if(!branch.equals(br)){
                    RollNoCheck rnc = new RollNoCheck("RollNo Not matched
with branch..");
                    throw rnc;
                }
            }else{
                RollNoCheck rnc = new RollNoCheck("RollNo Not matched
with branch..");
                throw rnc;
            }
        }catch(RollNoCheck rnc){
            throw rnc;//re-throwing
```

```
    }
}
}
```

### DException5.java

```
package maccess;

import java.util.Scanner;

import test.BranchCheck;

import test.RollNoCheck;

@SuppressWarnings("serial")

public class DException5 extends Exception{

    public DException5(String msg){

        super(msg);

    }

    public static void main(String[] args) {

Scanner s = new Scanner(System.in);

try{

System.out.println("Enter the branch:");

String br = s.nextLine().toUpperCase();

BranchCheck bc = new BranchCheck();

bc.verify(br);

System.out.println("Enter the rollNo:");

String rollNo = s.nextLine();

if(rollNo.length()!=10)//Exception

{

DException5 de = new DException5("Invalid rollNo...");
```

```
throw de;  
}  
  
RollNoCheck rnc = new RollNoCheck();  
  
rnc.verify(br,rollNo.substring(6,8));  
  
System.out.println("Branch:"+br);  
  
System.out.println("RollNo:"+rollNo);  
  
}catch(BranchCheck bc){  
  
System.out.println(bc.getMessage());  
  
}catch(DException5 de){  
  
System.out.println(de.getMessage());  
  
}catch(RollNoCheck rnc){  
  
System.out.println(rnc.getMessage());  
  
}finally{  
  
s.close();  
}  
}  
}
```

Dt : 2/2/2021

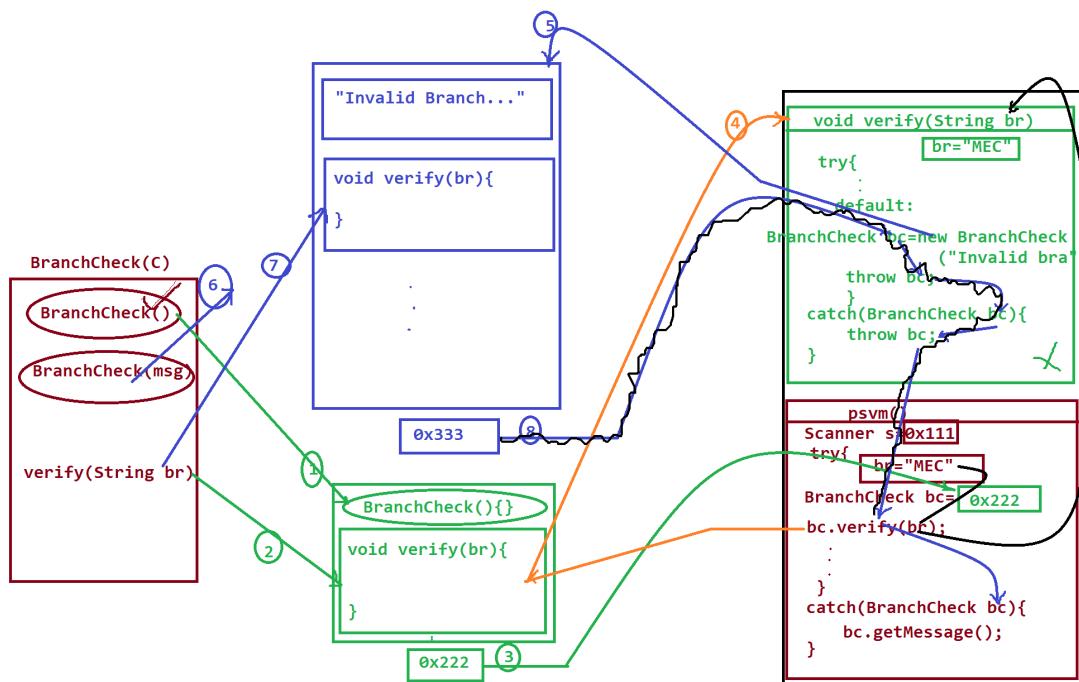
Execution flow of above program:

ClassFiles:

**BranchCheck.class**

**RollNoCheck.class**

**DException5.class(MainClass)**



faq:

define Exception re-throwing?

=>The process of declaring 'throw' keyword part of catch block and throwing the exception is known as Exception re-throwing process.

**Note:**

=>In Exception re-throwing process the exception is moved to the method call.(Raised at method call)

**faq:**

**define Exception propagation?**

=>In re-throwing process the exception is moved from one method to another method is known as Exception Propagation.

---

**define try-with-resource?**

=>'try-with-resource' statement is introduced by Java7 version and in which resource is declared with try.

**syntax:**

```
try(resource1;resource2;...)  
{  
    //set-of-statements;  
}
```

**Exp:**

```
try(Scanner s = new Scanner(System.in);)  
{  
    //set-of-statements;  
}
```

**Note:**

=>when we use 'try-with-resource' statement then finally block is not needed because which performs auto closing process.  
=>'catch' block is not mandatory for 'try-with-resource' statement.

---

**define 'Enhanced try-with-resource'?**

=>'Enhanced try-with-resource' statement introduced by Java9 version and in which resources are declared outside the try and the resource variables are declared with try.

**syntax:**

```
resource1;resource2;...  
try(res-var1;res-var2;...)  
{  
    //set-of-statements;  
}
```

**Exp:**

```
Scanner s = new Scanner(System.in);  
try(s;)  
{  
    //set-of-statements;  
}
```

---

**Note:**

=>From Java7 version onwards we can use single catch block to hold multiple exceptions.

**syntax:**

```
catch(Exception1 | Exception2 | Exception3 | ...)
```

```
{
```

```
//msg
```

```
}
```

---

**faq:**

**define 'java.lang.NullPointerException'?**

=>'java.lang.NullPointerException' is raised when use NonPrimitive data type variable which is assigned with 'null' value.

---

**Modified DException5.java**

```
package maccess;  
import java.util.Scanner;  
import test.BranchCheck;  
import test.RollNoCheck;  
@SuppressWarnings("serial")  
public class DException5 extends Exception{  
    public DException5(String msg){
```

```
super(msg);

}

public static void main(String[] args) {

try(Scanner s = new Scanner(System.in));//Java7

{

try{

System.out.println("Enter the branch:");

String br = s.nextLine().toUpperCase();

BranchCheck bc = new BranchCheck();//0-parameter constructor call

bc.verify(br);

System.out.println("Enter the rollNo:");

String rollNo = s.nextLine();

if(rollNo.length()!=10)//Exception

{

DException5 de = new DException5("Invalid rollNo...");

throw de;

}

RollNoCheck rnc = new RollNoCheck();

rnc.verify(br,rollNo.substring(6,8));

System.out.println("Branch:"+br);

System.out.println("RollNo:"+rollNo);

}

catch(BranchCheck | DException5 | RollNoCheck ob)//Java7

{

System.out.println(ob.getMessage());

}
```

```
    }  
}//End of outerTry  
}  
}
```

Dt : 3/2/2021

faq:

wt is the diff b/w

(i)throw

(ii)throws

=>'throw' is used to throw the reference onto catch block,in the process of handling the exception.

=>'throws' keyword is attached with method signature to specify,ignore the exception from the current method and raise at method\_call.

---

Assignment:

JavaProject : StudentResult

(Convert MainClass8.java program into ExceptionHandlingProcess)

packages

p1 : BranchCheck,RollNoValidate

p2 : SResulr

p3 : Student(MainClass)

---

define Encapsulation process?

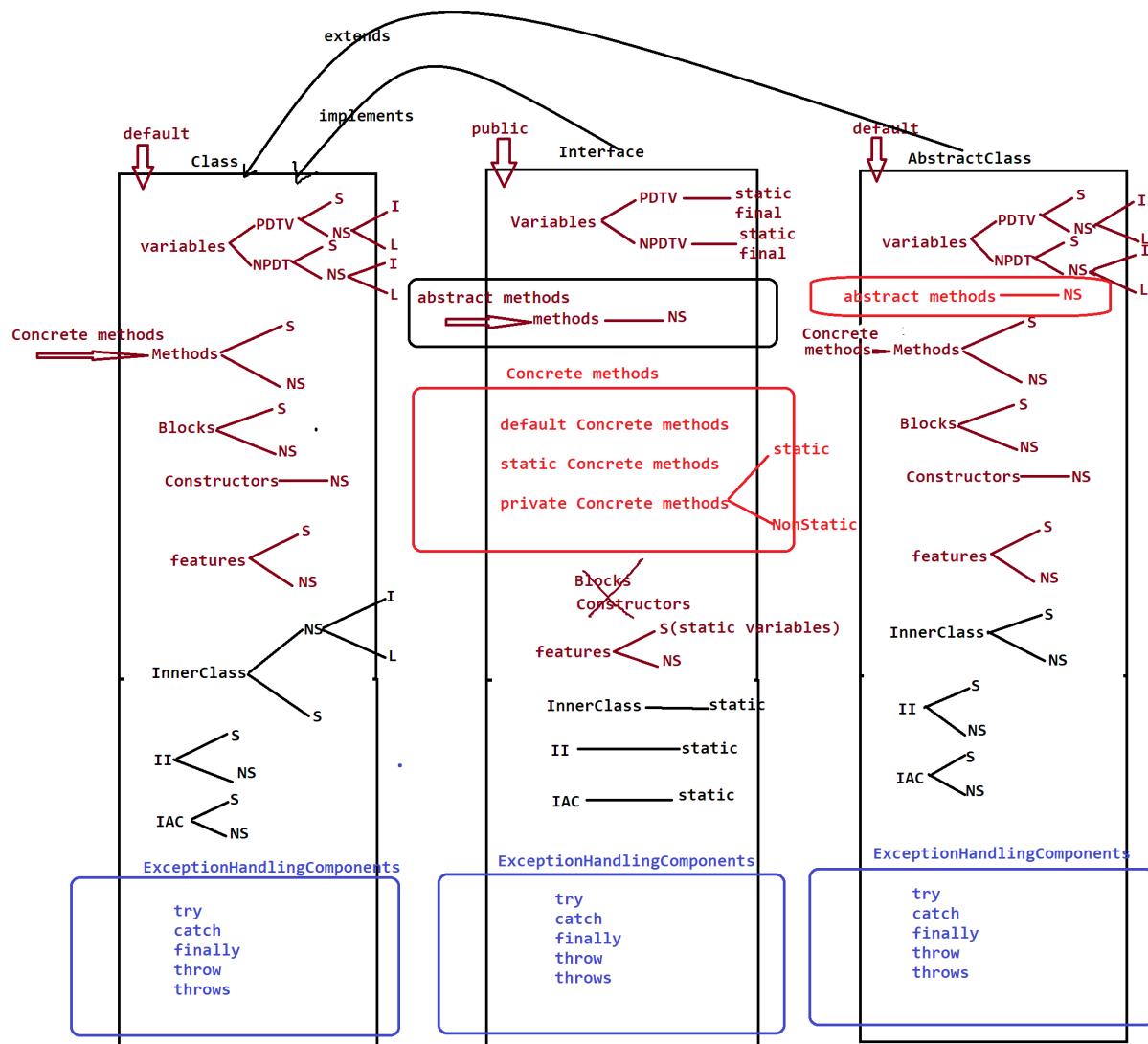
=>The process of binding all the programming components into a single unit 'class' is known as Encapsulation process.

Note:

=>Class in java is a collection of Primitive datatype variables, NonPrimitive datatype variables(references),methods,blocks,constructors ,features,InnerClasses,InnerInterfaces,InnerAbstractClasses and

## ExceptionHandling components(try,catch,finally,throw and throws)

=>Interface in Java is a collection of Primitive datatype variables,  
 NonPrimitive datatype variables(references),abstract methods,  
 Concrete methods,features,InnerClasses,InnerInterfaces,InnerAbstract  
 Classes and ExceptionHandling Components(try,catch,finally,throw,throws)



define Abstraction processes?

=>The process of hiding certain details and showing only essential information, is known as abstraction process.

**Note:**

=>This abstraction process can be achieved using Interfaces and abstract classes.

---

Dt : 4/2/2021

\*imp

**PolyMorphism in Java:**

=>The process in which the same programming component having different forms is known PolyMorphism.

**Poly - Many**

**Morphism - Forms**

=>This PolyMorphism is categorized into two types:

**1.Dynamic Polymorphism**

**2.Static Polymorphism**

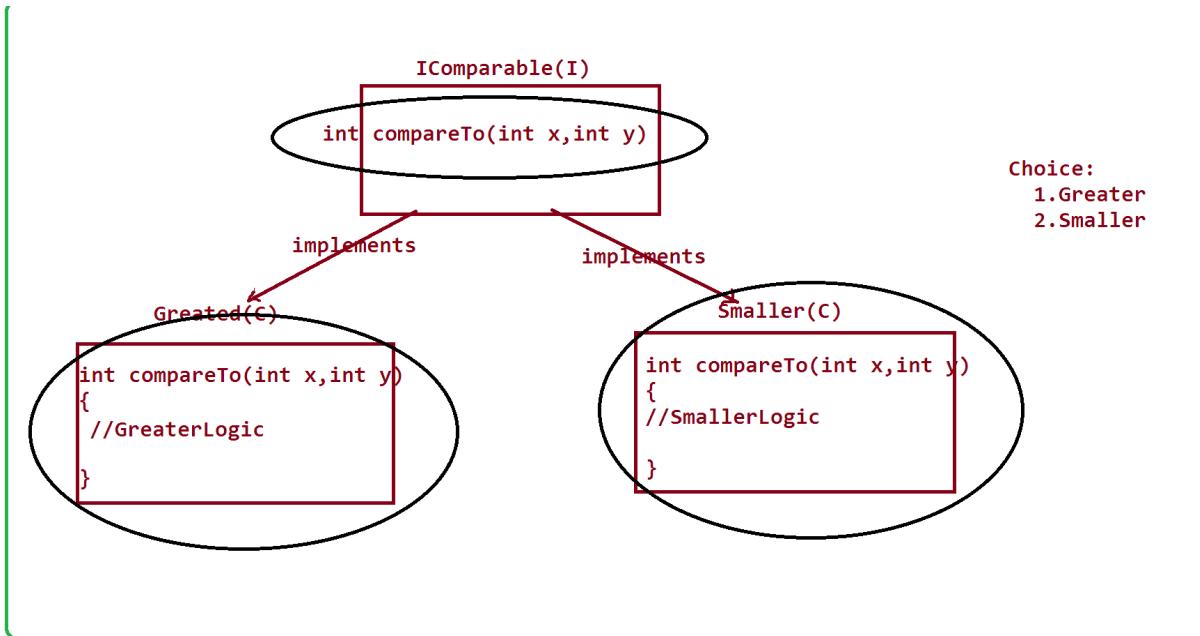
**1.Dynamic Polymorphism:**

=>The Polymorphism which is exhibited at execution stage is known as Dynamic Polymorphism or Runtime Polymorphism.

**Exp:**

## Method Overriding process

Diagram:



Note:

=>In the diagram `compareTo()` method is having the following two forms  
while execution process:

=>`greaterLogic`

=>`smallerLogic`

=>Using Overriding process we can make method available in diff forms  
at runtime,because of this reason Method Overriding comes under  
Dynamic PolyMorphism or Runtime Polymorphism.

---

2.Static Polymorphism:

=>The polymorphism which is exhibited at compilation stage is known as Static Polymorphism or Compile time polymorphism.

Exp:

Method Overloading process.

define Method Overloading process?

=>More than one method with same method\_name but differentiated by their para\_list or para\_type is known as Method Overloading process.

Exp program:

Addition.java

```
package maccess;

public class Addition {
    public void add(int x,int y){
        System.out.println("Sum1:"+ (x+y));
    }
    public void add(int x,int y,int z){
        System.out.println("Sum2:"+ (x+y+z));
    }
    public void add(int x,float y){
        System.out.println("Sum3:"+ (x+y));
    }
}
```

```
package maccess;
public class DPoly1 {
    public static void main(String[] args) {
        Addition ad = new Addition();
```

```
ad.add(1, 2);
ad.add(1, 2, 3);
ad.add(1, 2.3F);
}
}
```

**Note:**

=>In the above program add() method is having the following forms:

**add(int,int)**

**add(int,int,int)**

**add(int,float)**

=>These methods are identified by the compiler at compilation stage because of this reason Method Overloading comes under Static PolyMorphism or Compile time polymorphism.

---

Dt : 5/2/2021

=>The compiler at compilation Stage will control the following

**Keywords:**

**1.static**

**2.private**

**3.final**

**1.static:**

=>The following are the static programming components:

**(a)static variables**

- (b)static methods
- (c)static blocks
- (d)static classes
- (e)static Interfaces
- (f)static abstract classes

**Note:**

=>There is no concept of static Constructor in Java.

**Behaviour of static programming components:**

=>These static programming components will get the memory within the class while class loading and access with class\_name.

**faq:**

**why static programming components will get the memory within the class?**

=>The compiler at compilation stage will identify the static programming components and binded to the class known as'static binding' or 'compile time binding',because of this reason these static programming components will get the memory within the class while class loaded.

**faq:**

**define Dynamic binding?**

=>The process of binding the NonStatic programming components to

**the object while object creation is known as Dynamin binding or runtime binding.**

---

## **2.private:**

**=>The following are the private programming components:**

- (a)private variables**
- (b)private methods**
- (c)private Constructors**
- (d)private classes**

**Note:**

**=>There is no concept of Private blocks,Private Interfaces and Private abstract classes**

### **(a)private variables:**

**=>The variables which are declared with private keyword are known as private variables.**

**Behaviour:**

**=>Private variables are accessed by the methods of same class.**

**Note:**

**=>In realtime private variables are used part of Bean classes and POJO classes.**

**(POJO - Plain Old Java Object)**

**(b)private methods:**

=>The methods which are declared with private keyword are known as Private methods.

**Behaviour:**

=>Private methods are accessed by the NonPrivate methods of same class.

**(c)private Constructors:**

=>The constructors which are declared with private keyword are known as private constructors.

**Behaviour:**

=>Private Constructors are executed when the object is created inside the class where the Constructors are declared.

**Note:**

=>In realtime Private constructors are used to construct 'Singleton class design pattern'.

**(d)private classes:**

=>The classes which are declared with private keyword are known as private classes

**Note:**

**=>These private classes must be declared as only InnerClasses.**

---

Dt : 6/2/2021

Example program:

(Wap to demonstrate private variables and private methods)

```
package maccess;
public class Display {
    private int a=10;
    private void dis(){
        System.out.println("====Private method dis()====");
        System.out.println("The value a:"+a);
    }
    public void m()//NonPrivate method
    {
        this.dis(); //Private method call
    }
}
```

```
package maccess;
public class DPoly2 {
    public static void main(String[] args) {
        Display d = new Display();
        //d.dis(); //Compilation
        d.m();
    }
}
```

\*imp

define 'SingleTon class design pattern'?

=>In 'SingleTon class design pattern' only one object created inside the class and the reference of the object is accessed outside the class.

=>we use the following three components to construct 'SingleTone class design pattern':

- (i)private static reference variable
- (ii)private constructor
- (iii)static method

(i)private static reference variable:

=>private static reference variable will hold the reference of object created inside the class.

(ii)private constructor:

=>Private Constructor will restrict the object creation only inside the class.

(iii)static method:

=>we use static method to access the reference of object outside the class.

Exp program:

```
package maccess;
public class Test1 {
    private static Test1 t=null;
    private Test1(){}
    static{
```

```

        t = new Test1();
    }
    public static Test1 getRef(){
        return t;
    }
    public void dis(){
        System.out.println("==dis()==");
    }
}

```

```

package maccess;
public class DPoly3 {
    public static void main(String[] args) {
//Test1 ob = new Test1();
Test1 ob = Test1.getRef(); //Factory method
ob.dis();
    }
}

```

=>Based on the Object creation the 'SingleTon Class design pattern'

**is categorized into two types:**

**(a)Early Instantiation process**

**(b)Late Instantiation process**

**(a)Early Instantiation process:**

=>In Early Instantiation process, the object is created while class loading.

**Exp:**

**above program**

**Note:**

=>In Early Instantiation process the object is created using 'static' block.

**(b)Late Instantiation process:**

=>In Late Instantiation process the object is created after class loading.

**Note:**

=>In Late Instantiation process the object is created using Method.

**Exp program:**

```
package maccess;
public class Test2 {
    private static Test2 t=null;
    private Test2(){}
    public static Test2 getRef(){
        if(t==null){
            t = new Test2();
        }
        return t;
    }
    public void dis(){
        System.out.println("==dis()==");
    }
}
```

```
package maccess;
public class DPoly4 {
```

```
public static void main(String[] args) {  
//Test1 ob = new Test1();  
Test1 ob = Test1.getRef(); //Factory method  
ob.dis();  
}  
}
```

---

**Note:**

=>In realtime 'SingleTon class design pattern' is used part of DAO  
(Data Access Object)layer in MVC(Model View Controller)

---

**3.final:**

=>The following are the final programming components:

- (a)final variables
- (b)final methods
- (c)final classes

**Note:**

=>There is no concept of final blocks,final constructors,final  
Interfaces and final abstract classes.

**(a)final variables:**

=>The variables which are declared with final keyword part of classes  
are known as final variables.

**Behaviour:**

=>These final variables must be initialized with values and once  
initialized cannot be modified.

=>The final variables within the class can be initialized using Constructor.

(b)final methods:

=>The methods which are declared with final keyword are known as final methods.

Behaviour:

=>Final methods cannot be Overrided or cannot be replaced.

(c)final classes:

=>The classes which are declared with final keyword are known as final classes.

Behaviour:

=>final classes cannot be Inherited or cannot be extented.

(There is no inheritance for final classes)

---

Note:

=>In realtime,these final programming components are used to construct "Immutable classes".

---

Dt : 8/2/2021

define Immutable class?

=>The classes which are declared with the following rules are known

**as Immutable classes:**

**Rule-1 : The classes must be 'final' classes.**

**Rule-2 : The classes must be declared with private and final variables.**

**Rule-3 : The classes must be declared with 'Getter' methods.**

**Rule-4 : These 'Getter' methods must be final methods**

**define Getter methods?**

**=>The methods which are used to get the data from the objects are known as Getter methods.**

**define Setter methods?**

**=>The methods which are used to set the data to the objects are known as Setter methods.**

**Note:**

**=>These Immutable classes will generate "Immutable objects".**

**define Immutable objects?**

**=>The objects once generated cannot be modified are known as Immutable Objects.(Secured objects)**

**Exp program:**

```
package maccess;  
public final class User {
```

```

private final String userName,password;  

public User(String userName,String password){  

    this.userName=userName;  

    this.password=password;  

}  

public final String getUserName() {  

    return userName;  

}  

public final String getPassword() {  

    return password;  

}  

}

```

```

package maccess;  

public class DPoly5 {  

    public static void main(String[] args) {  

User u = new User("nit.v","mzu672");//Immutable  

object  

System.out.println("UserName:"+u.getUserName());  

System.out.println("Password:"+u.getPassword());  

    }  

}

```

**Note:**

=>In realtime Immutable objects are used to hold transactional details in Banking domain.

---

**Note:**

=>Based on Security, the objects in Java are categorized into two types:

**1.Immutable Objects**

**2.Mutable objects**

**1.Immutable Objects :**

=>The objects once created cannot be modified are known as Immutable objects or Secured Objects.

**2.Mutable objects:**

=>The objects once created can be modified are known as Mutable Objects.

---

**faq:**

**define factory methods?**

=>The methods which hide the object creation logic from the EndUsers are known as factory methods.

**Exp program:**

```
package maccess;
public interface IComparable {
    public abstract int compareTo(int x,int y);
}
```

```
package maccess;
public class Result {
    public static IComparable getResult(int choice){
        if(choice==1){
            return (int x,int y)->
            {
                if(x>y) return x;
                else return y;
            };
        }else{
            return (int x,int y)->
            {
                if(x<y) return x;
                else return y;
            };
        }
    }
}
```

```
package maccess;
import java.util.Scanner;
public class DPoly6 {
    public static void main(String[] args) {
Scanner s = new Scanner(System.in);
System.out.println("Enter the value1:");
int v1 = s.nextInt();
System.out.println("Enter the value2:");
int v2 = s.nextInt();
System.out.println("==Choice==");
System.out.println("1.Greater\n2.Smaller");
System.out.println("Enter the Choice:");
int choice = s.nextInt();
```

```
if(choice ==1 || choice==2){  
System.out.println("Result:"+Result.getResult(choice)  
.compareTo(v1,v2));  
        //getResult() is a factory method  
    }else{  
System.out.println("Invalid choice...");  
    }  
s.close();  
    }  
}
```

**faq:**

**define factory method pattern?**

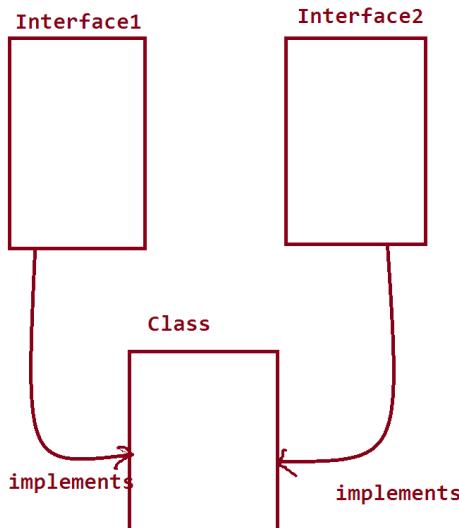
=>The process in which we hide the implementation and return the reference of object onto Interface reference variable is known as **Factory method pattern.**

---

Dt : 9/2/2021

### Multiple Inheritance Models:

**Model-1 : Extracting the features from more than one Interface into a Class.**



### Example program:

```
package maccess;
public interface ITest1 {
    public abstract void m();
    public abstract void m1();
    static void m3(){
        System.out.println("==ITest2 m3()==");
    }
}
```

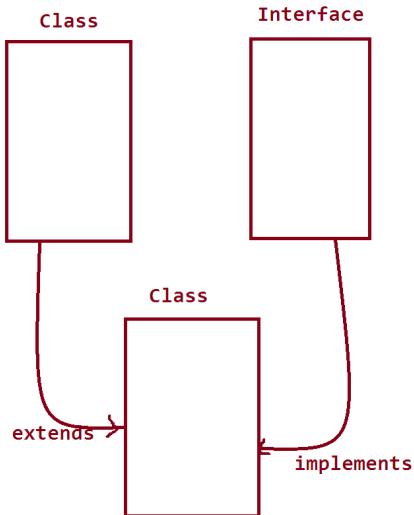
```
package maccess;
public interface ITest2 {
    public abstract void m();
    public abstract void m2();
```

```
static void m3(){
System.out.println("====ITest2 m3()====");
}
}
```

```
package maccess;
public class IClass implements ITest1,ITest2{
    public void m1(){
System.out.println("====m1()====");
    }
    public void m2(){
System.out.println("====m2()====");
    }
    public void m(){
System.out.println("====m()====");
    }
}
```

```
package maccess;
public class MianClass1 {
    public static void main(String[] args) {
IClass ob = new IClass();
ob.m1();
ob.m2();
ob.m();
ITest1.m3();
ITest2.m3();
    }
}
```

## Model-2 : Extracting the features from one class and any number of Interfaces into a class



### Example program:

```
package maccess;
public interface ITest1 {
    public abstract void m2();
}
```

```
package maccess;
public class PClass {
    public void m1(){
        System.out.println("==>PClass m1()==");
    }
}
```

```
package maccess;
public class CClass extends PClass implements ITest1{
```

```

public void m2(){
System.out.println("====CClass m2()====");
}
}

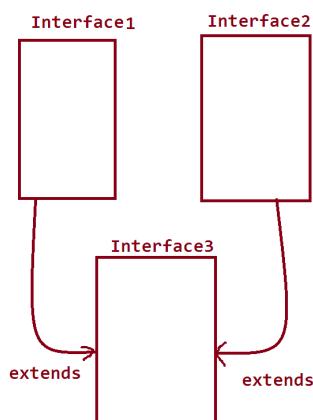
```

```

package maccess;
public class MainClass2 {
    public static void main(String[] args) {
CClass ob = new CClass();
ob.m1();
ob.m2();
    }
}

```

**Model-3 : Extracting the features from More than one Interface into a Interface.**



```

package maccess;
public interface ITest1 {
    public abstract void m1();
}

```

```
package maccess;
public interface ITest2 {
    public abstract void m2();
}
```

```
package maccess;
public interface ITest3 extends ITest1,ITest2{
    public abstract void m3();
}
```

```
package maccess;
public class MainClass3 {
    public static void main(String[] args) {
ITest3 ob = new ITest3(){
    public void m1(){
System.out.println("====m1()====");
    }
    public void m2(){
System.out.println("====m2()====");
    }
    public void m3(){
System.out.println("====m3()====");
    }
};
ob.m1();
ob.m2();
ob.m3();
    }
}
```

=====

**Note:**

=>In Object oriented programming,everything must be in the form of Objects.In this process we must make Primitive datatypes available in the form of Objects.

=>To Make Primitive DataTypes available in the form of objects we use the following:

(i)TypeCasting in Java

(ii)WrapperClasses

**(i)TypeCasting in Java:**

=>The process of converting one datatype value into another datatype value is known as TypeCasting process.

**Note:**

=>TypeCasting process is not applicable to convert Primitive DataTypes into NonPrimitive datatypes and ViceVersa.

**TypeCasting-case1 :**

=>TypeCasting process is performed on Primitive datatypes in the following two ways:

(a)Widening process

(b)Narrowing process

**(a)Widening process:**

=>The process of Converting Lower datatype values into Higher datatype values is known as Widening process,which is also known as Implicit TypeCasting process or UpCasting process.

**(b)Narrowing process:**

=>The process of Converting Higher datatype values into Lower datatype values is known as Narrowing process,which is also known as Explicit TypeCasting process or DownCasting process.

Exp program:

```
package maccess;
public class TypeCast1 {
    public static void main(String[] args) {
char ch1='A';
int k = (char)ch1;//Widening process
int x = 87;
char ch2 = (char)x;//Narrowing process
System.out.println("ASCII value of A:"+k);
System.out.println("Char at value 87:"+ch2);
    }
}
```

Dt : 10/2/2021

**TypeCasting-case2:**

=>TypeCasting process is not applicable on NonPrimitive datatypes,  
but the following two processes are performed in Inheritance process:

- (a)Generalization process
- (b)Specialization process

**(a)Generalization process:**

=>The process in which one reference is created and the reference  
is binded with all the members of PClass and Overriding members from  
the CClass is known as Generalization process.

=>This Generalization process can also be performed on Interfaces.

**syntax:**

PClass ob = (PClass)new CClass();

Interface ob = (Interface)new ImplClass();

**Note:**

=>This Generalization process is also known as Widening process or  
Implicit TypeCasting process or UpCasting process.

**(b)Specialization process:**

=>The process of constructing CClass by taking one feature from the  
PClass is known as Specialization process.

=>We use the following syntax to achieve Specialization process:

CClass ob = (CClass)new PClass();

**Note:**

=>This Specialization process is also known as Narrowing process or Explicit TypeCasting process or Downcasting process.

=>This Specialization process is not applicable on Interfaces.

=>In Specialization process the PClass must be Built-in class,if not generates 'java.lang.ClassCastException'.

---

\*imp

'java.lang.Object' class:

=>'java.lang.Object' class is the PClass or SuperClass of all the classes declared in the application.

=>The following are the methods from 'java.lang.Object' class:

1.clone()

2.hashCode()

3.toString()

4.equals()

5.wait()

6.notify()

7.notifyAll()

8.getClass()

9.finalize()

1.clone():

=>The process of creating the duplicate copy of an object is known as Cloning process.

=>This cloning process can be performed using 'clone()' method.

Method Signature:

```
protected native java.lang.Object clone() throws  
    java.lang.CloneNotSupportedException;
```

**syntax:**

```
Object o = obj.clone();
```

**Note:**

=>when this method is used,then'java.lang.CloneNotSupportedException' is raised at Method\_call.

Dt : 11/2/2021

**steps to perform cloning process:**

**step1 :** The user defined class must be implemented from 'java.lang.Cloneable' interface.

**step2 :** The user defined class must be constructed with User defined Object return\_type method

**step3 :** This User defined Object return\_type method will call clone() method to perform cloning process.

**step4 :** Execute User defined Object return\_type method to start cloning process.

**Exp program:**

```
package maccess;  
public class Product extends Object implements  
Cloneable//step1  
{  
    public String pCode,pName;  
    public Product(String pCode,String pName){  
        this.pCode=pCode;  
        this.pName=pName;  
    }
```

```

public String toString(){
    return "PCode:"+pCode+"\nPName:"+pName;
}
public Object cloning()//step2
{
    Object o=null;
    try{
        o = super.clone()//step3
    }catch(CloneNotSupportedException cnse){cnse.printStackTrace();}
    return o;
}
}

```

```

package maccess;
public class DClone {
    public static void main(String[] args) {
        //Original Object
Product p1 = new Product("A121","Mouse");
System.out.println("====Original Object====");
System.out.println(p1);
System.out.println("hashCode of p1:"+p1.hashCode());
        //Cloned Object
Product p2 = (Product)p1.cloning()//Step4
System.out.println("====Cloned Object====");
System.out.println(p2);
System.out.println("hashCode of p2:"+p2.hashCode());
Product p3 = p2;
System.out.println("hashCode of p3:"+p3.hashCode());
    }
}

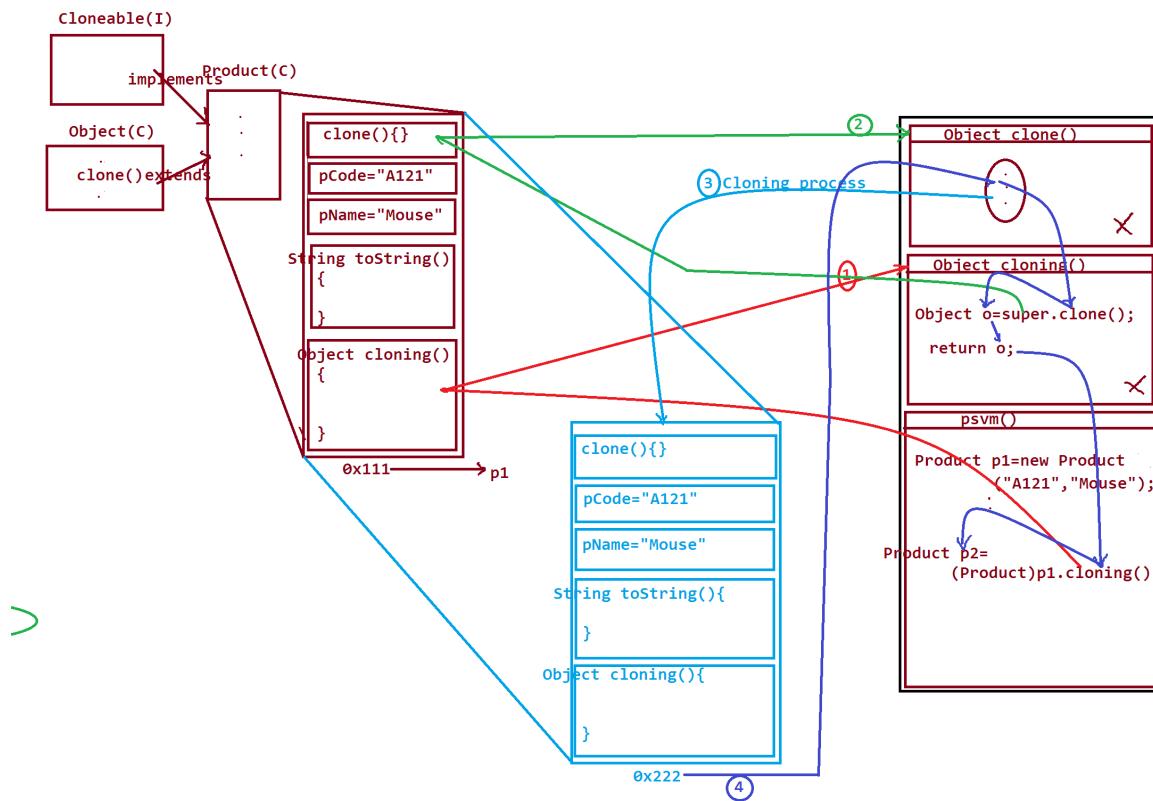
```

**Execution flow of above program:**

**ClassFiles:**

**Product.class**

**DClone.class**



**define Cloneable?**

=>Cloneable is an interface from 'java.lang' package and which specify the cloning process.

=>Cloneable interface is an empty interface because which donot have any members.

=>Which is also known as Marking interface or Tagging interface.

**Note:**

=>Cloning process cannot be performed without implementing from 'java.lang.Cloneable' interface.

**Advantage of Cloning:**

=>Using Cloning process we can take the backup of an objects part of Protection and Security.

**faq:**

**define execution behaviour of Constructor in Cloning process?**

=>Constructor in cloning process is executed for Original objects but not executed for Cloned objects or Duplicate objects.

---

**This Cloning process can be done in two ways:**

- (i)Deep cloning process.
- (ii)Shallow Cloning process.

**(i)Deep cloning process.:**

=>In Deep Cloning process all the objects in the application are cloned,which means OuterClass objects,InnerClass objects and Reffered class objects are cloned.

**(ii)Shallow Cloning process:**

=>In Shallow cloning process only OuterClass objects are cloned,in

which InnerClass objects and RefferedClass objects are not cloned.

---

Dt : 12/2/2021

**2.hashCode():**

=>The unique numeric number which is generated while object creation is known as hashCode.

=>we use 'hashCode()' to display hashCode of an object.

**Method Signature:**

**public native int hashCode();**

**syntax:**

**int code = obj.hashCode();**

**Note:**

=>we display the hashCode and check the Object is created or not.

**3.toString():**

=>**toString()** method is used to display the content from the Object.

**Method Signature:**

**public java.lang.String toString();**

**syntax:**

**String var = obj.toString();**

**Note:**

=>This **toString()** method is executed automatically when we display Object reference variable.

**4.equals():**

=>**equals()** method is used to compare two objects and generates boolean result.

**Method Signature:**

```
public boolean equals(java.lang.Object);
```

**syntax:**

```
boolean k = ob1.equals(ob2);
```

**Exp program:**

```
package maccess;
public class Test1 {
    public String name="Raj";
    public long phNo=9898989898L;
    public String toString(){
        return "Name:"+name+"\nPhNo:"+phNo;
    }
}
```

```
package maccess;
public class MainClass1 {
    public static void main(String[] args) {
Test1 ob1 = new Test1(); //Object1
Test1 ob2 = new Test1(); //Object2
System.out.println("====ob1====");
System.out.println(ob1);
System.out.println("hashCode of
ob1:"+ob1.hashCode());
System.out.println("====ob2====");
System.out.println(ob2);
System.out.println("hashCode of
ob2:"+ob2.hashCode());
System.out.println("==Objects Comparision==");
Test1 ob3 = ob1;
boolean k = ob1.equals(ob3);
```

```

        if(k){
System.out.println("Objects are equal..."); 
        }else{
System.out.println("Objects are NotEqual..."); 
        }
System.out.println("=====classNames====");
System.out.println("ClassName of
ob1:"+ob1.getClass());
System.out.println("ClassName of
ob2:"+ob2.getClass());
System.out.println("ClassName of
ob3:"+ob3.getClass());

        }
}

```

**5.wait()**

**6.notify()**

**7.notifyAll():**

=>These three methods are used to establish communication b/w threads

**in MultiThreading application.(MultiThreading)**

**8.getClass():**

=>This methods is used to display the class\_name of an object.

**Method signature:**

**public final native java.lang.Class<?> getClass();**

**syntax:**

**Class c = obj.getClass();**

**9.finalize():**

=>finalize() method is used to check the object is eligible for Garbage Collection process or not.

(Garbage Collection process)

---

**Note:**

=>Through TypeCasting process in Java,we cannot make Primitive datatypes available in the form of Objects.

=>This can be overcomed using "WrapperClasses".

---

\*imp

**(ii)WrapperClasses:**

=>The classes which are used to make Primitive datatypes available in the form of Objects are known as WrapperClasses.

=>Every primitive datatype will have its own WrapperClass and in this process there are eight WrapperClasses from 'java.lang' package.

**datatype WrapperClass**

**byte Byte**

**short Short**

**int Integer**

**long Long**

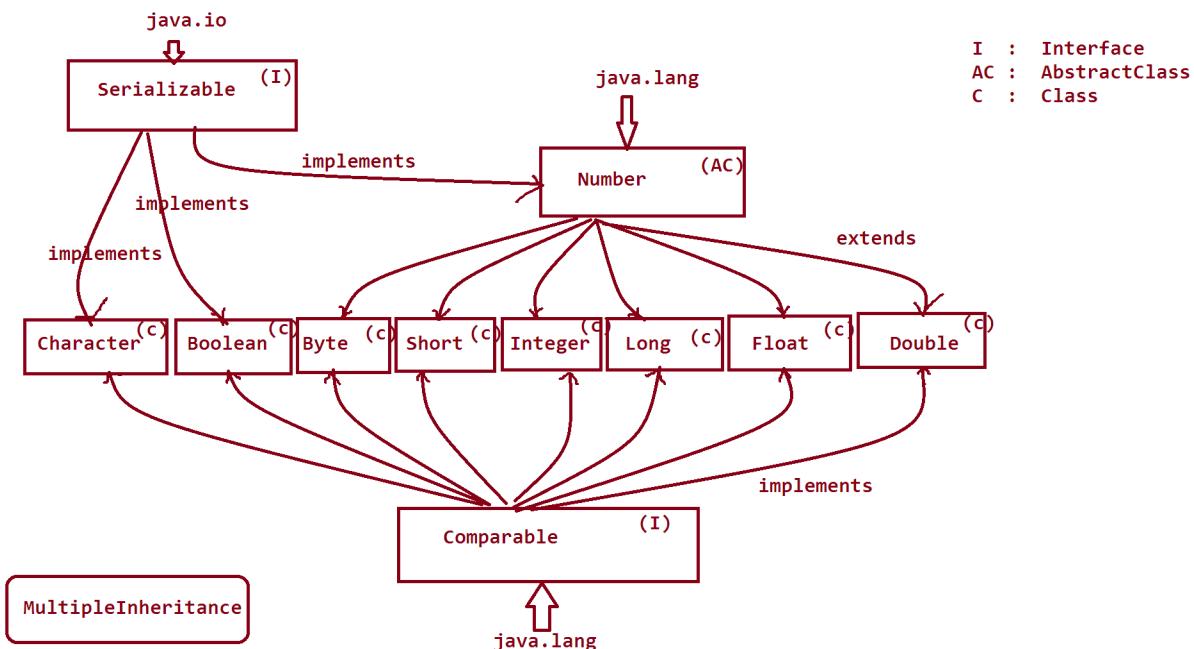
**float Float**

**double Double**

**char Character**

## **boolean Boolean**

### **Hierarchy of WrapperClasses:**



---

**faq:**

**define Boxing process?**

=>The process of binding primitive datatypes into WrapperClass objects

is known as **Boxing Process**.

=>This Boxing process can be performed using 'Constructors' or `valueOf()` method.

**Case-1 : Boxing process using 'Constructors'**

=>The following are the list of constructors available from Wrapper

**Classes:**

**WrapperClass Constructors**

<b>Byte</b>	<b>byte, String</b>
<b>Short</b>	<b>short, String</b>
<b>Integer</b>	<b>int, String</b>
<b>Long</b>	<b>long, String</b>
<b>Float</b>	<b>float, double, String</b>
<b>Double</b>	<b>double, String</b>
<b>Character</b>	<b>char</b>
<b>Boolean</b>	<b>boolean, String</b>

Dt : 13/2/2021

**Integer WrapperClass:**

=> Integer WrapperClass having two constructors.

1 - to bind int value

2 - to bind int value in the form of String

**Constructor Signatures:**

**public java.lang.Integer(int);**

**public java.lang.Integer(java.lang.String) throws**

**java.lang.NumberFormatException;**

**Note:**

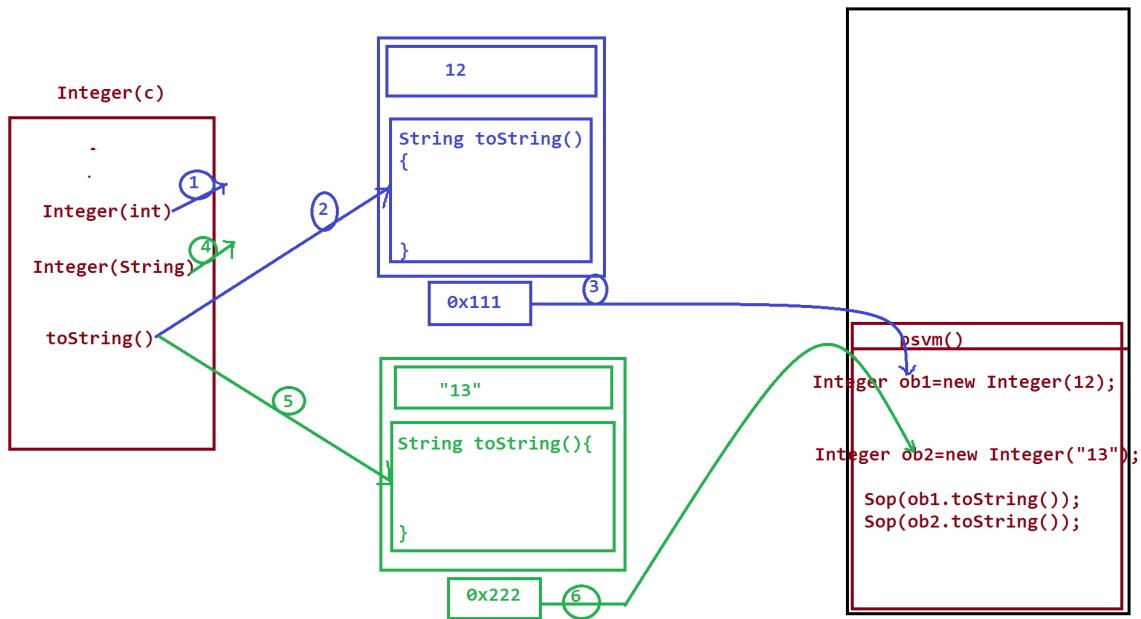
=>**java.lang.NumberFormatException** is raised when we bind data other

than int.

**Exp program:**

```
package maccess;
public class DWrapperClass1 {
    public static void main(String[] args) {
        //Boxing process
        Integer ob1 = new Integer(12);
        Integer ob2 = new Integer("13");
        System.out.println("ob1:"+ob1);
        System.out.println("ob2:"+ob2);
    }
}
```

**Execution flow of above program:**



**Note:**

=>Like Integer WrapperClass Byte,Short,Long,Double and Boolean

WrapperClasses are also having two constructors

1 - to bind its data

2 - to bind its data in the form of String.

**Float WrapperClass:**

=>Float WrapperClass having three constructors

1 - to bind float data

2 - to bind double data

3 - to bind float data in the form of String.

**Constructor Signatures:**

`public java.lang.Float(float);`

`public java.lang.Float(double);`

`public java.lang.Float(java.lang.String) throws`

```
java.lang.NumberFormatException;
```

**Character WrapperClass:**

=>Character WrapperClass having only one constructor to bind char data.

```
public java.lang.Character(char);
```

**Exp program:**

```
package maccess;
public class DWrapperClass2 {
    public static void main(String[] args) {
        //Boxing process
        Float ob1 = new Float(12.34F);
        Float ob2 = new Float(2345.67);
        Float ob3 = new Float("234.56");
        Character ob4 = new Character('A');
        System.out.println("ob1:"+ob1.toString());
        System.out.println("ob2:"+ob2.toString());
        System.out.println("ob3:"+ob3.toString());
        System.out.println("ob4:"+ob4.toString());
    }
}
```

Dt : 15/2/2021

**Case-2 : Boxing process using 'valueOf()'**

=>valueOf() method is a static method from all the WrapperClasses and which is used to perform boxing process.

Exp program:

```
package maccess;
public class DWrapperClass3 {
    public static void main(String[] args) {
        //Boxing process
        Integer ob1 = Integer.valueOf(12);
        Float ob2 = Float.valueOf(12.34f);
        Character ob3 = Character.valueOf('A');
        Boolean ob4 = Boolean.valueOf(true);
        System.out.println("==data from Objects==");
        System.out.println("ob1:" + ob1);
        System.out.println("ob2:" + ob2);
        System.out.println("ob3:" + ob3);
        System.out.println("ob4:" + ob4);
    }
}
```

**Note:**

=>valueOf() methods as a factory method which create objects.

---

**faq:**

**define AutoBoxing process?**

=>The Boxing process which is performed automatically is known as AutoBoxing process.

Exp program:

```
package maccess;
public class DWrapperClass4 {
    public static void main(String[] args) {
        //AutoBoxing process
        Integer ob1 = 12;
        Float ob2 = 12.34F;
        Character ob3 = 'A';
        Boolean ob4 = true;
        System.out.println("==>Data from objects==");
        System.out.println("ob1:" + ob1);
        System.out.println("ob2:" + ob2);
        System.out.println("ob3:" + ob3);
        System.out.println("ob4:" + ob4);
    }
}
```

---

faq:

**define UnBoxing process?**

=>The process of taking primitive datatypes out of WrapperClass objects  
is known as UnBoxing process.

=>The following methods are used to perform UnBoxing process:

```
public byte byteValue();
public short shortValue();
public int intValue();
public long longValue();
```

```
public float floatValue();  
public double doubleValue();  
public char charValue();  
public boolean booleanValue();
```

Ex program:

```
package maccess;  
public class DWrapperClass5 {  
    public static void main(String[] args) {  
        //Boxing process  
        Integer ob1 = new Integer(12);  
        Float ob2 = new Float(12.34F);  
        Character ob3 = new Character('A');  
        Boolean ob4 = new Boolean(true);  
        //UnBoxing process  
        int i = ob1.intValue();  
        float f = ob2.floatValue();  
        char ch = ob3.charValue();  
        boolean b = ob4.booleanValue();  
  
        System.out.println("==>Data from objects==>");  
        System.out.println("i:" + i);  
        System.out.println("f:" + f);  
        System.out.println("ch:" + ch);  
        System.out.println("b:" + b);  
    }  
}
```

---

faq:

define AutoUnBoxing process?

=The UnBoxing process which is performed automatically is known as AutoUnBoxing process.

Exp program:

```
package maccess;
public class DWrapperClass6 {
    public static void main(String[] args) {
        //AutoBoxing process
        Integer ob1 = 12;
        Float ob2 = 12.34F;
        Character ob3 = 'A';
        Boolean ob4 = true;
        //AutoUnBoxing process
        int i = ob1;
        float f = ob2;
        char ch = ob3;
        boolean b = ob4;

        System.out.println("==>Data from objects==");
        System.out.println("i:"+i);
        System.out.println("f:"+f);
        System.out.println("ch:"+ch);
        System.out.println("b:"+b);
    }
}
```

---

Note:

=>AutoBoxing and AutoUnBoxing processes introduced by Java5 version.

=>In FrameWorks like JCF(JavaCollectionFramework) and Hibernate the data must be available in the form of objects,in this process we use

**WrapperClasses to make Primitive datatype available in the form objects.**

**=>All the WrapperClass objects are Immutable objects.(Secured Objects)**

---

**\*imp**

**Strings in Java:**

**=>The sequenced collection of characters which are represented in double quotes is known as String.**

**Exp:**

**"nit","hyd",...**

**=>The following classes from 'java.lang' package are used to create String objects:**

**1.String class**

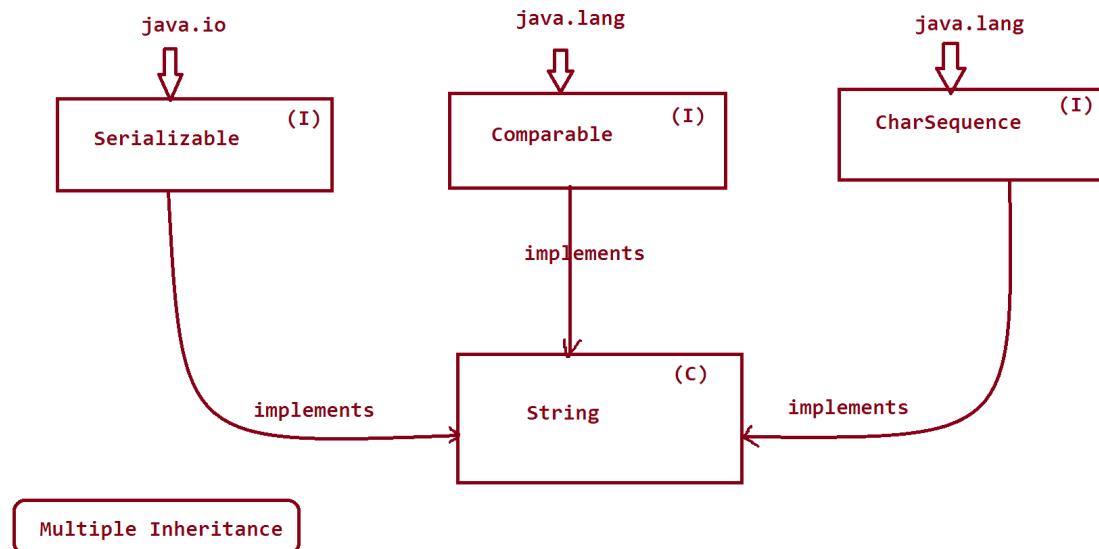
**2.StringBuffer class**

**3.StringBuilder class**

**1.String class:**

**=>The 'String' class is from java.lang package and the objects which are generated from 'String' class are Immutable objects.(Secured)**

**Hierarchy of 'String' class:**



=>'String' class is having 16 Constructors.

=>We use the following two syntaxes to create String class objects:

**syntax-1 : Using String literal process**

```
String s1 = "java";
```

**Syntax-2 : using new operator process**

```
String s2 = new String("program");
```

**Exp program1:**

wap to demonstrate the diff b/w the String Object creation syntaxes?

```
package maccess;
public class DString1 {
    public static void main(String[] args) {
String s1 = "java";
String s2 = new String("program");
System.out.println("s1:"+s1.toString());
System.out.println("s2:"+s2.toString());
    }
}
```

Exp program2:

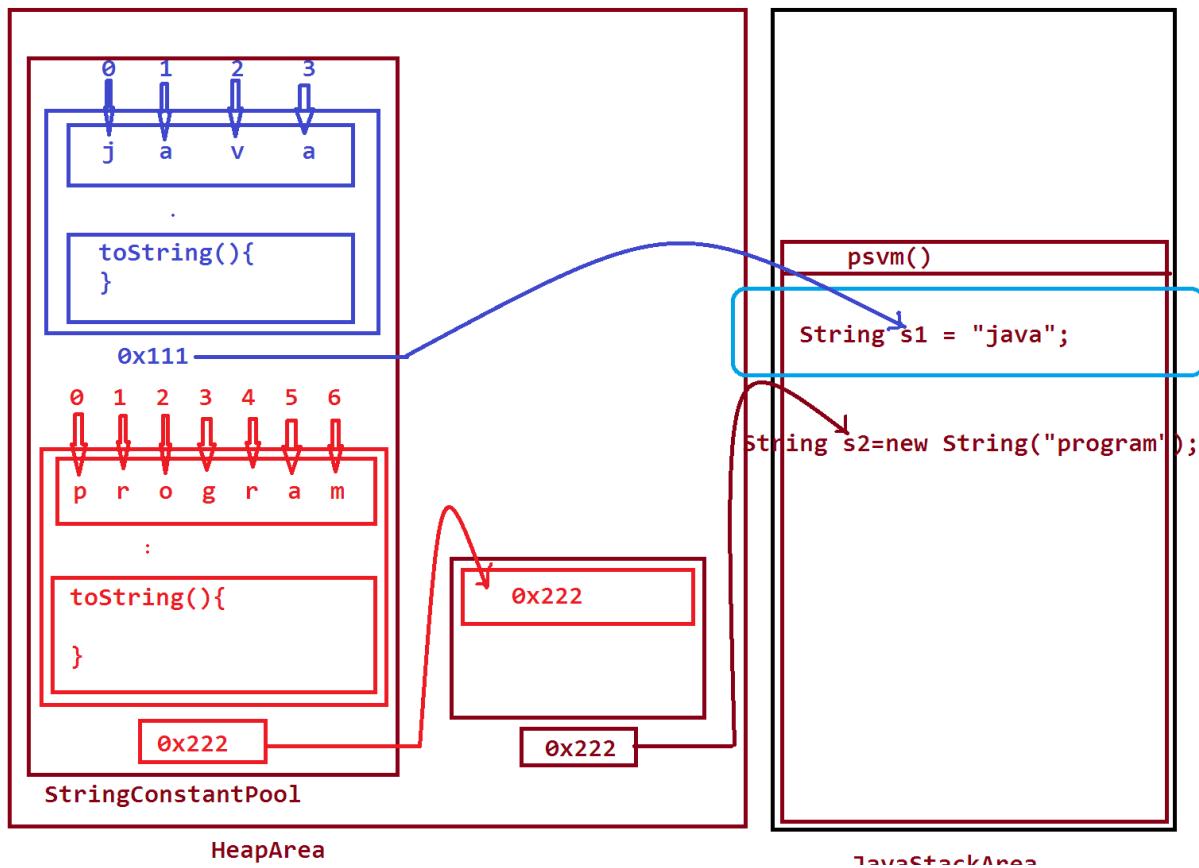
wap to count the number of Vowels from the given String?

```
package maccess;
import java.util.*;
public class DString2 {
    public static void main(String[] args) {
Scanner s = new Scanner(System.in);
System.out.println("Enter the String:");
String str = s.nextLine();
int count=0;
for(int i=0;i<str.length();i++){
    switch(str.charAt(i)){
        case 'a':
        case 'A':count++;
        break;
        case 'e':
        case 'E':count++;
        break;
        case 'i':
        case 'I':count++;
        break;
    }
}
System.out.println("Count of Vowels is "+count);
}
```

```
        case 'o':  
        case 'O':count++;  
        break;  
        case 'u':  
        case 'U':count++;  
        break;  
    }//end of switch  
}//end of loop  
System.out.println("No of Vowels:"+count);  
s.close();  
}  
  
}
```

Dt : 17/2/2021

Execution flow of above program:(DString1.java)



**Note:**

- when execution control finds String Literal process,  
=>Then the execution control will check the StringConstantPool  
is any Object having the same data  
=>If same object is available then the reference of the object  
is used without creating new object.  
=>If same object is not available then the new object is  
created.

(ii) In new operator process one reference is created part of HeapArea  
and the reference is binded with reference of Object created part  
of String Constant pool

Note:

=>we can organize the characters in the String based on index values.

---

Exp program3:

```
package maccess;
public class DString3 {
    public static void main(String[] args) {
String s1 = "java";
String s2 = new String("program");
System.out.println("s1:"+s1.toString());
System.out.println("s2:"+s2.toString());
System.out.println("char at index 2 of
s1:"+s1.charAt(2));
System.out.println("char at index 2 of
s2:"+s2.charAt(2));
System.out.println("length of s1:"+s1.length());
System.out.println("length of s2:"+s2.length());
System.out.println("Substring from 2-5 from
s2:"+s2.substring(2,5));
    }
}
```

define String Concatenation process?

=>The process of combining multiple Strings into a single String is  
known as String Concatenation process.

Note:

=>This Concatenation process can be done using 'concat()' method or using "+" symbol.

Exp program4:

```
package maccess;
public class DString4 {
    public static void main(String[] args) {
        String s1="java";
        String s2="language";
        String s3="programming";
        String s4 = s1.concat(s2);
        String s5 = s1.concat(s3);
        String s6 = s1+s2+s3;
        String s7 = s1.concat(s2).concat(s3);
        System.out.println("s4:"+s4);
        System.out.println("s5:"+s5);
        System.out.println("s6:"+s6);
        System.out.println("s7:"+s7);
    }
}
```

Note:

=>In String Concatenation process separate object is created to hold concatenated strings,because String objects are Immutable objects.

---

define String Comparision process?

=>The process of comparing two strings is known as String Comparision process.

=>This String Comparision can be done in the following ways:

**1.Using equals() method**

**2.Using compareTo() method**

**3.Using 'is equal to'(==) operator**

**1.Using equals() method:**

=>equals() method will compare two strings and generate boolean result.

**Method Signature of equals() method:**

```
public boolean equals(java.lang.Object);  
public boolean equalsIgnoreCase(java.lang.String);
```

**Note:**

=>In realtime equals() method is used in Authentication process.

**2.Using compareTo() method:**

=>compareTo() method also compares two Strings and generate int value.

**Method Signature of compareTo() method:**

```
public int compareTo(java.lang.String);  
public int compareToIgnoreCase(java.lang.String);
```

**syntax:**

```
int k = s1.compareTo(s2);
```

**if k==0 then the Strings are equal**

**if k>0 then s1>s2**

**if k<0 then s1<s2**

**Note:**

=>In realtime compareTo() method is used in Sorting process.

**Exp program5:**

```
package maccess;
import java.util.*;
public class DString5 {
    public static void main(String[] args) {
Scanner s = new Scanner(System.in);
System.out.println("Enter the String1:");
String s1 = s.nextLine().trim();
System.out.println("Enter the String2:");
String s2 = s.nextLine().trim();
System.out.println("====equals()====");
boolean z = s1.equalsIgnoreCase(s2);
    if(z){
System.out.println("Strings are equal...");
    }else{
System.out.println("Strings are NotEqual...");
    }
System.out.println("====compareTo()====");
```

```
int k = s1.compareToIgnoreCase(s2);
if(k==0){
System.out.println("Strings are equal...");
}else{
System.out.println("Strings are NotEqual...");
}
s.close();
}

}
```

### Assignment:

1.wap to display reverse of give String without using Built-In method?

2.wap to display the following format:

A

A E

A E I

A E I O

A E I O U

3.wap to display the sum of numbers from the given String?

I/P :

"java8 by 2014"

O/P :

sum = 8+2+0+1+4

= 15

Dt : 18/2/2021

Note:

=>'trim()' method is used to remove the spaces before and after the Strings.

=>IgnoreCase specify to check the content but not the case.

---

### 3.Using 'is equal to'(==) operator:

=>'is equal to'(==) operator will compare the references of an object and which will not compare the content of an Object.

Exp program:

```
package maccess;
public class DString6 {
    public static void main(String[] args) {
String s1 = "java";//AutoBoxing process
String s2 = "java";//AutoBoxing process
String s3 = new String("java");//Boxing process
String s4 = new String("java");//Boxing process
System.out.println("====String Literal process====");
    if(s1==s2){
System.out.println("Strings are equal...");
    }else{
System.out.println("Strings are NotEqual...");
    }
System.out.println("====new operator process====");
    if(s3==s4){
System.out.println("Strings are equal...");
    }else{
System.out.println("Strings are NotEqual...");
    }
    }
}
```

---

**Note:**

=>"is equal to" operator must not be used on NonPrimitive datatypes, because which compares references and may generate wrong results.

---

**faq:**

**define String Constant pool?**

=>The separate partition of HeapArea where the String objects are created is known as String Constant pool.

---

**2.StringBuffer class:**

=>StringBuffer class is from java.lang package and the objects which are generated from StringBuffer class are Mutable objects.

=>StringBuffer provides the following 4 constructors:

```
public java.lang.StringBuffer();
public java.lang.StringBuffer(int);
public java.lang.StringBuffer(java.lang.String);
public java.lang.StringBuffer(java.lang.CharSequence);
```

Dt : 19/2/2021

**Case-1 : using constructor 'public java.lang.StringBuffer()'**

=>In this syntax the StringBuffer object is created with the default capacity 16.(characters)

=>we use append() method to add the data to the StringBuffer object.

=>The capacity of StringBuffer object increases dynamically by doubling the capacity and adding 2.

16-->(16+16+2)-->34-->(34+34+2)-->70...

=>we use reverse() and insert() methods on StringBuffer object,because StringBuffer object is mutable object.

Exp program:

DBuffer1.java

```
package maccess;
public class DBuffer1 {
    public static void main(String[] args) {
StringBuffer sb = new StringBuffer();
System.out.println("default
capacity:"+sb.capacity());
System.out.println("length:"+sb.length());
sb.append("java");
sb.append("programming");
System.out.println("Data:"+sb.toString());
System.out.println("capacity:"+sb.capacity());
System.out.println("length:"+sb.length());
sb.append("language");
System.out.println("Data:"+sb.toString());
System.out.println("capacity:"+sb.capacity());
System.out.println("length:"+sb.length());
sb.insert(4,"Java8");
```

```

System.out.println("Data:"+sb.toString());
System.out.println("capacity:"+sb.capacity());
System.out.println("length:"+sb.length());
System.out.println("Reverse:"+sb.reverse());
}
}

```

**Case-2 : using constructor 'public java.lang.StringBuffer(int)'**

=>In this syntax the StringBuffer object is created with the capacity specified by the value passed as parameter while object creation.

Exp program:

DBuffer2.java

```

package maccess;
public class DBuffer2 {
    public static void main(String[] args) {
StringBuffer sb = new StringBuffer(5);
System.out.println("default
capacity:"+sb.capacity());
System.out.println("length:"+sb.length());
sb.append("java");
System.out.println("Data:"+sb);
System.out.println("capacity:"+sb.capacity());
System.out.println("length:"+sb.length());
sb.append("program");
System.out.println("Data:"+sb);
System.out.println("capacity:"+sb.capacity());
System.out.println("length:"+sb.length());
sb.append("lang");
System.out.println("Data:"+sb);
System.out.println("capacity:"+sb.capacity());

```

```
System.out.println("length:"+sb.length());
    }
}
```

### Case-3 : using constructor

'public java.lang.StringBuffer(java.lang.String)'

=>In this syntax the StringBuffer object is created with the capacity equal to sum of default capacity and length of String passed as parameter while object creation.

Exp program:

DBuffer3.java

```
package maccess;
public class DBuffer3 {
    public static void main(String[] args) {
StringBuffer sb = new StringBuffer("AA");
System.out.println("default
capacity:"+sb.capacity());
System.out.println("length:"+sb.length());
sb.append("javalanguageprogram");
System.out.println("Data:"+sb);
System.out.println("capacity:"+sb.capacity());
System.out.println("length:"+sb.length());
StringBuffer sb1 = new StringBuffer(sb);
System.out.println("Data from sb1:"+sb1);
    }
}
```

### Case-4 : using constructor

'public java.lang.StringBuffer(java.lang.CharSequence)'

=>This syntax is used to link two StringBuffer objects.

---

**Note:**

=>**StringBuffer class is Synchronized class.**

**faq:**

**define Synchronized class?**

=>**The class which is declared with synchronized methods is known as Synchronized class.**

**faq:**

**define Synchronized methods?**

=>**The methods which are declared with synchronized keyword are known as Synchronized methods.**

**faq:**

**wt is the advantage of Synchronized methods?**

=>**The Synchronized methods will be under the lock and the methods are available to one user at a time.**

**(Secured methods)**

**3.StringBuilder class:**

=>**StringBuilder class is from java.lang package and the objects which are generated from StringBuilder class are Mutable objects.**

=>**StringBuilder provides the following 4 constructors:**

```
public java.lang.StringBuilder();
public java.lang.StringBuilder(int);
public java.lang.StringBuilder(java.lang.String);
public java.lang.StringBuilder(java.lang.CharSequence);
```

**Note:**

=>**StringBuilder class is same like StringBuffer, but StringBuilder**

**class is NonSynchronized class, which means the methods which are declared within the StringBuilder are NonSynchronized methods.**

---

**Note:**

=>StrinBuffer is used in MultiThreading applications and StringBuilder is used in Non-MultiThreading applications.

---

\*imp

**(c)Arrays in Java:**

=>The sequenced collection of elements of same datatype is known as **Array**.

(or)

=>The Sequenced Collection of Similer elements.

(or)

=>The Sequenced Collection of objects of Same class.

(or)

=>The Sequenced Collection of Similer objects

Arrays in Java are categorized into two types:

**1.Single Dimensional Arrays**

**2.Multi Dimensional Arrays**

**1.Single Dimensional Arrays:**

=>The Arrays which are declared with one dimension are known as **S-D Arrays**.

**syntax:**

**Class\_name arr\_var[] = new Class\_name[size];**

Exp:

```
Integer i[] = new Integer[3];
```

```
String str[] = new String[3];
```

```
Test t[] = new Test[3];
```

```
Object o[] = new Object[3];
```

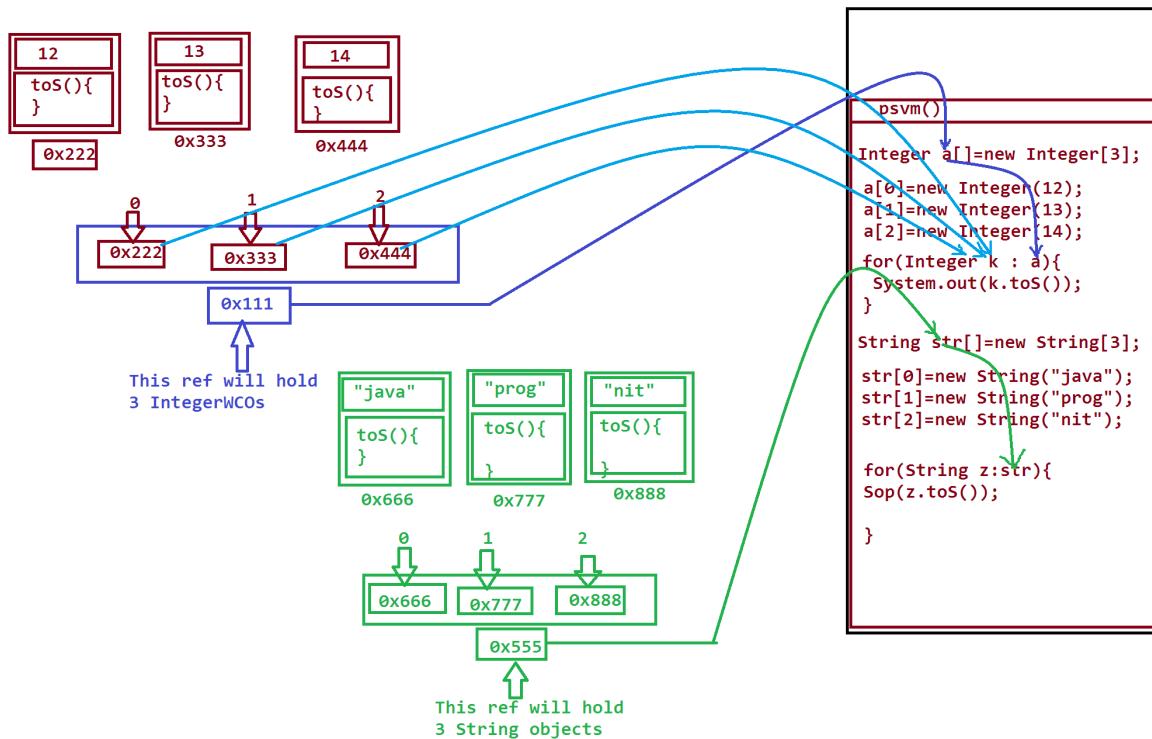
Exp program1:

wap to read and display Integer WrapperClass objects and String Objects using Array?

```
package maccess;
public class DArray1 {
    public static void main(String[] args) {
        Integer a[] = new Integer[3]; //Integer Array
        a[0] = new Integer(12);
        a[1] = new Integer(13);
        a[2] = new Integer(14);
        System.out.println("====Integer Array====");
        for(Integer k : a){
            System.out.println(k.toString());
        } //end of loop
        String str[] = new String[3];
        str[0] = new String("java");
        str[1] = new String("program");
        str[2] = new String("nit");
        System.out.println("====String Array====");
        for(String z : str){
            System.out.println(z.toString());
        } //end of loop
    }
}
```

Dt : 20/2/2021

Execution flow of above program:



Exp program2:

wap to read and display multiple Employee details?

```

package test;
public class Employee {
    public String name, id;
    public Address ad = new Address();
    public String toString()
    {
        return name + "\t" + id + "\t" + ad.toString();
    }
}

```

```
package test;
public class Address {
    public String hNo,sName,city;
    public int pinCode;
    public String toString()
    {
return hNo+"\t"+sName+"\t"+city+"\t"+pinCode;
    }
}
```

```
package maccess;

import test.*;
import java.util.*;

public class DArray2 {

    public static void main(String[] args) {

try(Scanner s = new Scanner(System.in));//Java7

{
    try{

        System.out.println("Enter the number of employees:");
        int n = Integer.parseInt(s.nextLine());
        Employee e[] = new Employee[n];
        System.out.println("Enter details of "+n+" employees:");
        for(int i=0;i<e.length;i++){

            e[i]=new Employee();
            System.out.println("Enter the eName:");
            e[i].name = s.nextLine();
        }
    }
}
}
}
```

```
System.out.println("Enter the eId:");
e[i].id = s.nextLine();

System.out.println("Enter the hNo:");
e[i].ad.hNo = s.nextLine();

System.out.println("Enter the sName:");
e[i].ad.sName = s.nextLine();

System.out.println("Enter the city:");
e[i].ad.city = s.nextLine();

System.out.println("Enter the pinCode:");
e[i].ad.pinCode = Integer.parseInt(s.nextLine());

}//end of loop

System.out.println("==Display Employee details==");

for(Employee z : e){

    System.out.println(z.toString());
}

}//end of loop

}//end of try

catch(InputMismatchException ime){

    System.out.println("Enter only Integer value..");

}

}//end of try

}

Dt:22/2/2021
```

**Execution flow of above program:**

**Address.class**

**Employee.class**

**DArray.class**

**Dt : 23/2/2021**

**define Object Array?**

**=>The Array which is declared with java.lang.Object class is known as Object Array.**

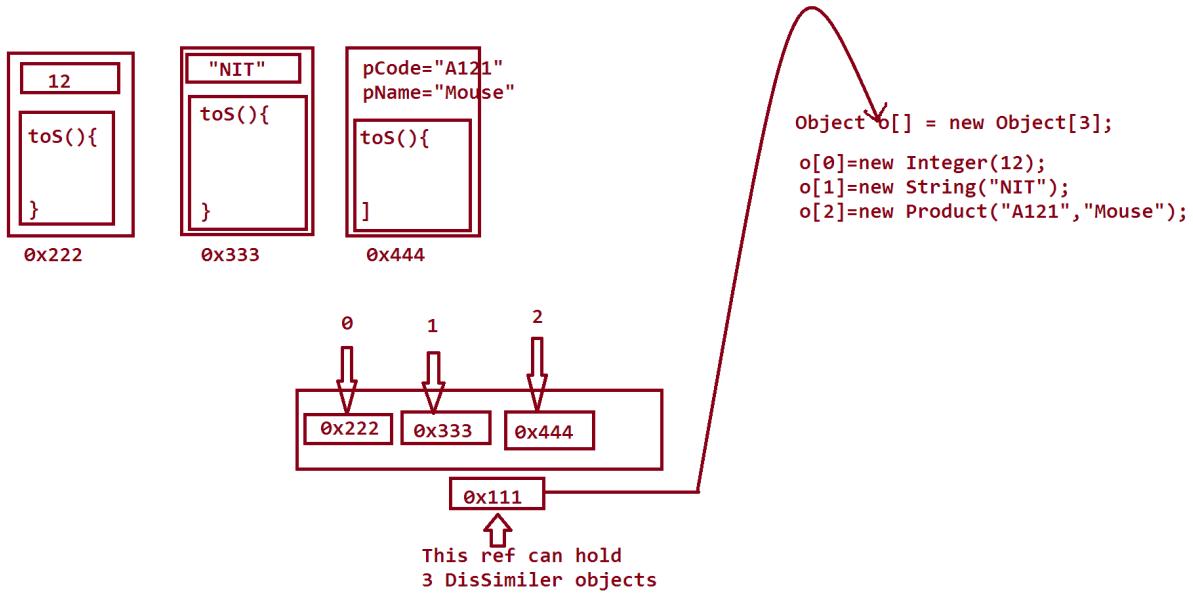
**Note:**

**=>This Object Array can hold Dis-Similar objects or Objects of different classes.**

**Exp program:**

```
package maccess;
public class DArray3 {
    public static void main(String[] args) {
Object o[] = new Object[3];//Object Array
o[0] = new Integer(12);
o[1] = new String("NIT");
o[2] = new test.Product("A121","Mouse");
System.out.println("====Display from Object
Array====");
    for(Object k : o){
        System.out.println(k);
    }//end of loop
    }
```

}



## 2. Multi Dimensional Arrays:

=>The arrays which are declared with multiple dimensions are known as

**Multi Dimensional Arrays.**

**Exp:**

**2-D Arrays**

**3-D Arrays**

**4-D Arrays**

...

**Note:**

=>In Java Multi-D Arrays are less used when compared to Single-D Arrays.

=>We use 2-D Arrays to construct 'Jagged Arrays'.

**syntax of 2-D Arrays:**

```
Class_name arr_var[][] = new Class_name[rows][cols];
```

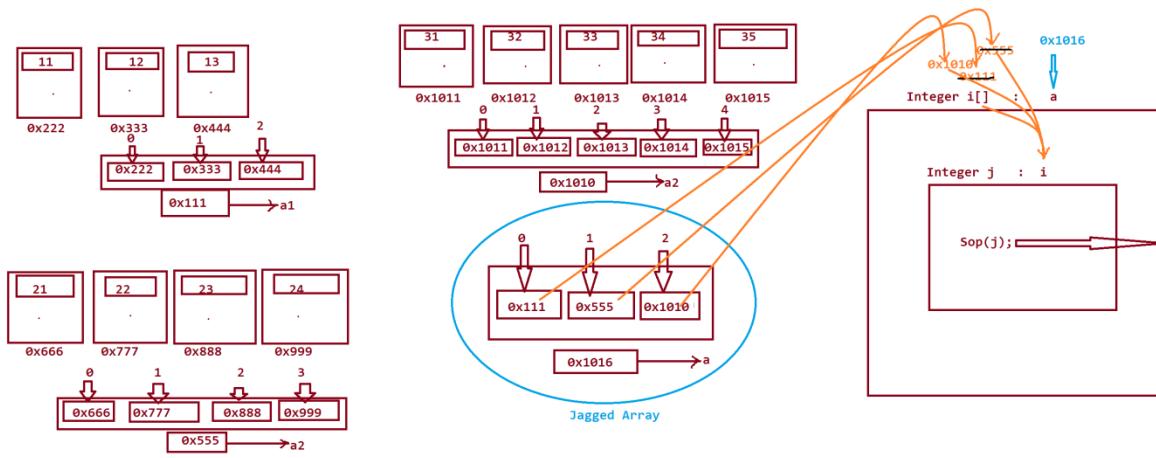
**define Jagged Array?**

=>The Array which is collection of arrays is known as Jagged Array.

**Exp program:**

```
package maccess;
public class DArray4 {
    public static void main(String[] args) {
        Integer a1[] = {11,12,13};
        Integer a2[] = {21,22,23,24};
        Integer a3[] = {31,32,33,34,35};
        Integer a[][] = {a1,a2,a3};
        System.out.println("==>Display from Jagged
        Array==>");
        for(Integer i[] : a){
            for(Integer j : i){
                System.out.print(j+" ");
            } //InnerLoop
            System.out.println();
        } //OuterLoop
    }
}
```

## Diagram of Jagged Array:



## DisAdvantage of Arrays:

=>Array size once defined cannot be modified at runtime ,because of this reason Arrays are not preferable to hold Dynamic data or runtime data.

## Note:

=>The DisAdvantage of Array can be overcomed using Collection.

\*imp

## Java Collection Framework(JCF):

**define Collection<E>?**

=>**Collection<E>** is an interface from **java.util** package and which is root of Java Collection Framework.

=>This **Collection<E>** interface is extended into the following SubInterfaces:

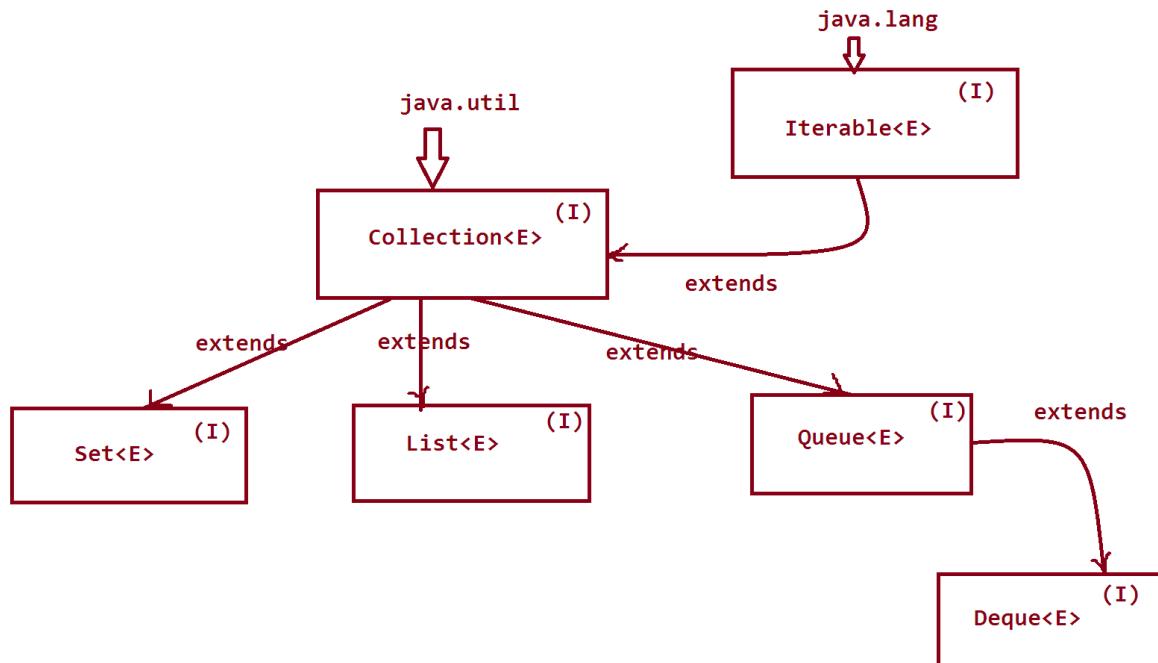
1.**Set<E>**

2.**List<E>**

3.**Queue<E>**

=>**Deque<E>**

**Hierarchy of Collection<E>:**



**define Framework?**

**=>The Structure which is ready available to construct applications is known as Framework.**

**1.Set<E>:**

**=>Set<E> organizes elements without index values and which cannot hold duplicate elements.**

**=>The following are some important methods od Set<E>:**

**public abstract int size();**

**public abstract boolean isEmpty();**

**public abstract boolean contains(java.lang.Object);**

**public abstract boolean add(E);**

**public abstract boolean remove(java.lang.Object);**

**public abstract java.lang.Object[] toArray();**

**public abstract java.util.Iterator<E> iterator();**

**public java.util.Spliterator<E> spliterator();**

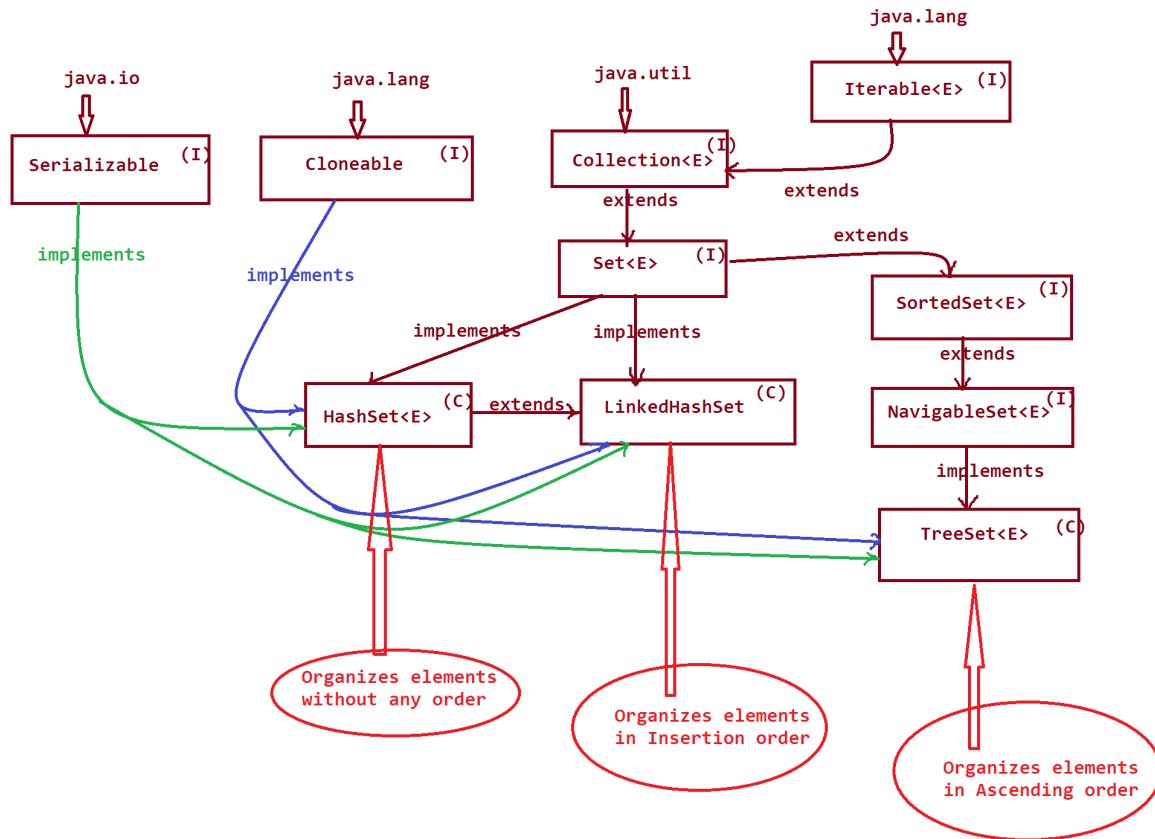
**=>The following are the implementation classes:**

**(a) HashSet<E>**

**(b) LinkedHashSet<E>**

### (c)TreeSet<E>

Hierarchy of implementation classes:



Note:

=>HashSet<E> organizes elements without any order.

=>LinkedHashSet<E> organizes elements in insertion order.

=>TreeSet<E> organizes elements in Ascending order.

Exp program:

```
package maccess;  
import java.util.*;
```

```
import test.Product;

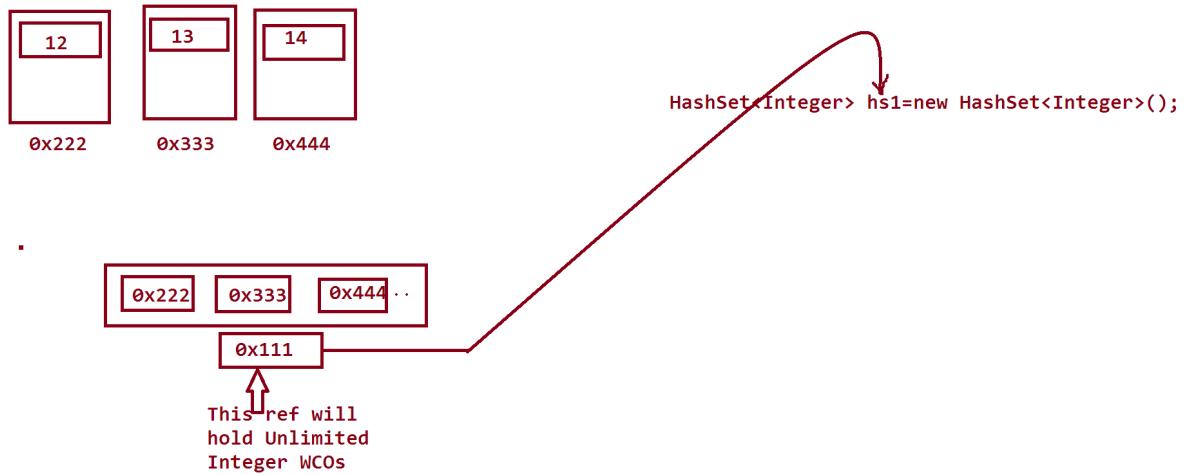
public class DSet1 {

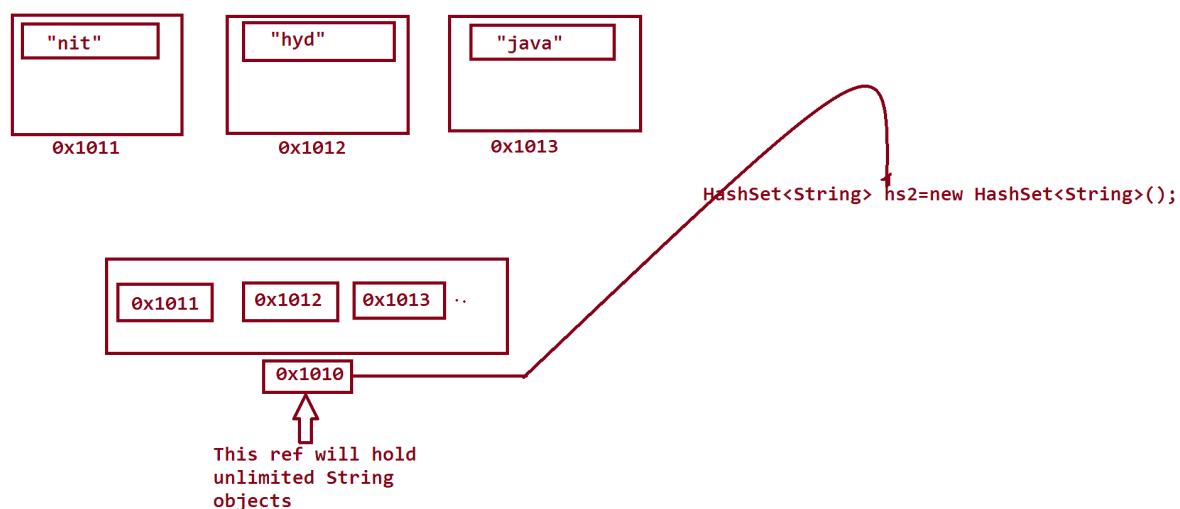
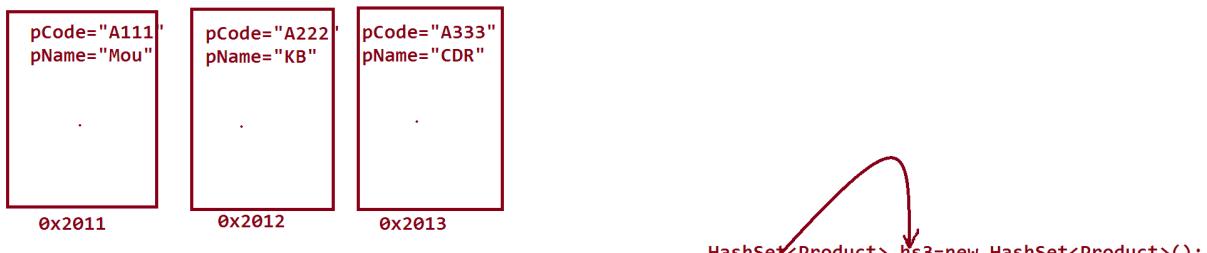
    public static void main(String[] args) {

HashSet<Integer> hs1 = new LinkedHashSet<Integer>();
hs1.add(new Integer(12));
hs1.add(13);
hs1.add(new Integer(14));
System.out.println("==hs1==");
System.out.println(hs1.toString());
HashSet<String> hs2 = new LinkedHashSet<String>();
hs2.add(new String("nit"));
hs2.add("hyd");
hs2.add(new String("java"));
System.out.println("==hs2==");
System.out.println(hs2.toString());

HashSet<Product> hs3 = new LinkedHashSet<Product>();
hs3.add(new Product("A111","Mou"));
hs3.add(new Product("A222","KB"));
hs3.add(new Product("A333","CDR"));
System.out.println("==hs3==");
System.out.println(hs3.toString());
    }
}
```

Diagram of Set<E> object:



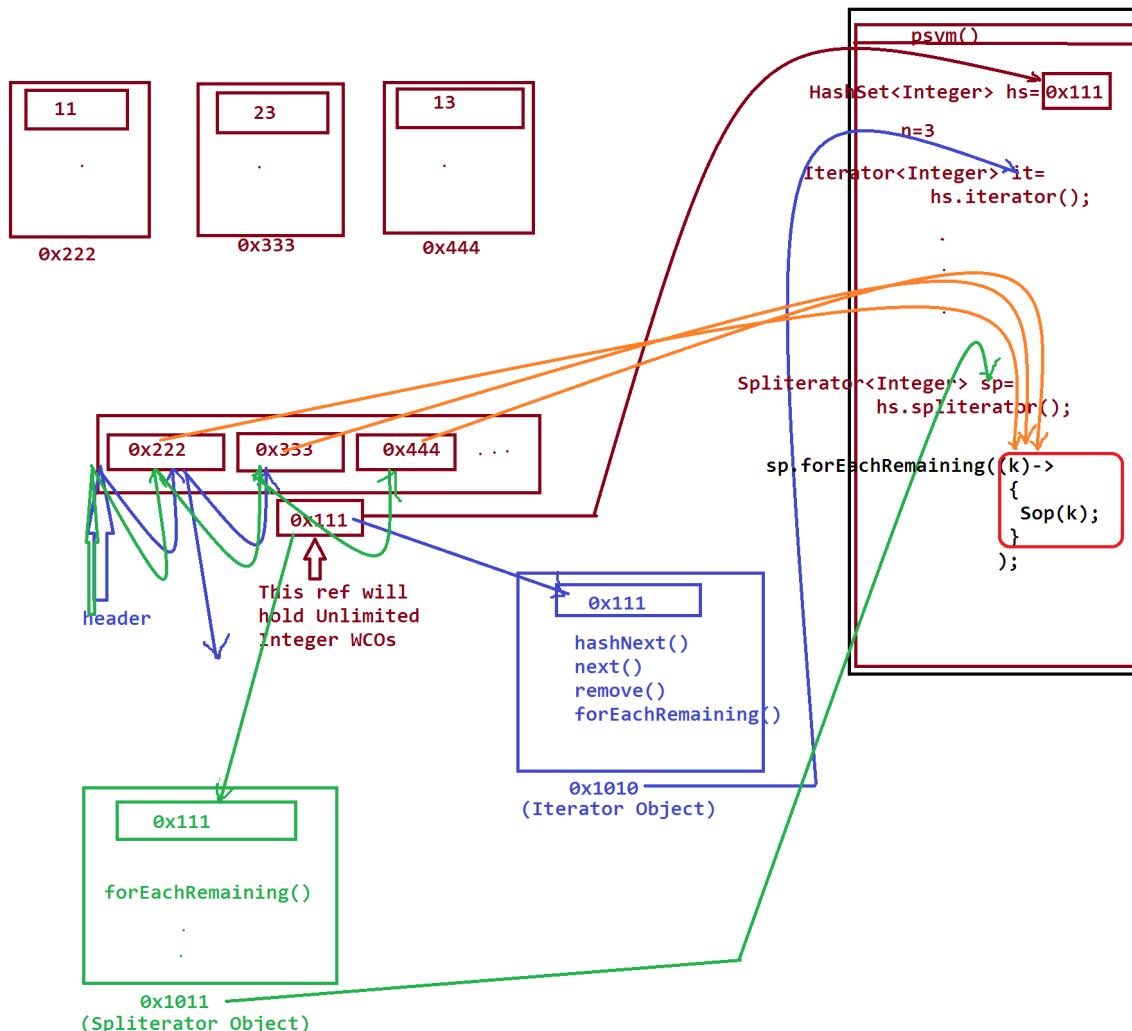


```
package maccess;
import java.util.*;
public class DSet2 {
    public static void main(String[] args) {
try(Scanner s = new Scanner(System.in)){
TreeSet<Integer> hs = new TreeSet<Integer>();
```

```
try{
System.out.println("Enter the no of elements:");
int n = s.nextInt();
System.out.println("Enter "+n+" elements:");
for(int i=1;i<=n;i++){
    hs.add(new Integer(s.nextInt()));
}//end of loop
System.out.println("==Display using
Iterator<E>==");
Iterator<Integer> it = hs.iterator();
while(it.hasNext()){
    System.out.print(it.next()+" ");
}//end of loop
System.out.println("\n==Display using
Spliterator<E>==");
Spliterator<Integer> sp = hsspliterator();
sp.forEachRemaining((k)->
{
    System.out.print(k+" ");
});
}catch(InputMismatchException ie){
    System.out.println("Enter only Integer
values..");
}
}//end of try
}
```

Dt : 25/2/2021

Execution flow of above program:



define **Iterator<E>**?

=>**Iterator<E>** is an interface from **java.util** package and which is used to retrieve elements from **Collection<E>** object.

=>The following are some methods from **Iterator<E>**:

```
public abstract boolean hasNext();
```

=>This method is used to check the availability of element.

**public abstract E next();**

=>This method will retrieve the element.

**public void remove();**

=>This method is used to remove the element.

**public void forEachRemaining(java.util.function.Consumer<? super E>);**

=>This method introduced by Java8 version and which is passed  
with LambdaExpression as method\_argument

**define iterator() method?**

=>iterator() method will create the implementation object of  
java.util.Iterator<E> interface and the object is binded with the  
reference of Collection<E> object,in this process it also generates  
header pointing before the first element of Collection object.

**Assignment:**

**wap to calculate result for multiple Students using Collection<E>  
object?**

**Dt : 1/2/2021**

**define Spliterator<T>?**

**=>Spliterator<T> is an interface from java.util package introduced**

**by Java8 version and which is used to retrieve elements from Array objects, Collection objects and Stream objects.**

**One important method from Spliterator<T>:**

```
public void forEachRemaining(java.util.function.Consumer<? super ?>);
```

**define spliterator() method?**

=> spliterator() method will create the implementation object of java.util.Spliterator<T> interface and the object is binded with the reference of Collection<E> object, in this process it also generates header pointing before the element of Collection Object.

**define Consumer<T>?**

=> Consumer<T> is a functional interface introduced by Java8 version and which provides abstract method signature to hold LambdaExpression passes as parameter to the forEachRemaining() method.

**Structure of Consumer<T>:**

```
public interface java.util.function.Consumer<T>
{
    public abstract void accept(T);
}
```

---

**\*imp**

**2.List<E>:**

**=>List<E> organizes elements with index values and can hold duplicate elements.**

**=>The following are some important methods from List<E>**

```
public abstract int size();
public abstract boolean isEmpty();
public abstract boolean contains(java.lang.Object);
public abstract boolean add(E);
public abstract boolean remove(java.lang.Object);

public abstract void add(int,E);
public abstract E remove(int);
public abstract E get(int);
public abstract E set(int,E);
public abstract int indexOf(java.lang.Object);
public abstract int lastIndexOf(java.lang.Object);

public abstract java.lang.Object[] toArray();
```

```

public abstract java.util.Iterator<E> iterator();
public abstract java.util.ListIterator<E> listIterator();
public java.util.Spliterator<E> spliterator();

```

=>The following are the implementation classes of List<E>:

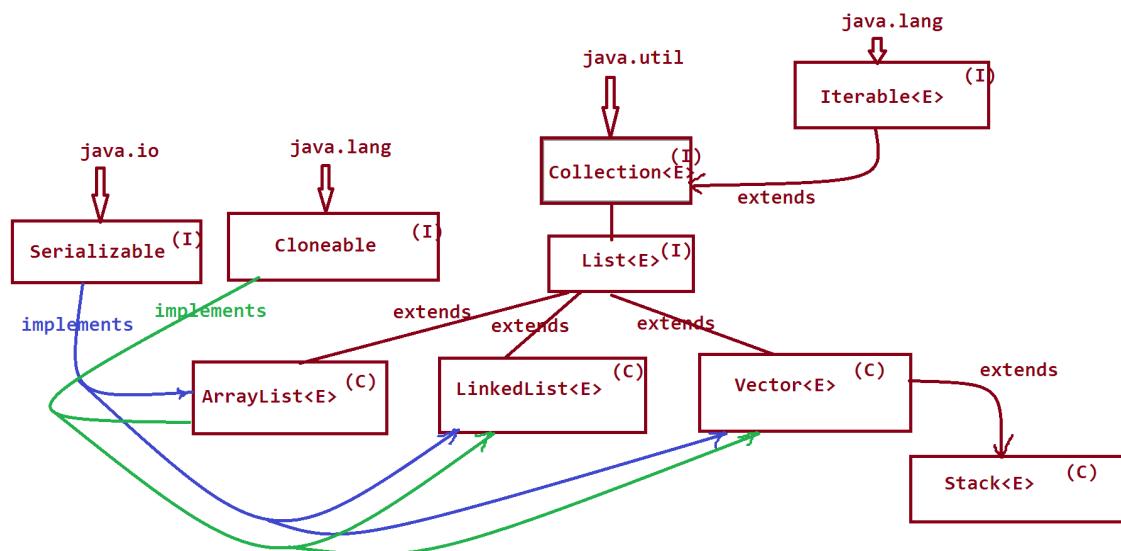
(a)ArrayList<E>

(b)LinkedList<E>

(c)Vector<E>

=>Stack<E>

**Hierarchy of implementation classes:**



**(a)ArrayList<E>:**

=>`ArrayList<E>` organizes elements in sequence and which is

**NonSynchronized class.**

syntax:

```
ArrayList<Class_name> al = new ArrayList<Class_name>();
```

Exp program:

```
package maccess;
import java.util.*;
public class DList1 {
    public static void main(String[] args) {
        ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(new Integer(12));
        al.add(new Integer(14));
        al.add(new Integer(11));
        al.add(new Integer(91));
        System.out.println(al);
        System.out.println("==>ListIterator<E>==");
        ListIterator<Integer> li = al.listIterator();
        System.out.print("Forward : ");
        while(li.hasNext()){
            System.out.print(li.next()+" ");
        } //end of loop
        System.out.print("\nBackward : ");
        while(li.hasPrevious()){
            System.out.print(li.previous()+" ");
        } //end of loop
    }
}
```

define ListIterator<E>?

=>ListIterator<E> is an interface from java.util package and which is

**used to retrieve elements from List<E> objects in both directions.  
(forward and backward)**

**=>The following are some important methods from ListIterator<E>:**

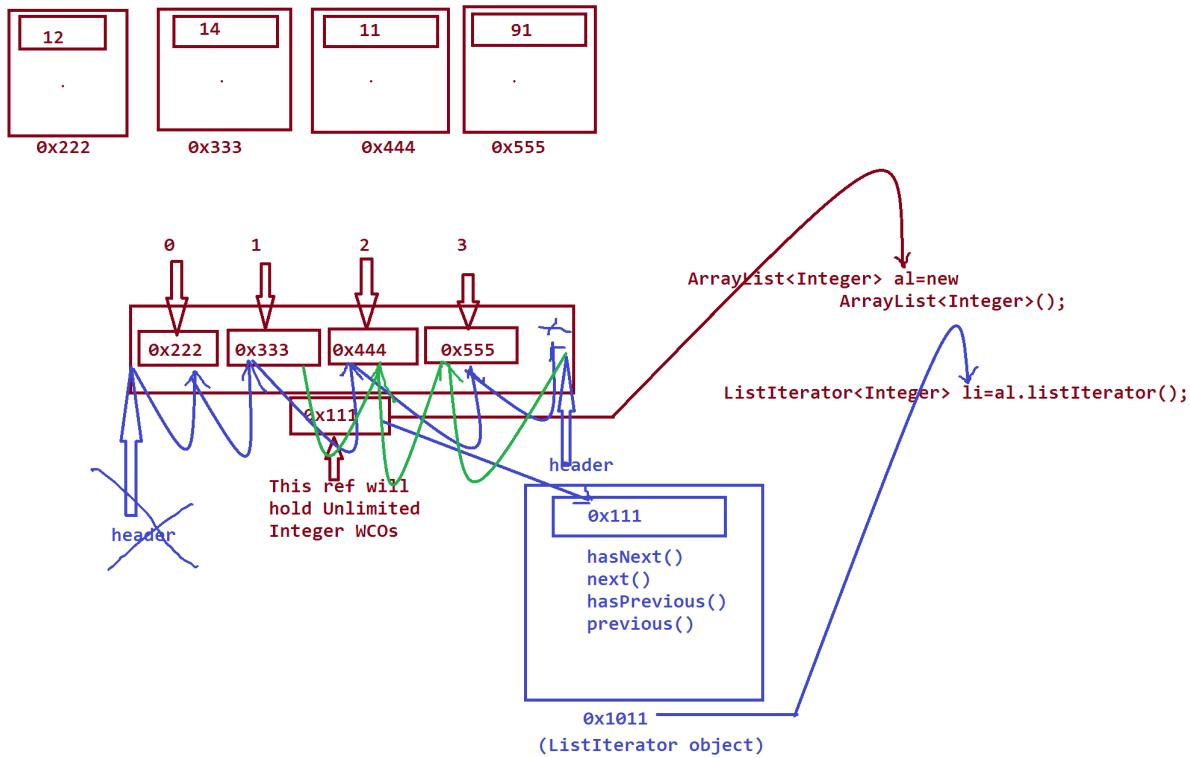
```
public abstract boolean hasNext();  
public abstract E next();  
public abstract boolean hasPrevious();  
public abstract E previous();
```

**define listIterator() method?**

**=>listIterator() method will create the implementation object of  
java.util.ListIterator<E> interface and the object is binded with the  
reference of List<E> object,in this process one header is generated  
pointing before the first element of List<E> object.**

**Dt : 3/3/2021**

**Diagram of List<E> object:**



**faq:**

**wt is the diff b/w**

**(i)Iterator<E>**

**(ii)ListIterator<E>**

=>`Iterator<E>` is used to retrieve elements from `Collection<E>` objects

in forward direction.

=>`ListIterator<E>` is used to retrieve elements from only `List<E>` objects in both directions(forward and backward).

**Exp program:**

```

package maccess;
import java.util.*;

```

```
public class DList2 {  
    public static void main(String[] args) {  
        LinkedList<Integer> al = new LinkedList<Integer>();  
        al.add(new Integer(12));  
        al.add(new Integer(20));  
        al.add(new Integer(17));  
        al.add(new Integer(13));  
        al.add(new Integer(12));  
        System.out.println(al);  
        al.add(2, new Integer(500));  
        System.out.println(al);  
        al.remove(2);  
        System.out.println(al);  
        System.out.println("Display ele at index  
2:"+al.get(2));  
        al.set(2,600);  
        System.out.println(al);  
        System.out.println("Index value of  
12:"+al.indexOf(12));  
        System.out.println("LastIndex value of  
12:"+al.lastIndexOf(12));  
  
    }  
}
```

#### DisAdvantage of ArrayList<E>:

=>In ArrayList<E> the elements are available in Sequence,because of this reason when we add element by index,then the elements are moved in backward direction and when we remove ele by index then the eles are moved forward direction.In this process some execution time is wasted

**in moving the eles in forward and backwrad,which degrades the performance of an application.**

**Note:**

=>**In realtime ArrayList<E> is not preferable where we have more number of insertions and deletions.**

**Exp:**

**Admin login of an application**

**Note:**

=>**The DisAdvantage of ArratyList<E> can be overcomed using  
LinkedList<E>**

**(b)LinkedList<E>:**

=>**LinkedList<E> will hold elements in NonSequence and which is  
NonSynchronized class.**

**syntax:**

**LinkedList<Class\_name> ll = new LinkedList<Class\_name>();**

=>**In LinkedList<E> the elements are available in the form of nodes  
and the node is divided into three parts:**

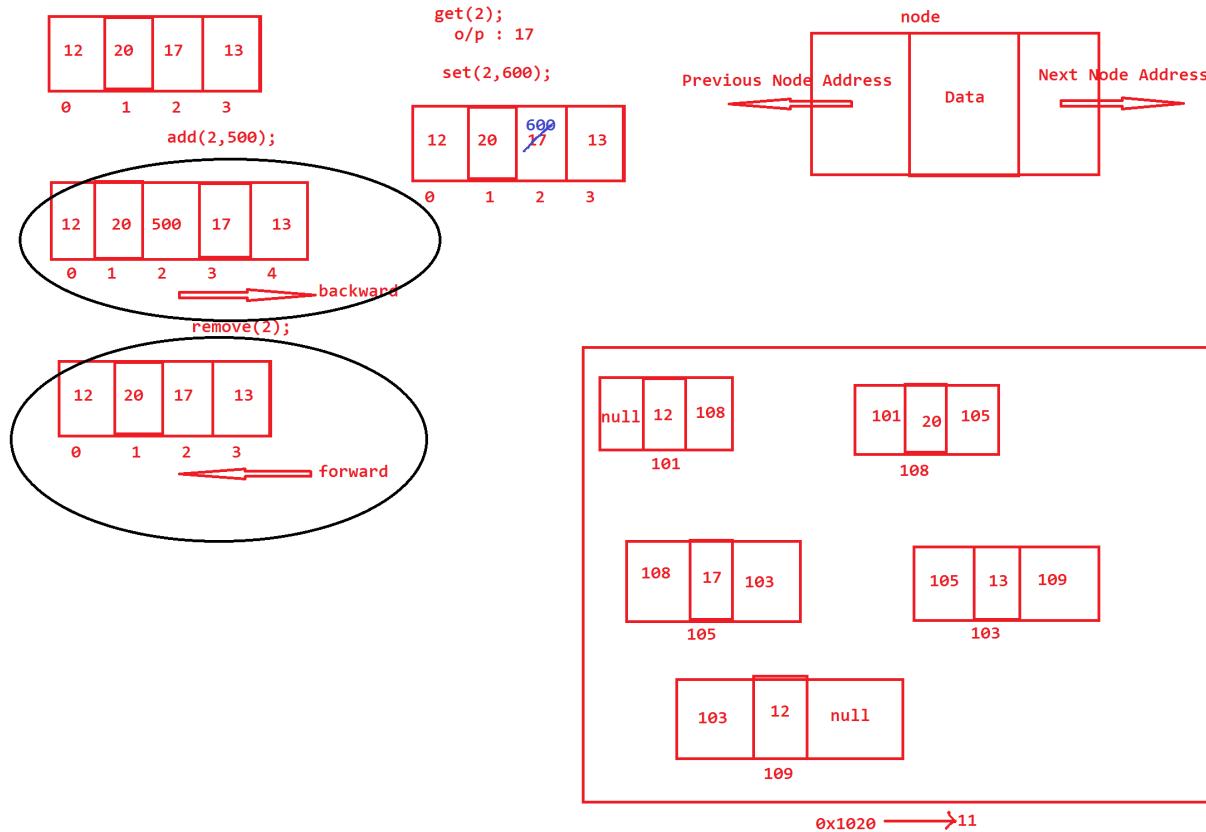
**(i)Previous Node Address**

**(ii)Data**

**(iii)Next Node Address**

**Note:**

=>In realtime **LinkedList<E>** is used where we have more number of insertions and deletions.



**(c)Vector<E>:**

=>**Vector<E>** also organizes elements in Sequence but which is Synchronized class.

**syntax:**

```
Vector<Class_name> v = new Vector<Class_name>();
```

Some important methods from Vector<E>:

```
public synchronized E elementAt(int);
public synchronized E firstElement();
public synchronized E lastElement();
public synchronized void setElementAt(E,int);
public synchronized void removeElementAt(int);
public synchronized void insertElementAt(E,int);
public synchronized void addElement(E);
public synchronized boolean removeElement(java.lang.Object);

public java.util.Enumeration<E> elements();
```

Exp program:

```
package maccess;
import java.util.*;
public class DVector {
    public static void main(String[] args) {
Vector<Integer> v = new Vector<Integer>();
v.addElement(new Integer(12));
v.addElement(new Integer(13));
v.addElement(new Integer(110));
v.addElement(new Integer(120));
System.out.println(v);
System.out.println("==>Enumeration<E><==");
Enumeration<Integer> e = v.elements();
while(e.hasMoreElements()){
    System.out.print(e.nextElement()+" "});
```

```
}

System.out.println("\nFirstElement:" + v.firstElement());
System.out.println("LastElement:" + v.lastElement());
}

}
```

**define Enumeration<E>?**

=>**Enumeration<E>** is an interface from **java.util** package and which is used to retrieve elements from **Vector<E>** objects.

**Methods of Enumeration<E>:**

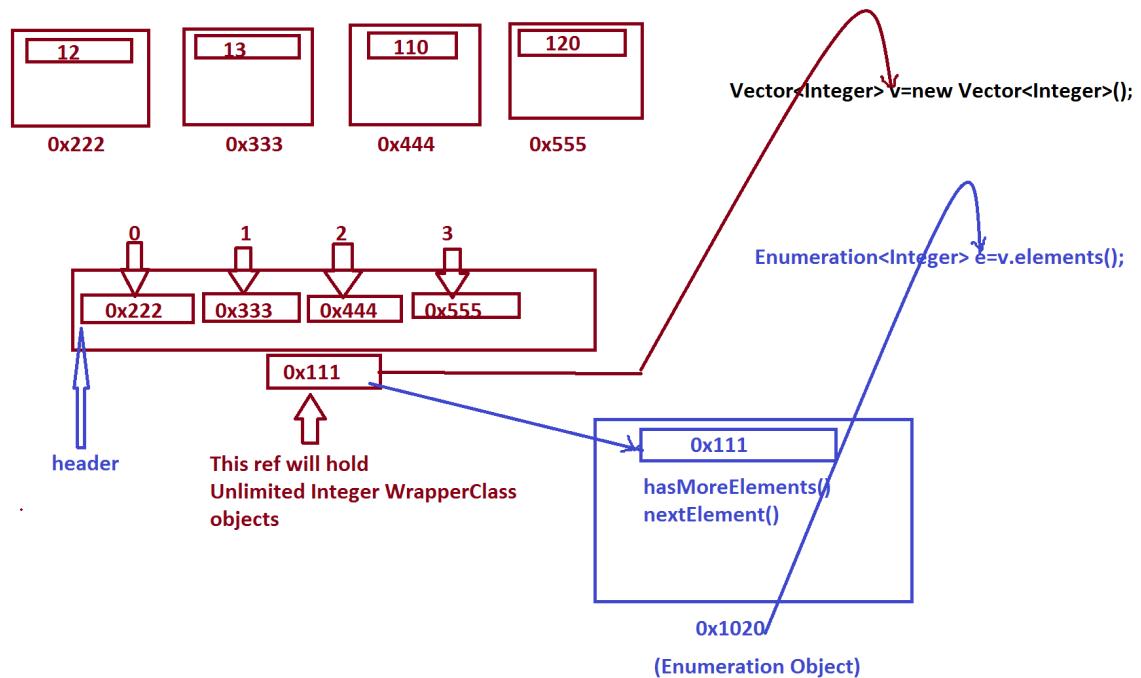
**public abstract boolean hasMoreElements();**  
**public abstract E nextElement();**

**define elements() method?**

=>**elements()** method will create the implementation object of **java.util.Enumeration<E>** interface and the object is binded with the reference of **Vector<E>** object,in this process one header is generates pointing before the first element of **Vector<E>** object.

Dt : 4/3/2021

### Diagram of Enumeration<E>:



### Note:

=>In realtime `Enumeration<E>` is used part of Servlet programming.

=>In realtime `Vector<E>` is used part of Connection pooling concept.

(Connection pooling means organizing multiple DataBase Connections)

---

define `Stack<E>`?

=>`Stack<E>` is a child class of `Vector<E>` and which organizes elements based on the algorithm First-In-Last-Out or Last-In-First-Out.

syntax:

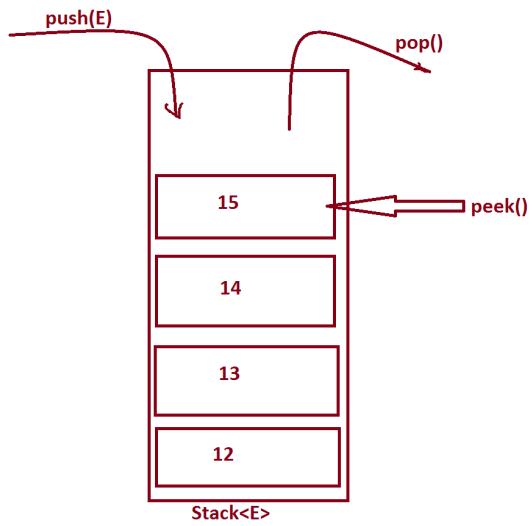
```
Stack<Class_name> st = new Stack<Class_name>();
```

=>The following some important methods of Stack:

```
public E push(E);
public synchronized E pop();
public synchronized E peek();
public boolean empty();
public synchronized int search(java.lang.Object);
```

Exp program:

```
package maccess;
import java.util.*;
public class DStack {
    public static void main(String[] args) {
Stack<Integer> st = new Stack<Integer>();
st.push(new Integer(12));
st.push(new Integer(13));
st.push(new Integer(14));
st.push(new Integer(15));
System.out.println(st);
System.out.println("peek element:"+st.peek());
int pos = st.search(14);
System.out.println("Ele 14 at pos:"+pos);
    }
}
```



**Note:**

=>**search()** method searches the ele from top of the Stack<E> and display the position of an element.

---

**Note:**

=>In realtime Stack<E> is used part of algorithmic design.

---

**Summary:**

**ArrayList<E>**

=>Elements in Sequence

=>NonSynchronized class

=>Used for UserLogin in realtime

**LinkedList<E>**

=>Elements in NonSequence

=>NonSynchronized class

=>Used for AdminLogin in realtime

**Vector<E>**

=>Elements in Sequence

=>Synchronized class

=>Used in Connection Pooling Concept.

**Stack<E>**

=>Elements in Sequence

=>Synchronized class

=>In Algorithm design.

---

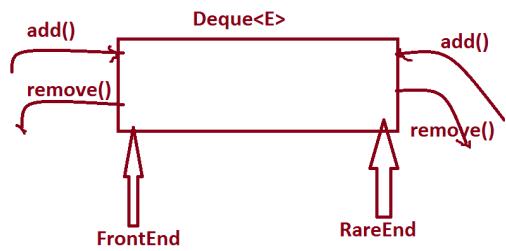
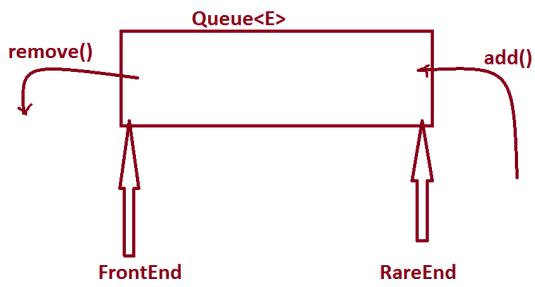
**3.Queue<E>:**

=>Queue<E> organizes elements based on the algorithm First-In-First-Out

or Last-In-Last-Out.

**define Deque<E>?**

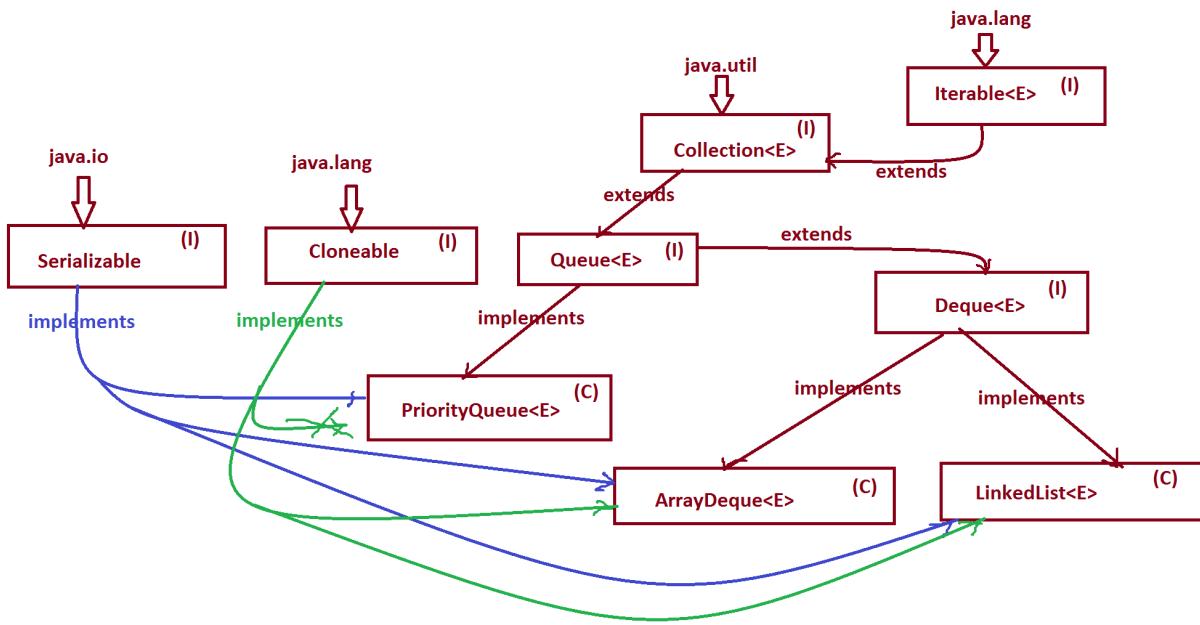
=>Deque<E> means Double-Ended-Queue and which organizes elements on both ends.



=>The following are the implementation classes of **Queue<E>** and **Deque<E>**:

- (a) **PriorityQueue<E>**
- (b) **ArrayDeque<E>**
- (c) **LinkedList<E>**

**Hierarchy of Implementation classes:**



## PriorityQueue<E>

=>PriorityQueue<E> organizes elements based on element's priority.

## ArrayDeque<E>

=>which is Double-ended-queue and which organizes elements in Sequence.

## LinkedList<E>

=>which is also Double-ended-queue and which organizes elements in

NonSequence.

## Exp program:

```

package maccess;
import java.util.*;
public class DQueue {
    public static void main(String[] args) {
        PriorityQueue<Integer> pq = new
        PriorityQueue<Integer>();
        pq.add(12);
    }
}

```

```
    pq.add(13);
    pq.add(14);
    pq.add(15);
    System.out.println(pq);
    ArrayDeque<Integer> ad = new ArrayDeque<Integer>();
    ad.add(12);
    ad.add(13);
    ad.add(14);
    ad.add(15);
    System.out.println(ad);
    ad.addFirst(11);
    ad.addLast(16);
    System.out.println(ad);
    LinkedList<Integer> ll = new LinkedList<Integer>();
    ll.add(12);
    ll.add(13);
    ll.add(14);
    ll.add(15);
    System.out.println(ll);
    ll.addFirst(11);
    ll.addLast(16);
    System.out.println(ll);
}

}
```

**Note:**

=>In realtime Queue<E> and Deque<E> are used in algorithm design.

=====

=====

Dt : 5/3/2021

**Limitation of Collection<E>:**

=>In the process of holding DB table data, Collection<E> cannot differentiate PrimaryKey and NonPrimaryKey values.

**Note:**

=>This DisAdvantage can be Overcomed using Map<K,V>.

\*imp

**define Map<K,V>?**

=>Map<K,V> is an interface from java.util package and which organizes elements in the form of Key-value pairs

**K - Key**

**V - Values**

=>The following are some important methods from Map<K,V>:

**public abstract int size();**

**public abstract boolean isEmpty();**

**public abstract boolean containsKey(java.lang.Object);**

**public abstract boolean containsValue(java.lang.Object);**

**public abstract V get(java.lang.Object);**

**public abstract V put(K,V);**

**public abstract V remove(java.lang.Object);**

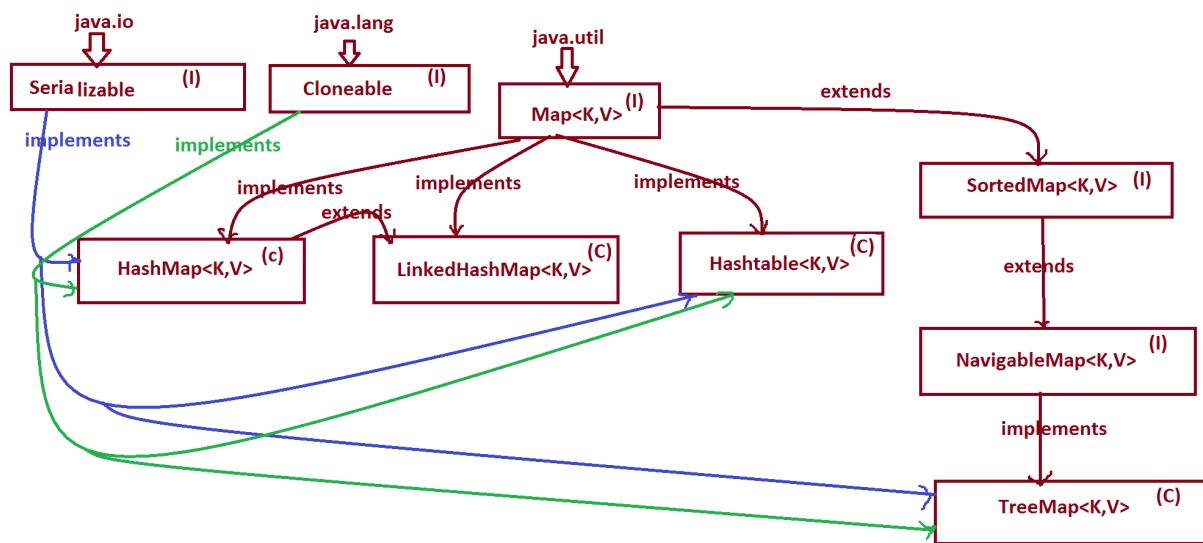
**public abstract java.util.Set<K> keySet();**

```
public abstract java.util.Collection<V> values();
```

=>The following are the implementation classes of Map<K,V>:

- (a)HashMap<K,V>
- (b)LinkedHashMap<K,V>
- (c)TreeMap<K,V>
- (d)Hashtable<K,V>

Hierarchy of implementation classes:



Note:

=>Hashtable<K,V> is synchronized class and remaining three classes are

NonSynchronized classes.

=>HashMap<K,V> and Hashtable<K,V> organizes elements without any order.

=>LinkedHashMap<K,V> organizes elements in insertion order.

=>TreeSet<K,V> organizes elements in ascending order based on key data.

Exp program:

```
import java.util.*;

class Product //SubClass

{
    String pName;
    float pPrice;
    int pQty;

    Product(String pName,float pPrice,int pQty){
        this.pName=pName;
        this.pPrice=pPrice;
        this.pQty=pQty;
    }

    @Override
    public String toString(){
        return pName+"\t"+pPrice+"\t"+pQty;
    }
}

class DMap1 //MainClass

{
    public static void main(String[] args)
    {
        try(Scanner s = new Scanner(System.in)){


            HashMap<String,Product> hm = new
            HashMap<String,Product>();
        }
    }
}
```

```
System.out.println("Enter the number of products:");
int n = Integer.parseInt(s.nextLine());
System.out.println("Enter "+n+" Products ");
for(int i=1;i<=n;i++){
    System.out.println("Enter the pCode:");
    String pCode = s.nextLine();
    System.out.println("Enter the pName:");
    String pName = s.nextLine();
    System.out.println("Enter the pPrice:");
    float pPrice = Float.parseFloat(s.nextLine());
    System.out.println("Enter the pQty:");
    int pQty = Integer.parseInt(s.nextLine());
    hm.put(new String(pCode),new
Product(pName,pPrice,pQty));
}
//end of loop
System.out.println("====Display from Map<k,V>
object====");
hm.forEach((k,v)->
{
    System.out.println(k+"\t"+v);
});
System.out.println("====Only key data====");
Set<String> st = hm.keySet();
st.forEach((p)->
{

```

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

    });

    System.out.println("====Only Values data====");

    Collection<Product> c = hm.values();

    c.forEach((q)->

    {

        System.out.println(q);

    });

    } //end of try

}

}
```

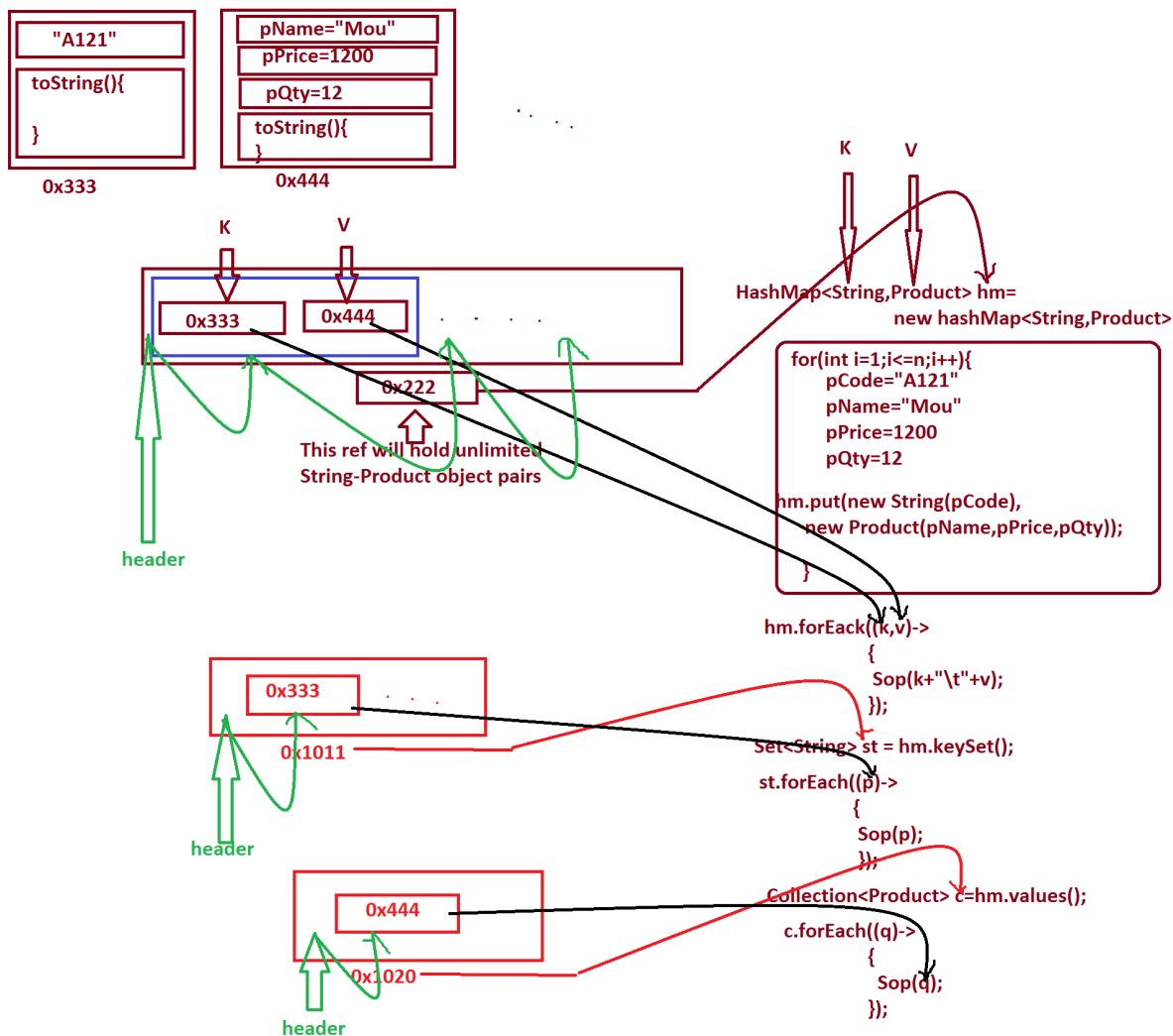
Dt : 6/3/2021

Execution flow of above program:

ClassFiles:

Product.class

DMap1.class



\*imp define forEach() method?

=>forEach() method introduced by Java8 version and which is used to retrieve the elements from Collection<E> objects and Map<K,V> objects.

**Method Signature of forEach() method on Map<K,V>:**

```
public void forEach(java.util.function.BiConsumer<? super K,>? super V>);
```

**Method Signature of forEach() method on Collection<E>:**

```
public void forEach(java.util.function.Consumer<? super E>);
```

**define BiConsumer<T,U>?**

=>**BiConsumer<T,U>** is a functional interface introduced by Java8 version  
and which provides the abstract method Signature to hold LambdaExpression  
passed as parameter to the forEach() method on Map<K,V> object.

**Structure of BiConsumer<T,U>:**

```
public interface java.util.function.BiConsumer<T,U>
{
    public abstract void accept(T,U);
}
```

**define Consumer<T>?**

=>**Consumer<T>** is a functional interface introduced by Java8 version  
and which provides the abstract method Signature to hold LambdaExpression  
passed as parameter to the forEach() method on Collection<E> object.

### **Structure of Consumer<T>:**

```
public interface java.util.function.Consumer<T>
{
    public abstract void accept(T);
}
```

---

#### **define keySet() method?**

**=>This keySet() method will generate Set<E> object and which is loaded with only key data.**

#### **define values() method?**

**=>This values() method will generate Collection<E> object and which is loaded with only values data.**

---

### **List of Cursor Statements(Iterative Statements):**

#### **1.Iterator<E>**

**=>Iterator<E> is used on Collection<E> objects.**

#### **2.ListIterator<E>**

**=>ListIterator<E> is used only on List<E> objects.**

#### **3.Enumeration<E>**

**=>Enumeration<E> is used only on Vector<E>**

#### **4.Spliterator<T>**

=>Spliterator<T> is used on Arrays,Collection<E> and Stream<T>

## 5.forEach()

=>forEach() method is used on Collection<E> and Map<K,V>

---

faq:

define 'Collections'?

=>'Collections' is a utility class from java.util package and which provides some methods to perform operations on Collection<E> objects.

=>The following are two important methods from Collections class:

```
public static <T> void sort(java.util.List<T>);
```

```
public static <T> int binarySearch(java.util.List<T>,..);
```

Exp program:

```
import java.util.*;

class DCollections //MainClass

{
    public static void main(String[] args)
    {
        ArrayList<Integer> al = new ArrayList<Integer>();
        al.add(new Integer(12));
        al.add(new Integer(10));
        al.add(new Integer(11));
```

```
al.add(new Integer(13));

System.out.println("==>Before Sorting==>");

System.out.println(al);

Collections.sort(al); //Sorting process

System.out.println("==>After Sorting==>");

System.out.println(al);

int p = Collections.binarySearch(al, 11);

System.out.println("Ele at index value: " + p);

}

}
```

faq:

**define 'Arrays'?**

=>'Arrays' is a utility class from java.util package and which provides some methods to perform operations on Array objects.

=>The following are two important methods from Arrays class:

**public static void sort(arr[]);**

**public static int binarySearch(arr[]);**

**Exp program:**

```
import java.util.*;

class DArrays //MainClass

{
```

```
public static void main(String[] args)
{
    Integer a[] = {12,10,11,13};

    System.out.println("====Before Sorting====");

    Spliterator<Integer> sp1 = Arraysspliterator(a);

    sp1.forEachRemaining((k)->
    {
        System.out.print(k+" ");
    });

    Arrays.sort(a);//Sorting process

    System.out.println("\n====After Sorting====");

    Spliterator<Integer> sp2 = Arraysspliterator(a);

    sp2.forEachRemaining((z)->
    {
        System.out.print(z+" ");
    });

    int p = Arrays.binarySearch(a,12);

    System.out.println("\nEle 12 available at index value:"+p);
}

-----
```

Dt : 8/3/2021

=>NonPrimitive datatypes or referential datatypes:

(a)Class

(b)Interface

(c)Array/Collection<E>/Map<K,V>

(d)Enum

(d)Enum<E>:

=>Enum<E> is a abstract class from java.lang package and which is used hold elements,variables,constructors and methods.

=>we use 'enum' keyword to declare enums.

syntax:

```
enum Enum_name
```

```
{
```

```
//elements
```

```
//variables
```

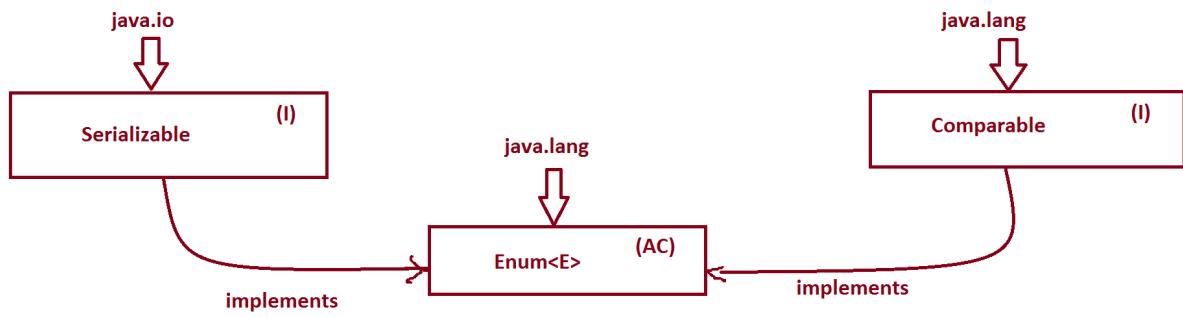
```
//constructors
```

```
//methods
```

```
}
```

=>we use 'values()' method to retrive data from Enum<E> objects.

Hierarchy of Enum<E>:



**Exp program:**

```

enum Cars

{
    figo(900),dezire(700),alto(500);

    public int price;

    private Cars(int price)

    {
        this.price=price;

    }

    public int getPrice()

    {
        return price;
    }

}

class DemoEnum //MainClass

{
    public static void main(String[] args)

    {
        System.out.println("====Display from Enum<E>====");
    }
}

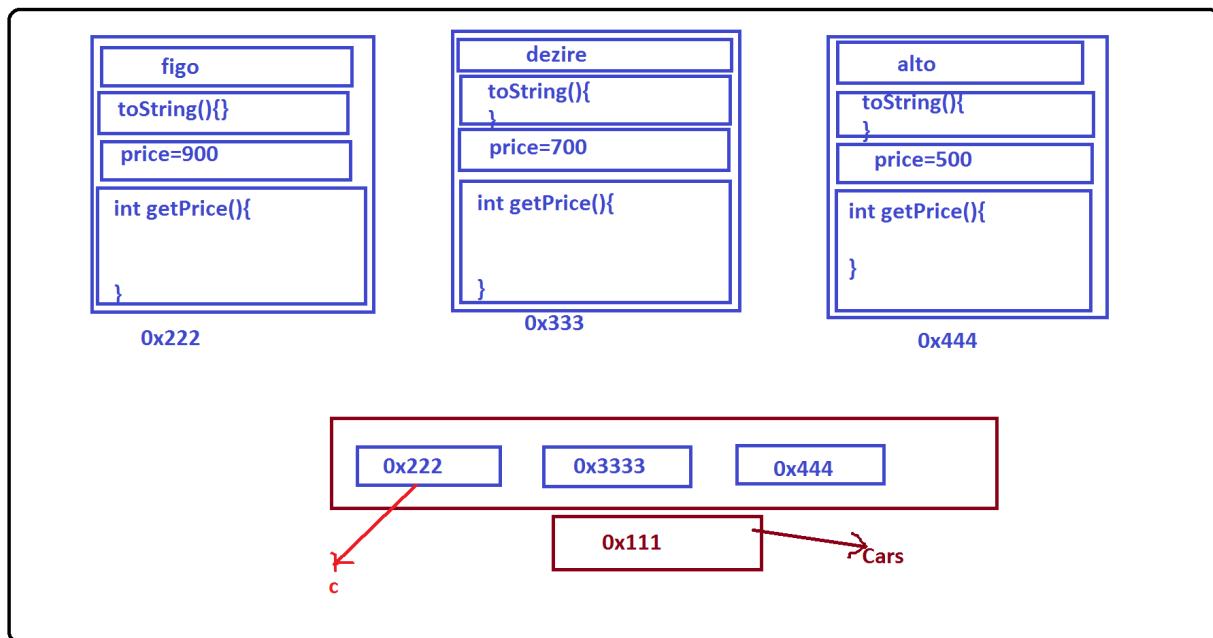
```

```

        for(Cars c : Cars.values())
        {
            System.out.println(c+" Costs "+c.getPrice()+" Thousand
Dollars");
        }//end of loop
    }
}

```

**Diagram of Enum<E> Object:**



#### Note:

=>The constructors which are declared part of Enum<E> must be private

Constructors because the objects are created internally.

=>In realtime Enum<E> is less used when compared to Class and Interfaces.

---



---

#### Summary:

=>The following are the list of objects generated from CoreJava:

**1.User defined class objects**

**2.WrapperClass objects**

**3.String Objects**

**4.Array objects**

**5.Collection<E> objects**

**6.Map<K,V> Objects**

**7.Enum<E> objects**

**1.User defined class objects**

**2.WrapperClass objects**

**(a)Byte Object**

**(b)Short Object**

**(c)Integer Object**

**(d)Long Object**

**(e)Float Object**

**(f)Double Object**

**(g)Character Object**

**(h)Boolean Object**

**3.String Objects**

**(a)String Class Object**

**(b)StringBuffer class Object**

**(c)StringBuilder Object**

**4.Array objects**

**(a)User Defined class objects Array**

**(b)WrapperClass objects Array**

**(c)String Objects Array**

**(d)Object Array**

## **5.Collection<E> objects**

**(i)Set<E>**

**(a)HashSet<E> object**

**(b)LinkedHashSet<E> Object**

**(c)TreeSet<E> object**

**(ii)List<E>**

**(a)ArrayList<E> object**

**(b)LinkedList<E> object**

**(c)Vector<E> object**

**(d)Stack<E> Object**

**(iii)Queue**

**(a)PriorityQueue<E> object**

**(iv)Deque<E>**

**(b)ArrayDeque<E> object**

## **6.Map<K,V> Objects**

**(a)HashMap<K,V> object**

**(b)LinkedHashMap<K,V> object**

**(c)TreeMap<K,V> object**

**(d)Hashtable<K,V> object**

## **7.Enum<E> objects**

**=====>Total Objects : 30 objects**

## AdvJava : 40 Objects

JDBC : 10

## **Servlet : 10**

JSP : 9

El-JSTL : 11

40

fac:

## define Generic Programming Components?

=>The programming components which are ready to accept any type at runtime are known as Generic Programming Components.

=>The following are the list of Generic Programming components:

## (a) Generic Types

## (b) Generic methods

### (c) Generic Classes

#### (d) Geric Interfaces

### (a) Generic Types:

=>The Types which are ready to accept any type of data are known as Generic Types.

**Exp:**

**T - Type**

**E - Element**

**V - Value**

**K - Key**

**(b)Generic methods:**

=>The methods which are ready to accept any type of data as parameters.

**syntax:**

```
<T>return_type method_name(T)  
{  
    //method_body  
}
```

**(c)Generic Classes:**

=>The Class Object reference can hold any type of Object references is known as Generic Class.

**syntax:**

```
class Class_name<T>  
{  
    //Class_body  
}
```

#### **(d) Generic Interfaces:**

**syntax:**

```
interface Interface_name<T>
{
    //Interface_body
}
```

---

**Note:**

=>The Object Storages which are generated part of JVM are temporary Storages because the Objects will be destroyed when the JVM ShutDowns.

=>whenever we want to have permanent storage for WebApplication then we use the following:

**(i)File Storage**

**(ii)DB Stoarge**

---

**\*imp**

**IO Streams and Files:**

**define Stream?**

=>The continuous flow of data is known as Stream.

**define InputStream?**

=>The continuous flow of data into JavaApplication is known as Input Stream.

**define OutputStream?**

=>The continuous flow of data out of Java Application is known as Output Stream

---

=>The Streams in Java are categorized into two types:

- (a) Binary Stream
- (b) Character Stream

(a) Binary Stream:

=>The Continuous flow of 8 bit data is known as Binary Stream or Byte Stream.

Note:

=>Binary Stream supports all the multimedia data formats like Text, Audio, Video, Image and Animation.

(b) Character Stream:

=>The Continuous flow of 16 bit data is known as Character Stream or Text Stream.

Note:

Character Stream best supports for Text data and which is not preferable for Audio, Video, Image and Animation.

---

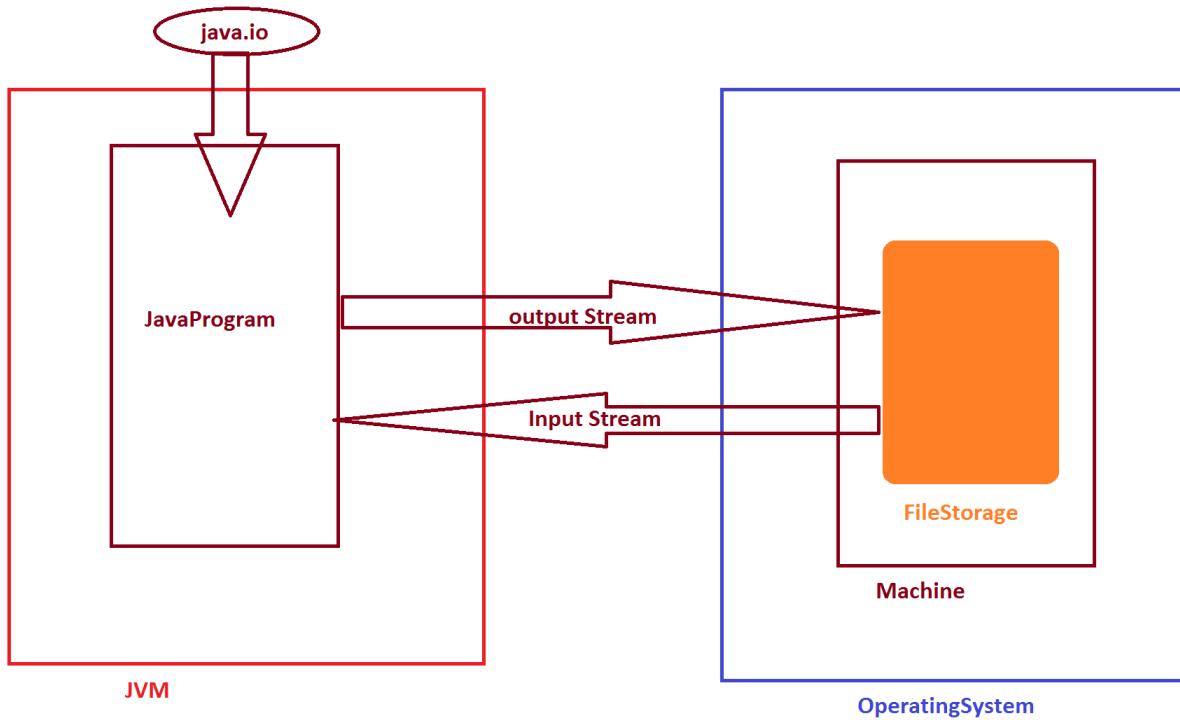
Dt : 10/3/2021

### Define File Storage?

=>The Smallest permanent storage of Computer System,which is organized by Operating System is known as File Storage.

### Note:

=>In the process of establishing communication b/w JavaProgram and File Storage,the JavaProgram must be constructed using Classes and Interfaces available from 'java.io' package



=>The following are the classes related to Binary Stream:

#### (i) DataInputStream:

=>DataInputStream class is from java.io package and which is used to

**read binary Stream into JavaProgram.**

**syntax:**

```
DataInputStream dis = new DataInputStream(Source);
```

**(ii)DataOutputStream:**

=>DataOutputStream class is from java.io package and which is used to send the data out of JavaProgram.

**syntax:**

```
DataOutputStream dos = new DataOutputStream(Destination);
```

**(iii)FileInputStream:**

=>FileInputStream class is from java.io package and which is used to find the file and open the file to read Binary Stream.

**syntax:**

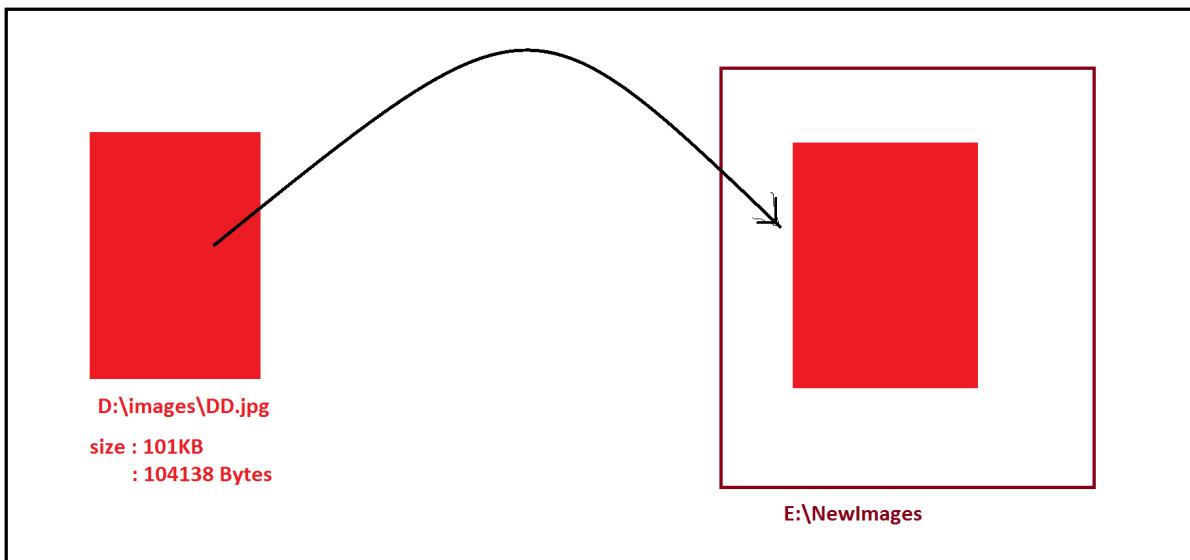
```
FileInputStream fis = new FileInputStream("fPath&fName");
```

**(iv)FileOutputStream:**

=>FileOutputStream class is from java.io package and which is used to create the file and opens the file to write binary Stream.

**syntax:**

```
FileOutputStream fos = new FileOutputStream("fPath&fName");
```



### Exp program1:

Wap to to copy the file from one location to another location?

```

package maccess;
import java.io.*;
public class DFile1 {
    @SuppressWarnings("deprecation")
    public static void main(String[] args) {
try{
DataInputStream dis = new DataInputStream(System.in);
System.out.println("Enter the fpath&fName(Source)");
File f1 = new File(dis.readLine());
    if(f1.exists()){
FileInputStream fis = new FileInputStream(f1);
System.out.println("Enter the
fPath&fName(destination)");
File f2 = new File(dis.readLine());
FileOutputStream fos = new FileOutputStream(f2);
int ch;
while((ch=fis.read())!=-1){
    }
}
}
}

```

```
    fos.write(ch);
}//end of loop
fos.close();
fis.close();
System.out.println("File Copied Successfully...");
}else{
    System.out.println("Invalid fPath or
 fName... ");
}
}catch(Exception e){e.printStackTrace();}
}
```

O/P:

**Enter the fpath&fName(Source)**

D:\images\DD.jpg

**Enter the fPath&fName(destination)**

E:\NewImages\TT.jpg

**File Copied Successfully...**

**define 'File' class?**

=>'File' class is from java.io package and which is used to find the properties of file like filePath,fileName,fileLength,...

**syntax:**

**File f = new File("fPath&fName");**

=>The following are the classes related to Character Stream:

### **1.BufferedReader:**

=>BufferedReader class is from java.io package and which is used to read the character data to the program.

**syntax:**

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

### **2.FileWriter:**

=>FileWriter class is from java.io package and which is used to create file and opens the file to write the data.

**Syntax:**

```
FileWriter fw = new FileWriter("fPath&fName");
```

### **2.FileReader:**

=>FileReader class is from java.io package and which is used to open the file and read the data.

**Syntax:**

```
FileReader fw = new FileReader("fPath&fName");
```

Exp program:

```
package maccess;
import java.io.*;
public class DFile2 {
    public static void main(String[] args) {
try{
    BufferedReader br =
        new BufferedReader(new
InputStreamReader(System.in));
    FileWriter fw = new
FileWriter("E:\\NewImages\\Text.txt");
    char ch1;
    System.out.println("Enter the data:@ at end");
    while((ch1=(char)br.read())!='@'){
        fw.write(ch1);
    }//end of loop
    fw.close();
    System.out.println("====Display from File====");
    FileReader fr = new
FileReader("E:\\NewImages\\Text.txt");
    int ch2;
    while((ch2=fr.read())!=-1){
        System.out.print((char)ch2);
    }
    fr.close();
}catch(Exception e){e.printStackTrace();}
    }
}
```

**\*imp**

**Object State on to File Storage:**

=>In the process of storing Object State onto File Storage, the Object State must be available in the form of Binary Stream.

**define Serialization process?**

=>The process of Converting the Object State into Binary Stream is known as **Serialization process**.

**Note:**

=>we use **writeObject()** method from '**java.io.ObjectOutputStream**' class to perform **Serialization process**.

**Method Signature of writeObject() method:**

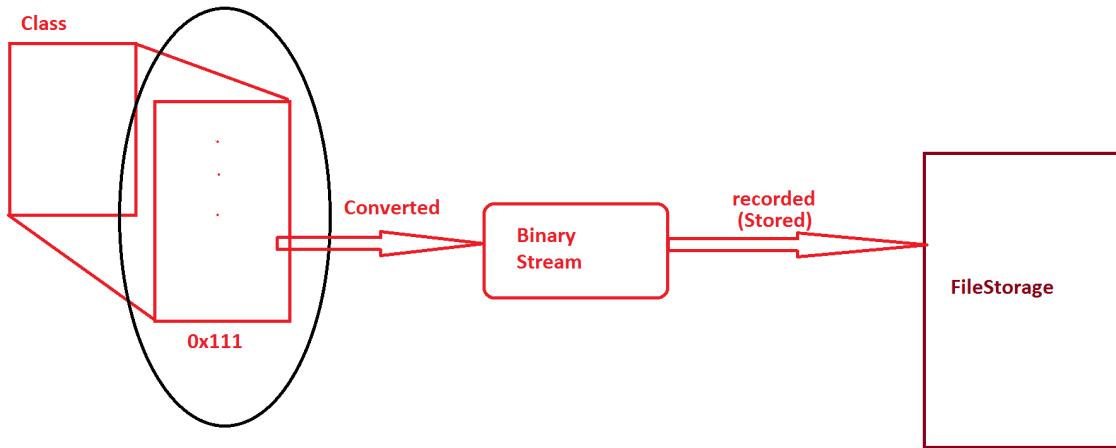
**public final void writeObject(java.lang.Object) throws java.io.IOException;**

**syntax:**

```
ObjectOutputStream oos = new ObjectOutputStream(fos);  
oos.writeObject(obj);
```

**Note:**

=>To perform **Serialization and DeSerialization processes** the class must be implemented from '**java.io.Serializable**' interface.



**Exp program:**

**(Demonstrating Serialization process)**

```

package maccess;

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

@SuppressWarnings("serial")
public class User implements Serializable
{
    public String uName,pWord;
    public Date d;
    public User(String uName,String pWord,Date d){
        this.uName=uName;
        this.pWord=pWord;
        this.d=d;
    }
    @Override
  
```

```
public String toString(){
    return uName+"\n"+pWord+"\n"+d;
}

}
```

```
package maccess;

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

public class DFile3 {

    public static void main(String[] args) {
        try{
            User u1 = new User("nit.v","mzu672",new Date());
            FileOutputStream fos =
                new FileOutputStream("E:\\NewImages\\Obj.txt");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(u1);//Serialization
            System.out.println("Object Stored Successfully...");
            oos.close();
        }catch(Exception e){e.printStackTrace();}
    }
}
```

**Define DeSerialization process?**

=>The process of converting binary Stream into Object State is known as

**DeSerialization process.**

**Note:**

=>we use **readObject()** method from 'java.io.ObjectInputStream' class to perform DeSerialization process.

**Method Signature of readObject() method:**

```
public final java.lang.Object readObject() throws java.io.IOException,  
        java.lang.ClassNotFoundException;
```

**syntax:**

```
ObjectInputStream ois = new ObjectInputStream(fis);  
Object o = ois.readObject();
```

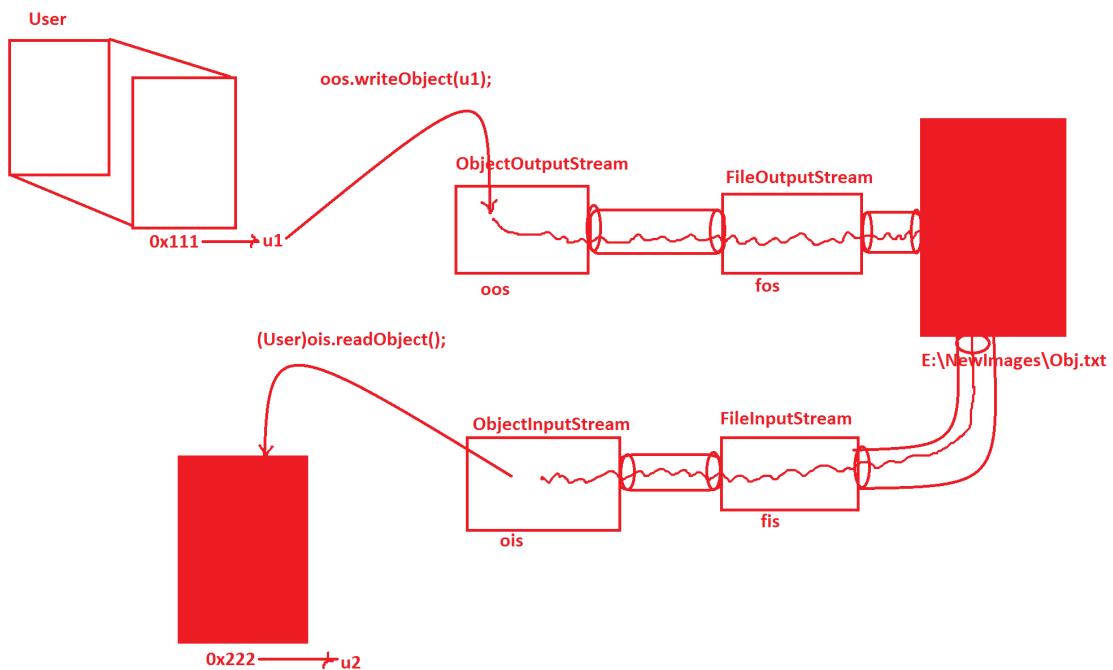
**Exp program:**

```
package maccess;  
import java.io.*;  
public class DFile4 {  
    public static void main(String[] args) {  
        try{  
            FileInputStream fis =  
                new  
FileInputStream("E:\\NewImages\\Obj.txt");  
            ObjectInputStream ois = new  
ObjectInputStream(fis);  
            User u2 = (User)ois.readObject();  
            System.out.println("====User  
details====");  
            System.out.println(u2);  
        }  
    }  
}
```

```

        ois.close();
    }catch(Exception e){e.printStackTrace();}
}
}

```



Dt : 12/3/2021

\*imp

**MultiThreading in Java:**

**define Application?**

=>The set-of-programs collected together to perform defined action is known as

**Application.**

**define program?**

=>Program is a set-of-Instructions.

**define process?**

=>According to Operating System, the program under execution is known as Process.

=>According to Programming Language, the Application under execution is known as process.

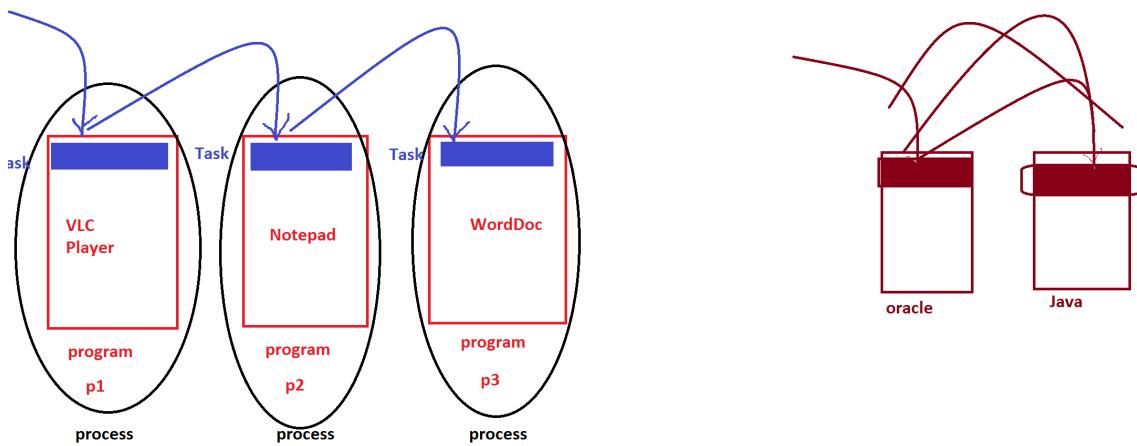
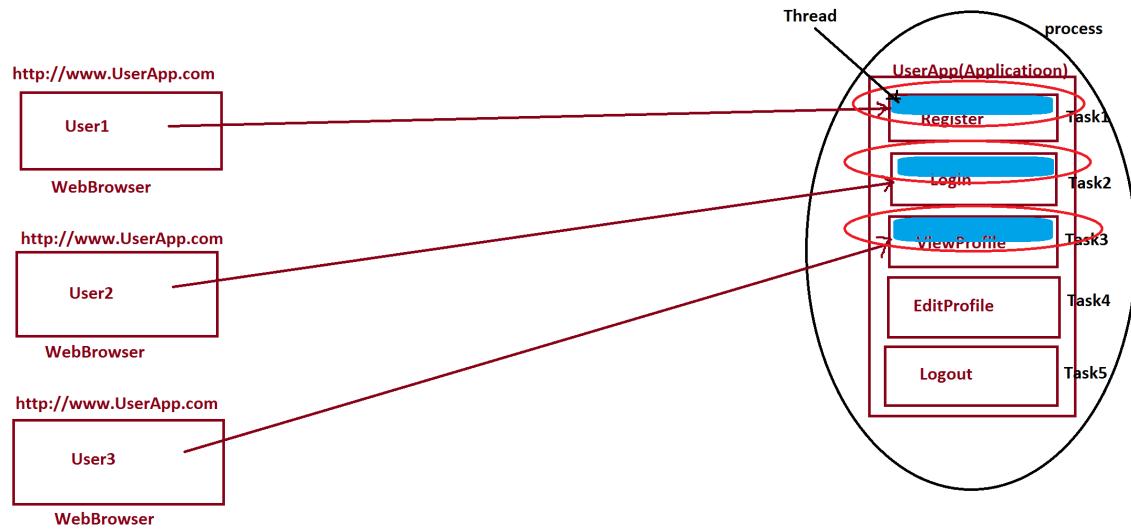
**define Task?**

=>The part of process which is executed is known as Task

**define MultiTasking?**

=>Executing Multiple Tasks Simultaneously is known as MultiTasking.

(Simultaneously means at-a-time but not parallel)



=>MultiTasking process is categorized into two types:

- 1.Process based MultiTasking
- 2.Thread based MultiTasking

### 1.Process based MultiTasking:

=>Executing multiple tasks from Multiple processes is known as Process based MultiTasking.

**Note:**

=>This process based MultiTasking is used in Operating System.

**2.Thread based MultiTasking:**

=>Executing Multiple Tasks from same process is known as Thread based MultiTasking.

**Note:**

=>This thread based MultiTasking is used in Programming Language.

Dt : 13/2/2021

**Note:**

=>Each program of application is considered as task

**define Thread?**

=>The part of task is known as thread.

**define MultiThreading?**

=>Executing multiple threads simultaneously is known as MultiThreading.

**Creating and Executing threads:**

**step1 :** The user defined class must be implemented from 'java.lang.Runnable' interface.

### **Structure of Runnable Interface:**

```
public interface java.lang.Runnable
{
    publuc abstract void run();
}
```

**step2 : The user defined class must be declared with run() method and which is overriding**

**method of 'Runnable'**

**Note:**

**=>The required logic is written within the run() method**

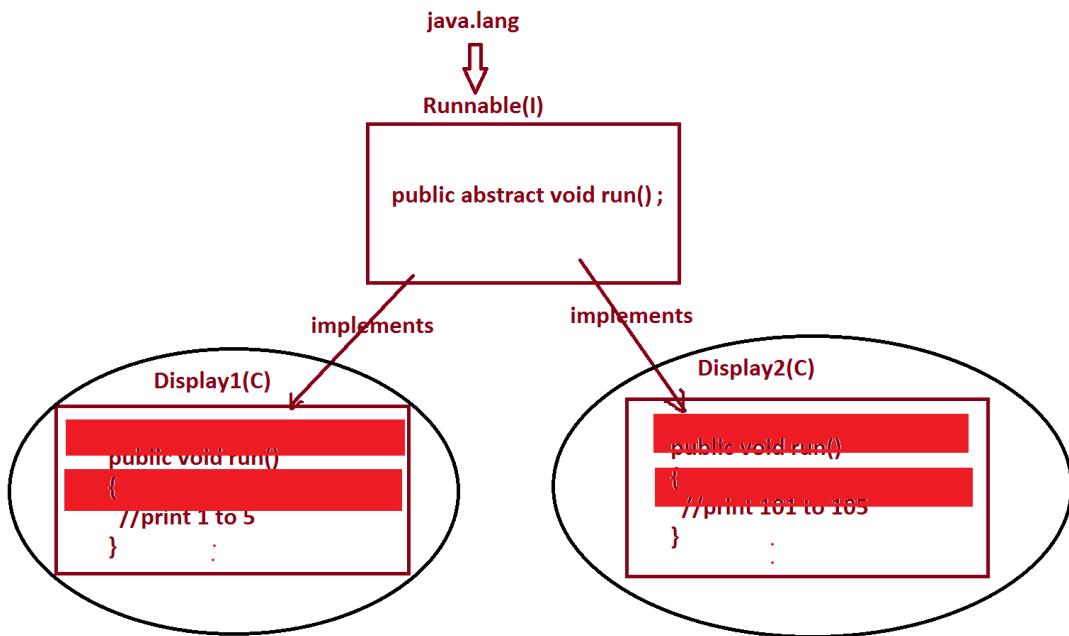
**step3 : Create object for User defined class**

**step4 : Create object for 'java.lang.Thread' class and while object creation pass the reference of**

**user defined class object as parameter.**

**step5 : call start() method to execute run() method**

**Exp program:**



### Display1.java

```

package maccess;
public class Display1 implements Runnable//step1
{
    @Override
    public void run()//step2
    {
        for(int i=1;i<=5;i++)
        {
            System.out.println("Task1 : "+i);
        }
    }
}
    
```

### Display2.java

```
package maccess;
public class Display2 implements Runnable//step1
{
    @Override
    public void run()//step2
    {
        for(int i=101;i<=105;i++)
        {
            System.out.println("Task2 : "+i);
        }
    }
}
```

#### DThread1.java(MainClass)

```
package maccess;
public class DThread1 {
    public static void main(String[] args) {
Display1 d1 = new Display1(); //step3
Display2 d2 = new Display2(); //step3

Thread t1 = new Thread(d1); //step4
Thread t2 = new Thread(d2); //step4

t1.start(); //step5
t2.start(); //step5
    }

}
```

#### Output:

Task1 : 1

Task2 : 101

**Task1 : 2**

**Task2 : 102**

**Task1 : 3**

**Task2 : 103**

**Task1 : 4**

**Task2 : 104**

**Task1 : 5**

**Task2 : 105**

**Execution flow of above program:**

**ClassFiles:**

**Display1.class**

**Display2.class**

**DThread1.class(MainClass)**

