CORE JAVA Notes

JAVA doesn't have any expansion because it is not acronym

Creator of JAVA – JAMES GOSLING

He worked for SUN micro systems

1996 – First version JDK1.0

JDK – JAVA development kit – A kit that contain tools to develop the programme in JAVA

Java 1.2 – J2

Latest version is 1.8 – 2014

What is JAVA? Java is high level object oriented programming language introduced by SUN micro systems in 1996

IMP: JAVA IS NOT ACRONYM (which means not expandable)

What is logo of JAVA? Hot coffee in cup and soccer

Mascot – Is a Character or human or animal which is supposed to bring the luck to team or company or group.

DUKE

Server – serves for multiple systems

Different types of OS –

1. Windows
2. Linux
3. Solaris
4. MAC

In 2010, Oracle Corporation baught SUN micro systems

Now officially Oracle is the responsible for JAVA updates or changes

J2SE - JAVA 2 Standard Edition

J2EE – JAVA 2 Enterprise Edition

J2ME - JAVA 2 Micro Edition

Features of JAVA –

1. Compiled and Interpreted
2. Plantform Independent
3. Object Oriented
4. High Level
5. Robust
6. Multithreaded and Interactive
7. Simple, Safe, Secure and Familier
8. Dynamic and Extensible code
9. Distributed

CMD prompt – Type Java to check whether the JAVA is installed or not

Or type javac

CMD to check java versvion – Java -version

If already installed and mentioned the path, we will get USAGE

Else – we will get the message like Java is not reconglised as internla or extrenal command

More than 5 Billion devices are running on JAVA

To clear the usage – CLS command

My computer – properties – Advanced system settings – Environemntal Variables – System Variables – Paths – paste the path that copied for bin folder (i.e. C:\Program Files\Java\jdk1.8.0_102\bin) by separating with semicolon (;)

- jdk-8u102-windows-x64.exe – in this one 8 means 1.8 and u means – update and 102 means – 102 releases were happened

Down load JAVA from

https://www.oracle.com/index.html - Downloads - Popular Downloads – Java SE – JDK Download- Windows x64

Install latest version

When will you get the "Java or Javac is not recognised as internal or external command'?

- If java is not installed at all
- If java is installed and not defined the path

3

To write java program we use:

1. Code or Text editors (Notepad, Notepad++, Edit plus, Text edit)
2. IDE's (Eclipse, IntelliJIdea, NetBeans)

IDE: Integrated Development Environment

1. What is Compiler?
2. What is Interpreter?
3. What is HLL, ALL and LLL?

Compiler: Is a program that transforms source code written in a program language into BINARY format or executable program

Ex: C, C++, C# and Java

Interpreter: Executes the source code directly or translates the code one line at a time. These programs can run on computer having interpreter. Interpreter do generate the binary code, but the code is never compiled instead interpreted each & every time the program executes.

HLL (High Level language): Programming language which are more understand by human and it is more natural language element, easier, making the processor of developing program simple and understandable

Ex: C, C++, C# and Java

LLL (Low Level language): These languages which are close to machine understandable languages

Ex: Binary language

Machine Language: This is the only language a microprocessor can process without any previous transformation

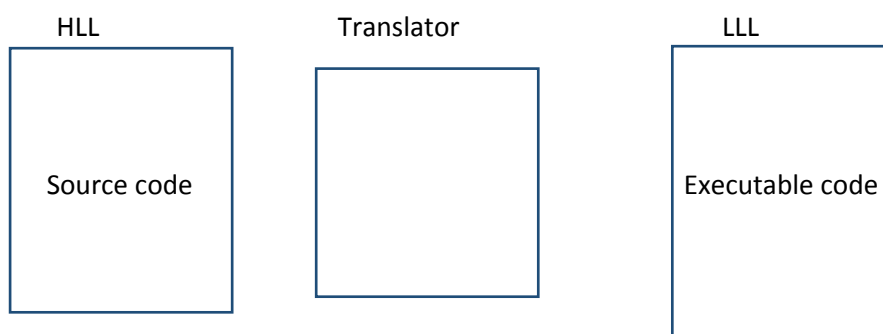**ALL (Assembly Level language):** It is also a low level language which uses assembler

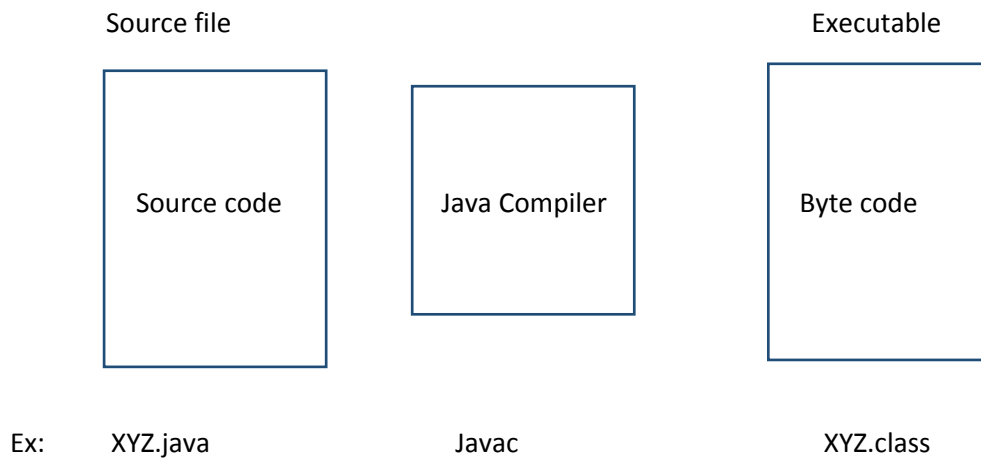Below are some commands and it is difficult to understand
ALL
Mov X
Add AOX

In Java:

HLL	Translator	LLL
Source code		Executable code

Source file		Executable
Source code	Java Compiler	Byte code

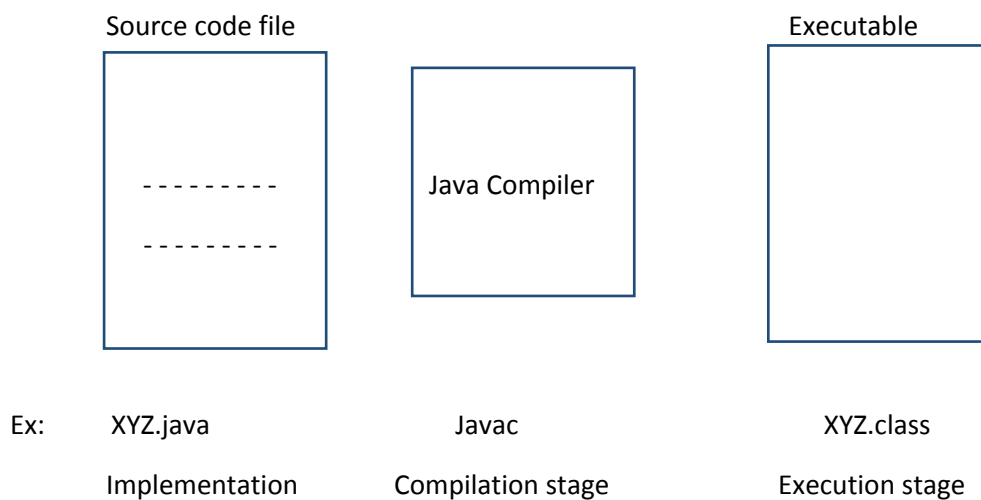Ex: XYZ.java   Javac   XYZ.class

The Program written in java has to be send in a file with extension '.java'. This file is known as source file.

Source file (.java file) is fed to java compiler. The java compiler converts source code into '.class' file. This is an executable file and stored in byte code format.

The class files (.class) are automatically generated by the java compiler. The JVM (Java Virtual Machine) understands only bytecode hence where ever JVM is available class files can be executed.

Source code file		Executable
- - - - - - - - - - - - - - - - - -	Java Compiler	

Ex: XYZ.java   Javac   XYZ.class

         Implementation   Compilation stage   Execution stage

Development Phase:

==Compilation: (Command – Javac)==

Ex: Javac Demo.java (Demo is the file name)

The source code written inside java file are converted into byte code (class file) by using a command Javac. This Javac command is available in the bin folder where Java is installed

The Syntax is as below

```
D:\Java programs>javac Demo.java
```

Javac is the command

Demo.java is the source file

==Execution: Command – Java==

Ex: java Demo

After compilation, the class files are generated in the default location where source code is available. The class files can be executed by using a command 'java'. This command is also available in bin folder

The Syntax is as below

```
D:\Java programs>java Demo.java
```

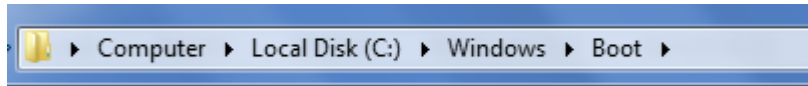Note: While running the class file '.class' extension should not be specified

Program ex:

```
class Demo
{
        Public static void main(String[] args)
{
        System.out.println("Welcome to Demo class");
}
}
```
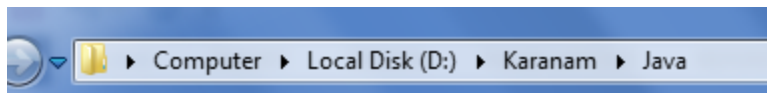
7ᵗʰ Sep 2016

Create a folder to save java programs under any drive other than 'C'

Examples to go to different paths and folders using commands in command prompt

In C: create the following folder structure

> Computer ▸ Local Disk (C:) ▸ Windows ▸ Boot ▸

In D: create the following folder structure

> Computer ▸ Local Disk (D:) ▸ Karanam ▸ Java

Dos commands:

1. Open command prompt:
    Start – run – type cmd and click enter

2. To change the drive letter:
    Type the drive letter and put colon and click enter
    Ans – C:\E: enter (C:\ is the default path and we are changing this to E drive)

3. Go to child directory of folder
    From: C:\sample
    To: C:\sample\job1 – use cd child folder name
    Ans – C:\sample cd job1 and click enter

4. Go to root or drive from a child or sun child folder
    From: C:\sample\job1
    To: C: - use cd \ and click enter
    Ans: C:\sample\job1 cd \ and click enter

5. Go to parent directory or folder
    From: C:\sample\job1
    To: C:\sample – use cd .. and click enter
    Ans: C:\sample\job1 cd .. and click enter

6. Clear the screen
    Ans: cls and click enter

7. Go to child's child directory
    From: C:\
    To: C:\sample\job1 – enter cd desired path name and click enter

    Ans: C:\ cd sample\job1 and click enter

8. Go to specific folder from another folder in different path under same drive
   From: C:\sampl\job1
   To: C:\Bombay\cricket - enter cd desired path name and click enter
   Ans: C:\sample\job1 cd C:\Bombay\cricket and click enter

9. Go to specific folder from another folder in different path under different drive
   From: C:\sampl\job1
   To: D:\Bombay\cricket - enter cd /d desired path name and click enter
   Ans: C:\sample\job1 cd \d D:\Bombay\cricket and click enter

10. Display the content of current directory
    C:\ dir and click enter

Any java file will have

Class ClassName

{

}

Ex:

Class Demo

{

}

Note: This Demo.java can be compiled but cannot be executed because it doesn't have main method defined

To execute any java file will have

Class Classname

{

      public static void main(String[] args)

{

}

}

Imp Note: to execute the class file there should be main method defined as below

public static void main(String[] args)

{

}

To print a message on the screen

System.out.println("Welcome to Java class");

S – Should be in upper case or caps

Program 1:

Class Demo1

{

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

{

          System.out.println("Welcome to Java class");

}

}

Note: While taking notes, please use PSVM for 'public.static.void.main(String[] args)' in short form

Also pleas use SOP for 'System.out.println("Welcome to Java class");' in short form

To compile and execute:

1. Write java program in notepad
2. Save the program with extension '.java'

   Ex:
   If class Mango, then save it as Mango.java
3. Save it in a folder with your name (Preferable in a drive other than 'C')
4. Open cmd prompt and go to the specific folder using required commands
5. You should be in the folder containing java program to compile and execute
6. We can use 'javac' command to compile and 'java' command to execute

Rules for saving a java file for learners:

   1. Starting character of a class name should be in upper case (Industry Convention)

Ex: a. class Test (correct)

         b. class testclass (in correct)

         c. class TestClass (correct)

Note: When we are using multiple words for class name, every word first letter should be in caps

Ex: TestClass, MangoJuiceMixture.

   2. Class name and file name when saving as .java should be same

8ᵗʰ Sep 2016

Setting the path temporarily

1. Open command prompt
2. Copy the path of JDK/bin directory
3. Type set path=path of JDK.bin (paste it)

Ex: C:\users\Admin\set path=C: \Program files\Java\JDK1.8.0.102\bin
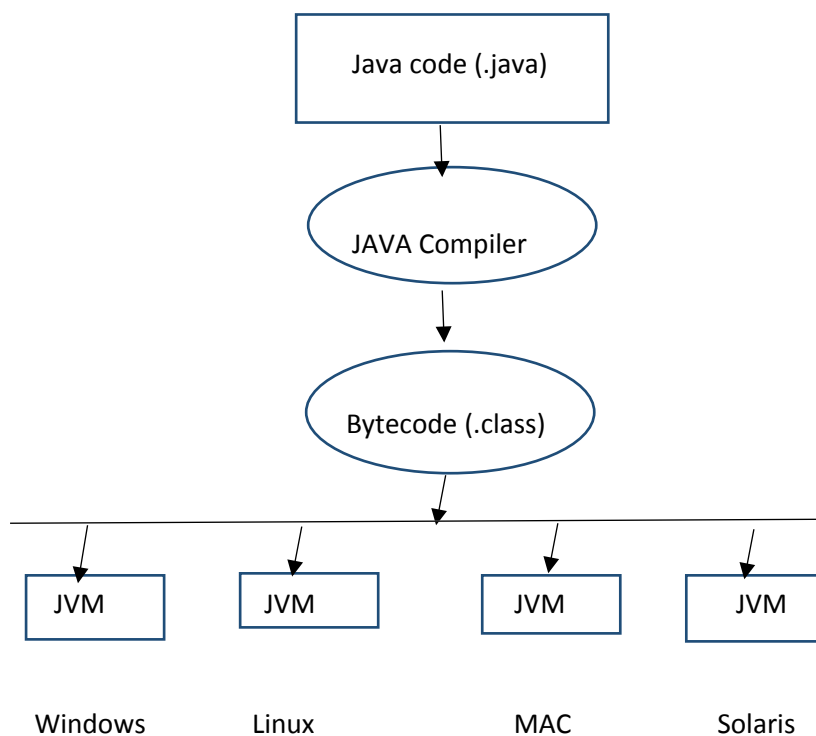
4. Click enter

What is platform?

It is a hardware or software environment where a s/w program can run

Why is Java both a programming language and a platform?

Java has its own runtime environment (JRE) to run the program and can be used to create s/w applications

How do you say Java is platform independent?

We can compile in to class file anywhere (any machine) and run it on any machine which has JVM

```
              ┌─────────────────────┐
              │   Java code (.java)  │
              └─────────────────────┘
                        │
                        ▼
                 (  JAVA Compiler  )
                        │
                        ▼
                ( Bytecode (.class) )
     ───────────────────┼───────────────────────
         │          │          │          │
         ▼          ▼          ▼          ▼
     ┌──────┐   ┌──────┐   ┌──────┐   ┌──────┐
     │ JVM  │   │ JVM  │   │ JVM  │   │ JVM  │
     └──────┘   └──────┘   └──────┘   └──────┘

      Windows     Linux       MAC       Solaris
```

What kind of applications can be created using Java?

- Desktop applications: such as MS word and Calculator
- Web applications: such as flipkart.com, facebook.com etc.
- Enterprise applications: such as banking applications, mobile, android applications etc.
- Games etc.

Interview question: what is the minimum lines of code required to compile/execute java program?

Answer: 1

Ex: class Demo{}

Note: We can write & compile using main {} method. But cannot run or execute without it.

What do you mean by Object oriented programming (OOPs)? - The program which is oriented around its objects

The programming language should support

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

What is JDK, JRE and JVM?

JDK: is an acronym for Java Development Kit. It physically exists. It contains JRE+ development tools

JRE: JRE stands for Java Runtime Environment. It is this environment where JVM executes the bytecode and contains class libraries & supporting files

JVM: JVM is an acronym for Java Virtual Machine. It is an abstract machine where java bytecode is executed. JVM's are available for many hardware and software platforms (So JVM is platform dependent)


The JVM performs

- Loads code
- Verifies code
- Executes code

Class loader: is the part of subsystem of JVM that is used to load class files

Bytecode verifier: Verifies the bytecode if it contains anything illegal can violate access rights to objects

Interpreter: It will read bytecode stream and then executes the instructions

```
Class file

Class loader

Bytecode verifier

Interpreter

Runtime

Hardware
```
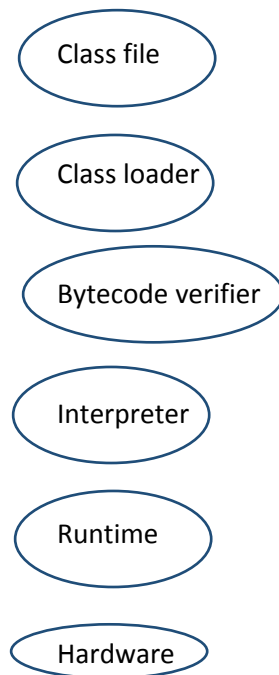
10ᵗʰ Sep 2016

Multiple lines of program

```
Class Demo2
{
        Public static void main(String…args)
        {
                System.out.println("Welcome all"+" to java class");
                System.out.println("sum pf 4 and 5 is" + 4+5);
                System.out.println("sum pf 4 and 5 is" + (4+5));
                System.out.println(); //to print a blank line
                System.out.print("Hello"); //when we use 'print', cursor remains in the same line.
                System.out.print("Kurnool");
        }
}
```

Very important Notes:

- **We can use (String[] args) or (String []args) or (String args[]) or (String… args)**
- **We can use public static or static public to define main method**
- **When should not use 'print' to print a blank line**

    Ex: println(); is correct
        Print(); is wrong

Rules for naming a class

1. Class names must begin (first letter) with a letter, an underscore (_) or a dollar sign ($)

Ex: [A-Z] [a-z] [$] [_]

2. From second character, class names may contain a letter, digits, an underscore (_) or a dollar sign ($)

Ex: [A-Z] [a-z] [$] [_] [0-9]

3. Class names cannot be a keyword (Java keyword)
4. Class name cannot have space between the words

50 Keywords in Java:

abstract	assert	boolean	break	byte
case	Catch	char	class	const
continue	default	do	double	else
enum	extends	final	finally	float
for	Goto	if	implements	import
instanceof	Int	interface	long	native
new	package	private	protected	public
return	Short	static	strictfp	super
switch	synchronized	this	throw	throws
transient	Try	void	volatile	while

Class: Class can be defined as a blue print/ template to create an object. A class specifies the design of an object. It states what data an object can hold and the way it can behave

Fields or variables: These are properties of the class or object which we are going to create

Ex: If we are creating a class called car, then they have properties like model no, Colour, seats etc.

In simpler way it can said as variables which holds values

"Variables are also called as fields"

Methods: Behaviour or the actions that can be performed by an object or a class

Class Name

{

Members of class -----} Body

}

In Members of class

Class - Methods (performs an operation)

Members - Variables or fields (Stores information)

Int I = 10;

In this 'int' is keyword

       'i' is identifier

       10 is literal

Tokens: The smallest individual entities in a Java program are called as tokens
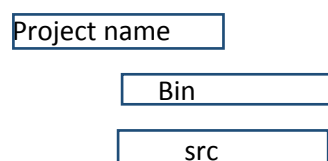
1. Reserved words (Key words (50))
2. Identifiers (Classes, methods, variables, objects, packages and interfaces)
3. Literals (values -10, 10 23, true, false, null)
4. Operators (+, -, --, ++ etc)
5. Separators (defines the boundary) ([], {}, (), ; etc)

12th Sep 2016

General practice in industry:

Project name

    Release name

      Bin

      src
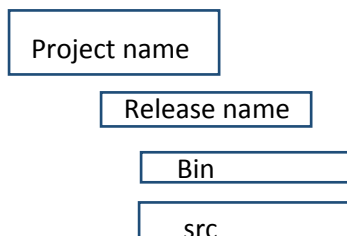
but we use as follows

Project name

    Bin

    src

If you organize this way, compiling and running will be in different path. You have to cut and paste the calls file in bin manually. This becomes a problem. You can alternately use below command

Ex: E:\Projectname\src> javac –d ..\bin Demo.java

The above command will automatically create and saves the class file in bin folder. It means instead of creating class file in the same path as java file, it generates the calls file in the path specified

To execute the class file (go to bin directory path)

Command is cd ..\bin

Then type

E:\projectname\bin java Demo and execute

Class file is the one which gets executed

Variables: Variable is named memory location which can hold value and we can change value any no of times during execution. It is also an identifier in java.

Rules to declare a variable:

1. Variable names must begin (first letter) with a letter, an underscore (_) or a dollar sign ($)

Ex: [A-Z] [a-z] [$] [_]

2. From second character, Variable names may contain a letter, digits, an underscore (_) or a dollar sign ($)

Ex: [A-Z] [a-z] [$] [_] [0-9]

3. Variable names cannot be a keyword (Java keyword)
4. Variable name cannot have space between the words

Legal or valid:

int _a;

 int a_very_long_variable_name_$

int $a;

int _$;

Illegal or invalid:

Int :a;   Int –d;   Int e#;   Int .f;   Int 7g;   Int 89_;

Comments: Comments help the person resulting the code better understand the intent and functionality of the program.

The comments are ignored by the compiler

There are three types of comments

1. // text – is the format for single line commenting
2. /* text */ - is the format for block commenting or multiline commenting
3. /** documentation */ - documentation commenting

Every class, methods, blocks inside the class should start with "{"and end with "}"

```
Class Demo3
{
   Public static void main(String…args)
   {
      int i=10;
      System.out.println("i"); //prints I (String)
      System.out.println('i'); //prints I (char)
      System.out.println(i); //prints value in i(variable)
   }
}
```

1. Inti; // declaration

   I=10; // initialization

2. Int i=10; // both declaration and initialization

3. System.out.println(i=10); // this will also works and prints the value of 'I'

Type: it describes the type of data which a variable can hold.

In java, type can be classified as
1. Primitive type
2. Non Primitive type

Primitive type:

byte, short, int, long, float, double, char, boolean

primitive variables: variables of all primitive data types are declared using data type name

int I;

double d=20.22;

'I' and 'd' are primitive variables

Type	Contains	Default	Size	Range
byte	Signed integer	0	8 bits or 1 byte	-128 to 127
short	Signed integer	0	16 bits or 2 bytes	-32768 to 32767
int	Signed integer	0	32 bits or 4 bytes	-2147483648 to 2147483647
long	Signed integer	0	64 bits or 8 bytes	-9223372036854775808 to 9223372036854775807
float	IEEE 754 floating point	0	32 bits or 4 bytes	±1.4E-45 to ±3.4028235E+38
double	IEEE 754 floating point	0	64 bits or 8 bytes	±4.9E-324 to ±1.7976931348623157E+308
char	Unicode character	\u0000	16 bits or 2 bytes	\u0000 to \uFFFF
boolean	true or false	FALSE	1 bit	NA

UNICODE - Universal character encoding

14th Sep 2016

Non primitive type

- Class
- Interface
- Array

Non primitive or Reference variables: variables derived from a class/any type or created using class/ type name (Derived)

String s;

Orange o1 = new Orange();

S and o1 are reference variables

Few more examples

Apple a1

Car c1

Demo d1

What to store for a1?

a1 is a variable of type Apple and can hold something which is of type Apple

Apple a1 = new Apple();

Car c1 = new car();

```
Class Orange
{
}
Class Demo5
{
    PSVM();
    {
        Int I = 123;
        Orange o1 = new Orange();
        SOP(o1); // prints address of the object
    }
}
```

Save it as Demo5. Java. When you compile you get 2 class files. That means Demo5. Class and Ornage. class.

You can execute a class file which has main method

Technically the reference variable will hold the address of the object it has been assigned

What is object and how is it created?

Object is an instance of a class. We can create using new keyword

Ex: new Orange();

New Apple();

Orange o1 = new Orange();

Why String is called a derived data type?

A variable of String class can hold a data of type String and since it is derived from a class, it is called as derived data type

Int I = 10;
Char ch = 'A'
String s1 = "Hello"; (we will call this as object)

Where can we use null?

-   We can use null for a reference variable

What is the default value of a reference variable?

-   Default value for any reference variable is null

How can you remove the reference of an object from a variable?

-   By assigning null value to it.

15th Sep 2016

There are 7 kinds of variables (as per jls7). Kinds determine the scope and behaviour of a variable.

They are
1. Local variable
2. Static variable
3. Non static (instance) variable
4. Method parameter
5. Constructor parameter
6. Exception parameter
7. Array components

Local variable: A variable declared with in a method or a block. Its scope is limited within the same method or block

```
Class Orange
{
}
Class Demo6
{
        Public static void main(String…args)
        {
                Int I; //declaration of a local variable
                I=10; // local variable should be initialized before using
                String s=null;
                Orange o1=null;
                Char ch=0; or char ch='\u0000';
                System.out.println(i);
                System.out.println(ch);
        }
}
```

O/p – 10
        Null

Static variable: It is a variable declared with in a class and with keyword 'static'. It is also called as class variable

```
Class Orange
{
}

Class Demo7
{
        Static int i=12; //static variable declaration
        Static orange o1= new Orange ();
        //Static int j;
```

```
        //J=20; // Static variable cannot be initialized in different lines
        // Initialization should be in single line
        // if not initiated it will take default value

        Public static void main(Strings…args)
        {
                System.out.println(i);
                System.out.println(o1);

        }
}
```

Output:

12
Ornage07852e922 (address of the object)

Local variable	Static variable
variable is declared within a method or block	variable is declared within a class and outside of any method or a block with keyword static
local variable should be initialized if we are using it. Otherwise it throws error	Static variable if not initialized will have default values (done by compiler)

The initialization can be done at same time of declaration or separately or at the time of usage(local)	If the static variable has to be initialized, then it can be done at global level like
	1. static int i=10 // correct
	2. static int I; //wrong and throws error
	3. or initialize locally with in method or block

Constant (final variable):

It is a variable value do not change during execution. We create it using the keyword 'final'.
The value assignment should be done at the time of declaration only.

Final variables are usually named in CAPS

Ex: final float PI=3.14;
Static final int NO_WHEELS=4;

Note: Uninitialized final variable is called blank final and this has to be initialized in static block.

```
Class Demo8
{
        Static final double PI=3.14; // constant value should be assigned in the same statement

        Public static void main(Strings…args)
        {
        Final int NO_WHEELS;
        NO_WHEELS=4; // can be done in same line or different line
```

```
        System.out.println(PI);
        System.out.println(NO_WHEELS);
        //NO_WHEELS=8; // will not work, cannot reassign value to a final variable
        // System.out.println(NO_WHEELS); // gives error


        }
}
```

Output: 3.14
       6

```
Class IQ1
{
        Public static void main(String…args)
        {
        System.out.println("10\n20\n30"); // prints the numbers in new line
        System.out.println("10\t20\y30"); // Prints numbers with tab space
        System.out.println(); // prints blank line
        System.out.println("print:\"world's end\""); // output is print: "World's end". We have to
use '\' to separate the double quotes or slashes in the statement
        System.out.println("\\n"); // output is '\n'
        }
}
```
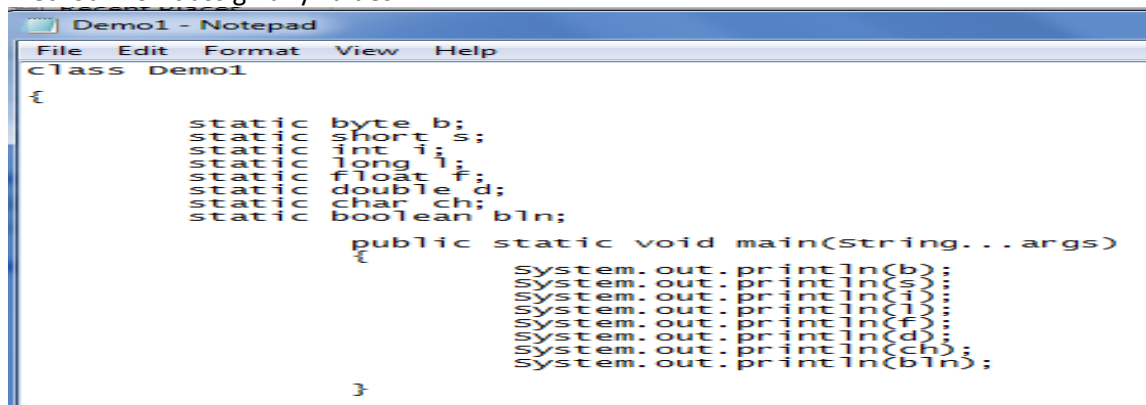
Assignment:

1. Write a program and declare all primitive variables as static, print the variables in main
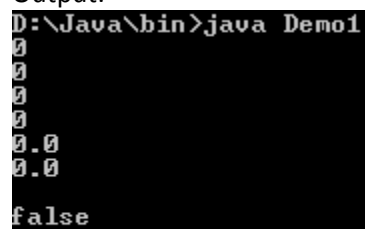   method. Don't assign any values

Output:

2. Write a program to print a local int variable don't initialize - **we will get error**
3. Program to print value 'true' -
4. Program to print value null – **we will get error**
5. Program to print value 100
6. Program to print a reference variable (static) and don't initialize -

Class Asg2
{

      Static String s1;

      Public static void main(String…args)

      {

            // Int I;
            // System.out.println(i); //local variable should be initialized
            System.out.println (true); // note that true is a value and "true' is a String
            // System.out.println(TRUE); // TRIE will be considered a variable and gives compile
            //error, because we have not declared any such variable
            // System.out.println(null); // cannot print value null
            System.out.println(100); // note that 100 is value and "100" is string
            System.out.println(s1);
      }
}

Note: true, false and null are literals

16th Sep 2016

Operators

An operator is a symbol or character that allows a programmer to perform certain operations like (Arithmetic and logical) on data and variables (Operands)

There are six types of operators are there

1. Arithmetic (+, -, *, /, %, ++, --)
2. Assignment (=, +=, -=, *=, /=, %=)
3. Relational (==, !=, <, >, <=, >=)
4. Logical (&, |, ^, !, &&, ||)
5. Conditional (?, ;)
6. Instanceof (instanceof)

Depending on the number of operators, operators can be of following three types.

Unary Operator: It takes operand such as ++X, Y--. Here X and Y are variables where ++ and – are operators.

Binary Operators: It takes **2 operands** such as X+Y, X-Y, X>Y etc. Here X and Y are variables where +, - and > are operators

Ternary Operators: It takes **3 operands** such as Z=X>Y?X:Y. here X, Y and Z are operands where ?, :, > are operators

Arithmetic operator	Description
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus
++	Increment
--	Decrement

```
Class Demo9
{
        Static public void main(String…args)
        {
                Int I,j,k;
                I=30;
                J=20;

                K=i+j;

                System.out.println("Sum of I and j is "+k);
                //System.out.println("Sum of " + I + " and" + j + "is "+k);

                K=i-j;

                System.out.println("Difference of I and j is "+k);

                K=i*j;

                System.out.println("Product of I and j is "+k);
                // System.out.println("Product of I and j is "+ (i*j));

                K=i/j;

                System.out.println("Division of I by j is "+k);
                // System.out.println("Division of I by j is "+ (i/j));


                Int quotient= i/j
                System.out.println("Quotient of I by j is "+quotient);

                Double d = i/j;
                System.out.println(d); // output is – 1.0 (double type value)

                d= 30.0/20.0;
                System.out.println(d); // output is – 1.5 (double type value)

                d= 30/20.0;
                System.out.println(d); // output is – 1.5 (double type value)

                d= 30.0/20;
                System.out.println(d); // output is – 1.5 (double type value)

                System.out.println(30/20); // output is – 1 (int type value)

                //K=30.0/20.0; // compile time error (cannot assign double to int directly)
                // System.out.println(k);

                Int remainder = i/j;
                System.out.println(Remainder of I by j is " + remainder);
```

System.out.println('A' + 'B');// output - 131, because it takes that characters Unicode values
System.out.println('A' + "B");// output – AB, because "B" is string. It will concatenate both
System.out.println("A" + 'B');// output – AB, because "A" is string. It will concatenate both
System.out.println("Hello" + 'A' + 'B');// output – Hello AB, because "Hello" is string. It will concatenate all
System.out.println('A' + 'B'+ "Hello");// output – 131 Hello, because first it will check left to right and takes first 2 characters and checks whether they have any Unicode values and do that operation. Next it will concatenate this value to string "Hello" value.

System.out.println('A'); // output – A. prints char value
System.out.println("A"); // output – A. prints string value

   }
}

Note: **int/int = int**
                 Int/double = double
                 double/int = double
                 double /double = double

Important notes:

'A'+'B' = 131
'A'+'\n' = 75 // Unicode value of \n is 10. So it will add this 10 to 65 (Unicode value of A)
'A'+''\n'' = 75 // It will concatenate, since ''\n'' is <u>string</u>
'A'+20 = 85
''A''+20 = A20
'A'+''A'' = AA
''A'' + 'A' = AA
''A''+''A'' = AA

Assignment operators: The equal (=) symbol is the assignment operator. The assignment operator (=) is used to store or assign a value to a variable

Simple assignment operator – example - i=10

In compound assignment operator we have (+=, -=, *=, /=)

Ex: i+=10 => i=i+10

         i-=20 => i=i-20

         i*=20 => i=i*20

         i/=30 => i=i/30

         i%=30 => i=i%30

         Orange o1 = null;

         O1 = new Orange();

Class Demo9

{

        Static public void main(String…args)

        {

        Int i=2, j=3, k=4, l=25, m=7;

        I+=5;

        J*=6;

        k/=2;

        l-=10;

        m%=5;

        System.out.println(i);

        System.out.println(j);

        System.out.println(k);

        System.out.println(l);

        System.out.println(m);

        }

}

Output is

7
18
2
15

2

17ᵗʰ Sep 2016

Relational operators:

Operator	Description	Example
== (equal to)	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!= (not equal to)	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
> (greater than)	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
< (less than)	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>= (greater than or equal to)	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<= (less than or equal to)	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

```
Class Demo10
{
        PSVM()
        {
        Int X=5, Y=10;
        Boolean res= X==Y;
        SOP(res); // output – false
        SOP(X==Y); // output – false
        SOP(X==10); // output - false
        SOP(5==10); // output – false

        res = X!=Y;
        SOP(res); // output – true
        SOP('res'); // gives compile error, because it is character (which is mentioned in single
quote) and only one letter should be there.
        SOP("res"); // output – res, because it is a string and a string can have any no of letters

        Res = X>Y;
        SOP(res); // output – false
```

```
        Res = X<Y;
        SOP(res); // output – true

        Res = X>=Y;
        SOP(res); // output – false

        Res = X<=Y;
        SOP(res); // output - true

        SOP(x=10);// x will be assigned with value 10

        SOP(X==10); // output - true, because the value 10 is assigned to x in previous line

        SOP("x= " +x);

        SOP("x= " +(x=123));

        //SOP(TRUE); // compile time error, because TRUE is not value and it will consider as
variable. We have not declared the variable as such in the program.

        SOP(true); //output – true

        SOP(100); //output – 100

        Int a=100; b=100; c=100;

        Int I,j,k;
        I=j=k=100;

        //Int i=j=k=100; // wrong. We should not assign the same value to multiple variables in single
line

        Int l,m,n;
        //l=10, m=20, n=30; // wrong, we have to separate the value by 'semicolon (;)'

        l=10; m=20; n=30; // correct

        //Orange o1, o2=new Orange(), o3, o4=new Orange();

        SOP("a= "+a+"b= "+b+"c= "+c);


        }
}
```

==Next program==

```
Class Orange
{
}

Class Demo11
{
        PSVM()
        {

        Int x=5, y=10;
        SOP(x==y);
        SOP(x==5);
        SOP(10==y);
        SOP(10==5);

        Orange o1= new Orange(), o2= new Orange();

        SOP(o1);
        SOP(o2);
        SOP(o1==o2); // false
        SOP(o1!=o2); // false
        //SOP(o1>o2); // wrong - we cannot compare address or reference values as greater or
lesser
        SOP(x>y);

        o1=o2;
        SOP(o1==o2);//true

        O1=null;
        O2=null;

        SOP(o1==o2); // true

        }
}
```

== is a relational equality (==) and (!=) operator which is used

1. Can compare to two primitive variables. where it will compare values

Int x=10, y=20;

   a. x==y
   b. x==10
   c. 10==20

2. It can compare two reference variables whet it will compare addresses (compare the references of the two objects

instanceof operator;

The instanceof operator is binary operator that checks whether an object is of a particular type (here type can be class, interface or any array). **It is used for object or reference variables only**

You cannot use the instanceof operator to test across two different class hierarchies

```
Class Orange
{
}
Class
{
        Static orange o1;
        PSVM()
        {
                SOP(o1);
                SOP(o1 instanceof Orange); // true
                O1=null// means we are removing reference
                SOP(o1 instanceof Orange); // false. Because we have removed the reference in the above line.


                String s= new String ("Hello");
                If(s instanceof String)
                SOP("S is instanceof String class");
                S=null;
                If(s instanceof String)
                SOP("S is not instanceof String class");
                If(s instanceof String)
                SOP(!("S is not instanceof String class"));// true
        }


}
```

18th Sep 2016

Logical Operators:

Category	Operator
AND	&
OR	|
NOT	!
OR (Short Circuit)	||
AND (Short Circuit)	&&

The NOT (!) operator returns the opposite of the current value of a Boolean operand.

The operators || and && evaluate only Boolean value

a	b	a&b/ a&&b	a|b/ a||b
TRUE	TRUE	TRUE	TRUE
TRUE	FALSE	FALSE	TRUE
FALSE	TRUE	FALSE	TRUE
FALSE	FALSE	FALSE	FALSE

Interview question:

What is the difference b/n | and ||, & and &&?

In a situation of multiple condition

   a. (|| and |) if first condition is true then || does not check for second condition, whereas even if the fist condition is true | will check for second condition also
   b. (&& and &) if first condition is false then && does not check for second condition, whereas even if the fist condition is false & will check for second condition also

```
Class DemoBitwiseAndOR
{
  PSVM()
  {
        Int x=10, y=8;
        SOP(x&y);// o/p - 8
        SOP(x|y); // o/p - 10
        SOP(x^y); // o/p - 2
  }
}
```

Here it will do bitwise operations

Bitwise representation for 10 is – 1010

Bitwise representation for 8 is – 1000

Now the results for & is – 1000 means – 8

Now the results for | is – 1010 means – 10

Now the results for ^ is – 0010 means – 2 (here if both are different then it is 1 or it is 0)

Conditional Operator – The Conditional Operator (? :) is a ternary operator that takes three operands. It works similar to if else statement.

Syntax:

Operand 1? Operand2: operand3;

The first operand is a Boolean expression, if the expression is true then the value of the second operand is returned, otherwise the value of third operand will be returned

Boolean expression? value1:value2;

```
Class Demo13
{
  PSVM()
  {
        Int x=10;
        Int y=x>12?100:200;
        SOP(y);
        Char ch = 12.34==12.3?'A':'B';
```

```
SOP(ch);

Int marks = 55

String res = marks>50? "pass":"fail";

SOP(res);

 }
}
```

Input from user

Demo to take the input from user

```
import.java.util.Scanner;

class Demo12

{

 PSVM()

 {

        Scanner sc = new Scanner(System.in);

        SOP("Enter your name");

        String name = sc.nextline ();// takes String input

        SOP("Enter your marks");

        Int marks = sc.nextint();//takes int input

        SOP("Enter your subject");

        Sc.nextline();

        String subject = sc.nextline();//takes String input

        String res = marks>=50?"pass":"fail";

        SOP("Hello " + name + "your result is " + res + "in the subject" + subject);

 }
}
```

Few more examples:

//Result is assigned to the value 1.0

Float result = true?1.0f:2.0f;

//Result is assigned the value

"Sorry dude, it is false"

String result = false?"Dude it is true":"Sorry dude, it is false";

// int x=1;

String returnString = "There" + (x>1? "are " + x + "biscuits": "is one biscuit");

SOP (return String);

// highest to two numbers

Int i=10, j=20;

SOP(i>j? i+ "greater than "+j: j+ "is greater than "+i);

String result = i%2==0?"a":i%3==0?"b":i%5==0?"c":"d";

Assignment 1: write a program to take name, marks and subject from user and shoul print name and the results

Ex: hello Pinki you have passed in science

Assignment 2: Take marks as i/p from user and classify it as fail, just pass, first class, distinction or invalid using nested ternary operator.

```
Import java.util.Scanner;

Class Demo19
{
  PSCM()
 }
        Scanner sc = new Scanner(System.in);

        SOP("Enter your name");

        String name = sc.nextline ();// takes String input

        SOP("Enter your marks");

        Int marks = sc.nextint();//takes int input

        String res = marks>=0&marks<50? "fail: marks>=50&marks<60? "just pass":
        marks>=60&marks<75? "first class": marks>=70&marks<100? "Distinction":"invalid";
```

//String res = (marks>=75)? "Distinction: (marks>=60)?"First calss": (marks>=35)?"Just pass": (marks>=0)?"Fail": Invalid;

Sop(res);

}

}

19th Sep 2016

Unary operator

Operator which work on single operand. It is an arithmetical operator

(++):- increment operator

(--):- decrement operator

These are again classified in 2 types

Pre increment: ++I (first increment and next use)

Post increment: i++ (first use and next increment)

Pre decrement: --I (first decrement and next use)

Post decrement: i—(first use and next decrement)

Class Demo16

{

  PSVM()

  {

        Int i=0;

        Int j=i++;

        SOP*(i);

        SOP*(j);

  }

}

o/p –

1 (I value)

0 (j value)

```
Class Demo17
{
 PSVM()
 {
        Int i=0, j=0;
        J= I + i++ + I + i++;// (0+0+1+1)
        SOP*(i); // o/p - 2
        SOP*(j);// o/p - 2
        Int k=0,l=0;
        K=++1;
        SOP(k);// o/p - 1
        SOP(l); // o/p - 1
 }
}
Class Demo18
{
 PSVM()
 {
        Int i=0, j=1;
        Int k = I + j++ + ++I + ++j = i++;// (0+1+1+3+1)
        SOP*(i); // o/p - 2
        SOP*(j);// o/p - 3
        SOP(k);// o/p - 6
         }
}
```

Methods: A method describes the behaviour or actions that can be performed by an object or a class.

Method can return a value and take arguments (parameters).

Syntax:

Modifier 'return type' name of method (parameters list)

{

//method of body

}

- Modifier – it describes the access type/ behaviour of the method and it is optional to use (ex: public or static)
- Return type: method may return value (ex: void)
- Name of Method: This is the method name. the method signature consists of the method name and parameter list (ex: main())
- Parameter list: the list of parameters, it is the type, order and number of parameters of a method. These are optional
  Method may contain zero or no parameters
- Method body: the method body defines what the method does with statements

Class Demo19

{

  PSVM()

 {

     Print(); // calling 'print' method

     Int total=sum(10,20);// o/p is stored in a variable

     SOP(total);

     SOP(sum(100,200));//o/p is input to println

     Sum(123,456); //we will not get any o/p, because no method defined for this

     SOP(sum(sum(123,456), sum(111,222))); o/p is sum of first inner method and second dinner method. Now these will acts as input to other sum method

     SOP(avg Temp (22.6, 26.4));

     SOP(wish ("Lolly"));

     Ststic void print()

     {

     SOP("I love java and selenium");

     Return; //optional, since void is the return type

```
        }

        Static int sum (int n1, int n2)

        {

        Int toal = na+n2;

        Retrn total; // returns int

        }

        Static double avgTemp(double t1, double t2)

        }

        Double avg = (ti+t2)/2.0

        Return avg;

        }

        Static String wish (String name)

        {

        Return "Hello"+ name;

        }

   }
}
```

20th Sep 2016

Very important notes:

- If method is not returning anything use void as returns type
- If the method is not returning anything you can skip return or use just 'return;'
- To call the method, use method name.
- Write methods in class definition block
- If return type is mentioned, compulsorily it should returned the value of type mentioned
- If return type is mentioned as void & returns something from method, it will generates compile time error
- A method can return nothing (void), can return any primitive type or can return any object or can return an array or collection

```
Class Orange
{
}
Class ReturnTypeDemo
{
        PSVM()
        {
                //sample methods
                Static void print ()//method doesn't return anything, because void is the return type
                {
                        Return; //return is optional, because the return type is void
                }
                Static byte print ()
                {
                Byte b=25;
                Return b;
                }
                Static int print ()
                {
                int i=100;
                Return i;
                }
                Static char print ()
                {
                Return 'A';
                }
                Static boolean ismarried ()
                {
                Boolean status = false;
                Return status;
                }
```

```
Static string wish (String name)

{

Return "Hello " + name;

}

Static Orange getOrangce()

{

Return new Orange();

}

Static Orange getApple()

{

Apple a1 = new Apple();

Return a1;

}

Static int[] getmarks()

{

Int[] marks = {65, 75, 85,95};

Return marks;

}

Static double[] gettemp()

{

double[] temps = {22.4, 25.4, 23.5, 26.2};

return temps;

}

Static Orange[] getOrange()

{

Orange o1 = new Orange();

Orange[] oranges (Orange oranges[]) = { new Orange(),new Orange();o1};

Return Oranges;

}

}

}
```

Assignment:

What are methods? What is method syntax?

Write a method which can return two int values?

What can methods return?
Write a method to find the area of a circle?

Write a method to find the area of a square?

Write a method to find the volume of a cube?

Write a method to find the volume of a cone?

Write a method to find the volume of a cylinder?

Write a method to find the total amount to be paid
prinipal,rateof interest and time -all user input
hint-ptr/100

Write a method to return Orange object?

Write a method to return marks of 4 subject

42

21st Sep 2016

```
Class Dmo17
{
        PSVM()
        {
                Int i=0;
                Int j;
                J=test(i) + ++j + i++; // test is a method defined below
                SOP("i= " +i);
                SOP("j= " +j);


        }


        {
                Static int test (int a) // for this a the value will pass from the test (i)
                {
                        SOP("a= " + a++);
                        Return ++a;
                }
        }
}
```

Note: Here we are calling 'test' method name and the main method computed value will be input to the method below. **It means test (i) value is input to int a.**

```
Class Demo 18
{
        PSVM()
        {
                Int a=2; int b = 5;
                Int c = a++ + b++ + test(b++);
                SOP("a= " + a);
                SOP("b= " + b);
                SOP("c= " + c);

                Static int test (int a)
                {
                        A++;
                        Return a++;
                }
        }
}                               o/p – a=3; b=7; c=14;
```

Control Statements: these are the statements which helps in controlling the flow of code.

There are 2 types of statements

1. Branch (Decision) statements
2. Loop statements

Decision or Branch statements:

These statements allow you to create programs where decisions are taken based on the expression specified in the statement.

1. If statement
2. If.. else (else..if) statement
3. Switch statement
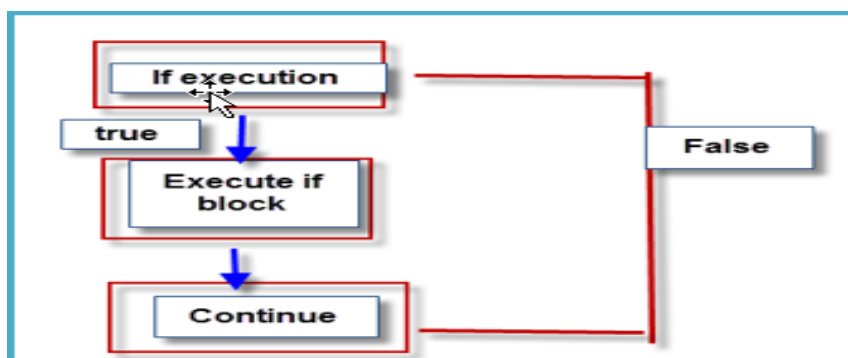
Basic structure of if statement

If block is executed only when if expression is true, otherwise if block is skipped.

Note: We must use Boolean expression to make decisions

Public class Demo21

{

PSVM()

{

Int age = 16;

If(age<18)

{

SOP("Not eligible to vote");

}

}

}

O/P – not eligible to vote

If (true)// works

If(age=16)// compile error, because we should not use assignment operator. Means it will not return Boolean value

If(age==16)// works

If(false)// works but no o/p. because the condition=ion is false.

If(!false)// works and return o/p
If)false==false)// works and returns o/p
Boolean var = true;
If(var)// works
If(var=true)// works
If(false);// (end of statement. block gets executed irrespective whether the condition is true or false as it become independent. Because when we put ;, it will consider as end of that line or statement. It do not have any relations with below lines.

Note: we will see o/p when if condition is true. When the condition is false we will not see any o/p

Block gets executed irrespective whether the condition is true or false as it become independent

When we mention the multi lines below to if statement and not mentioned braces, then only the immediate next statement will consider for if condition if the results is true. Other lines are considered as independent and they will execute as usual

When we mention the multi lines below to if statement and mentioned braces, then all the statements within the braces will consider for if condition

Ex: if we change the age to 18 or more, then the next statements will not get executed. Because the condition is false. The other lines will get executed and give o/p. please find below program for the same

Int age=18;

If(age<18)

{

SOP("not eligible to vote");

Age=age+2;

SOP("age of person is" + age);

}
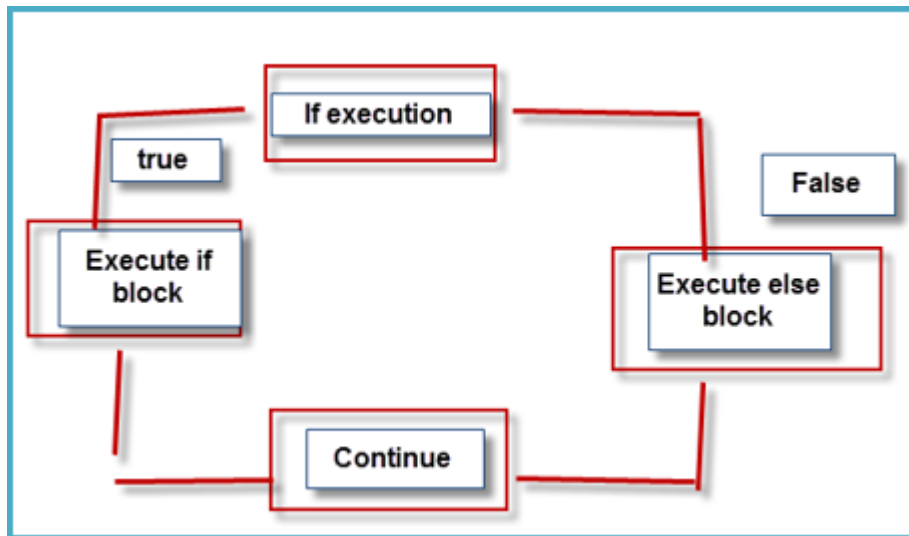
Basic structure of the if else statement

Public class Demo22

{

       PSVM()

       {

       Int i=100, j=200;

       If (i>j)

       SOP(i+ "is greater than "+j);

       Else

       SOP(j+ "is greater than "+i);

       }

}

O/p –

200 is greater than 100.

Note: in the above **if block**, you have only one line. If you want multi lies, you have to use braces.

In else block, if you do not put braces ({}), will take first line. Other lines it will consider as independent.

Assignment on 9/21/2016

1.public class Test{

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

     int x=1;
   int x = x++;
   int x = ++x;
   int x = x++;
   System.out.println(x);


  }

}
```

2.public class B{

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

   {
     int x=1;
    int x= x++;
    int y= x++;
    int z= x++;
   System.out.println(x);

 }
```

```
}
```

3.public class A{

```
static int a = 1111;
static
{
    a = a-- - --a;
}


{
    a = a++ + ++a;
}


public static void main(String[] args)  {

    System.out.println(a);

  }

}
```

4.public static void main(String [] args){

```
  int i;

  for( i=1;i<4;i++){

    System.out.println(i);

  }
```

```java
    System.out.println("value of i after completion of loop: "+i);

}
}


5.class IncrementDemo{

public static void main(String [] args){

  int a=10,b=10;

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

    if(++a>2||++b>2){
      a++;
     }


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

}
}


6.class Demo{

public static void main(String [] args){

  int a=1;

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

```java
System.out.println(a++);
System.out.println(++a);


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


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


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

}
}
```

7.class Demo{

public static void main(String [] args){

```java
    int a=20;
    a= ++a + ++a;


    System.out.println(a);

}
}
```

8. what will the following code print;

```java
class Test{
```

```java
  public static void main(String []args){
   int i = 1;
  int j = i++;
  if( (i==++j) | (i++ == j) ){
   i+=j;
  }
  System.out.println(i);
  }
 }
```

9.
```java
public class BreakTest{
  public static void main(String[] args){
   int i = 0, j = 5;
   lab1 : for( ; ; i++){
    for( ; ; --j)  if( i >j ) break lab1;
   }
   System.out.println(" i = "+i+", j = "+j);
  }
}
```

10.
```java
public class SimpleLoop {
   public static void main(String[] args) {
      int i=0, j=10;
      int count = 0;
      while (i<j) {
       i++;
        j--;
        count++;
   }
      System.out.println(i+" "+j+" "+count);
   }

}
```

Sep 22th 2016

//Multiple if else statement

```
Public class Demo21
{
PSVM ()
{
        Int a=0, b=20, c=30;
        If (a>b && a>c)
        SOP(a + " is greater than " + b + " and "+ c);
        Else If (b>a && b>c)
        SOP(b + " is greater than " + a + " and "+ c);
        Else If (c>a && c>b)
        SOP(c + " is greater than " + a + " and "+ b);
}
}
```

Note:

1. An if statement can be used without an else statement
2. Multiple is else statements can be used in a program
3. Once an if else statement causes an action in a program, then the remaining if else statements will be ignored

Switch case statement

The switch case statement is used to select an action from a given set of actions, based on a specified expression.

Syntax:

Switch (expression/ variables)

{

Case value 1: statement1

Break;

Case value 1: statement1

Break;

Case value 1: statement1

Break;

Default: default statement;

}

The expression/ variable in the preceding code snippet can be any expression depicting a char, byte, short, int or enum variable. The switch case also support some wrapper classes like integer, byte, short

(In JDK1.7, we can also use String in values)

Import java.util.Scanner;

Class switchDemo

{

PSVM()

{

Scanner sc = new Scanner(System.in)

SOP("Enter your atomic number");

Int atomno = sc.nectInt();

Final int x=34;// we can use only constant variables. When we use keyword final means, it is a constant value

```
Switch (atmno)

{

        Default:

        SOP("Invalid no");

        Break;

        Case 1:

        SOP("Hydrogen");

        Break;

        Case 2:

        SOP("Helium");

        Break;

        Case 3:

        SOP("Lithium");

        Break;

        Case 8:

        SOP("Oxygen");

        Break;

        Case x:// here 'x' is a constant variable declared as final in above line and the value is 34.

        SOP("Selenium");

        Break;

}

}

}
```

We have to use 'break' statement in each case, otherwise it will executes next line till it finds break statement or end of the program.

Default statement when we use at end, we need not to write break statement, because it is at the end. If we use starting of the program, we have to use break statement to stop the flow.

Assignment on 22ⁿᵈ.

```java
import java.util.Scanner;

class Demo24
{
  public static void main(String...args)
  {
   Scanner input=new Scanner(System.in);
   System.out.println("Enter your marks");
    int marks=input.nextInt();


    if(marks>=0 && marks<50)
       System.out.println("Fail");
    else if(marks>=50 && marks<65)
        System.out.println("Pass");
    else if(marks>=65 && marks<75)
       System.out.println("First class");
    else if(marks>=75 && marks<=100)
        System.out.println("Distinction");
    else
       System.out.println("Invalid");*/
  }

}




class Demo24
{
  public static void main(String...args)
  {
   Scanner input=new Scanner(System.in);
   System.out.println("Enter your marks");
    int marks=input.nextInt();


    if(marks>=0 && marks<50)
       System.out.println("Fail");
    else if(marks>=50 && marks<65)
        System.out.println("Pass");
    else if(marks>=65 && marks<75)
       System.out.println("First class");
    else if(marks>=75 && marks<=100)
        System.out.println("Distinction");
    else
       System.out.println("Invalid");*/
  }

}
```

1.Wap to Simulate Traffic Signal Light,and also it should support amber,yellow and orange which means caution. Green means Proceed and red means stop.

```java
import java.util.Scanner;

class TrafficSignal
{
  public static void main(String[] args)
  {
    Scanner input=new Scanner(System.in);
    System.out.println("Enter traffic color");
    String color=input.nextLine();
        color=color.toLowerCase();
    switch(color)
    {
      case "green":
           System.out.println("Proceed");
           break;
      case "orange":

      case "yellow":

      case "amber":
           System.out.println("Caution!");
           break;

      case "red":
          System.out.println("Stop!");
          break;

    default:
        System.out.println("Invalid color");
    }


  }


}
```

2.Wap to print the element for given country code

```java
import java.util.Scanner;
class Demo22{
  public static void main(String args[])
  {
    System.out.println("Enter country code");
     Scanner sc=new Scanner(System.in);
     String input=sc.nextLine();
    input=input.toUpperCase();
     char cc=input.charAt(0);
```

```java
//char countryCode='Z';

switch(cc)
 {
   case 'I':
        System.out.println("India");
        break;

   case 'J':
        System.out.println("Japan");
        break;
   case 'A':
        System.out.println("America");
        break;
   default:
        System.out.println("Invalid choice");

 }
}
}
```

Electric bill program:

```java
import java.util.Scanner;

class EBill
{
   public static void main(String[] args)
  {
    Scanner input=new Scanner(System.in);
    System.out.println("Please Enter units consumed");
    int units= input.nextInt();

    System.out.println("Enter arrears if any else enter 0");
    double arrears=input.nextDouble();
    double amt=0.0;
    double discount=0.0;
    double fineamt=0.0;

    if(units>=0 && units<=50)
     amt=units*1.20;
    else if(units>50 && units<=100)
     amt=(50*1.20)+(units-50)*2.40;
    else if(units>100 && units<=150)
     amt=(50*1.20)+(50*2.40)+(units-100)*3.60;
    else if(units>150)
    {
     amt=(50*1.20)+(50*2.40)+(50*3.60)+(units-150)*4.80;

     if(units>200)
       discount=amt *(5.0/100.0);

     if(discount>200)
       discount=200;
    }
```

```
    if(arrears>0)
     fineamt=arrears*(10.0/100.0);


    double totalamt=amt+arrears+fineamt-discount;

    System.out.println("Total amount payable: "+ totalamt + " Rs");

   }
}
```

23rd Sep 2016

When can we use if else and switch case?

If statements are used to evaluate Boolean expression. A switch statement allows a variable to be tested for equality against a list of values. Each value is called a case and the variable being switched on is checked for each other.

When we range of values, relational operators or logical operators, then we can use if else

If we are using only values (list of values or 1-1 mapping) then we go for switch case.

Loop statements

Loops are statements preferably with braces where the code in the block ({}) gets executed certain number of times based on the condition.

Implementing the for loop:
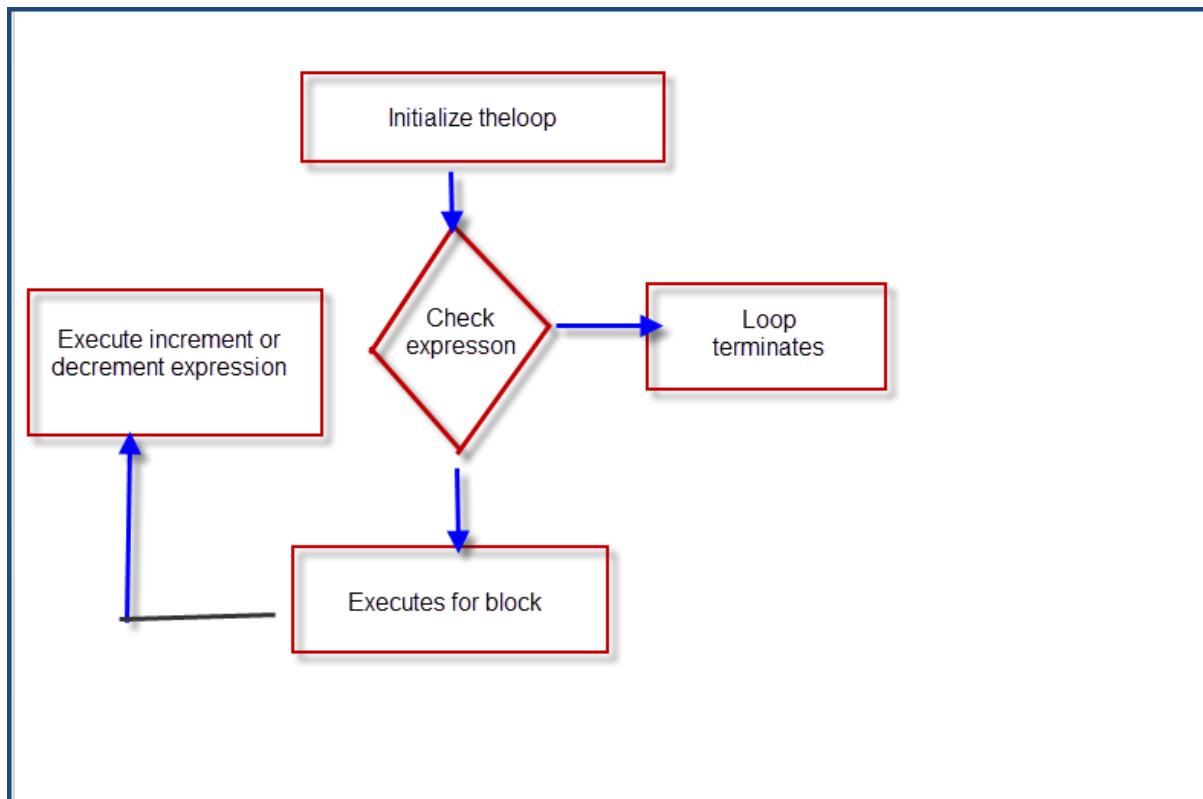
Basic structure of for loop

For loop is initialized first and then the Boolean expression is checked. If the expression evaluates to true, then the for block is executed, otherwise the loop terminates. If the for block is executed, then the increment or decrement expression is updated to continue loop.

Synatx:

For (initialization;condition;update)// if we put ; at the end of line, it means it will end the statement and no further loop will be executed. That line is independent.

{

//body

}

The 3 parts of the for loop – initialization, expression and update, must be separated by semicolon (;). Or else the program will lead to a compile error.

Public class Demo23

{

      PSVM()

      {

            For (int I =1; i<=20;i++)

```
                    {

                    SOP(i);

                    }

          }

}




class LoopDemo1
{
  public static void main(String...ags)
  {

    for(int i=1;i<=10;i++)
        System.out.println("I love Java");

  }


}
```

Different for loops

```
    a)
      for(i=1,j=1;i<=20;i++)
      {
       System.out.println(i);
       System.out.println(j);
       //it works but not right to use it this way
      }




    b)
      for(int i=1,j=1;i<=5 && j>1;i++,j++)
                          //it is legal
      {
         System.out.println(i);
         System.out.println(j);
      }

    c)
      for(int i=1,j=1;i<=5,j>1;i++,j++)
      {
        //multiple expressions (compiler error)
         System.out.println(i);
         System.out.println(j);
      }
    d)
         int i=1,j;
         for(j=1;i<=20;)
         {
           System.out.println(i);
```

```
            System.out.println(j);
                i++;
        }
```

//above code will work but it acts more like while loop

e)
```
    for(int i=10;i>5;i--)
    {
        System.out.println(i);
    }
```
// above code is an example for decrementing for loop
f)
```
    for(int i=1; ;i++)
    {
        System.out.println(i);
    }
```
// infinite loop
g)
```
    int i=1;
    for( ; ;) in
    {
        System.out.println(i);
    }
```
// infinite loop
h)
```
        int i;
        for(i=1;i<=20;++i) //works absolutely fine
        {
            System.out.println(i);}   }
```

i)
```
    int i=1;

    for(;true;)
    {
        System.out.println(i);
            ++i;

        if(i==10)
        break;
    }
```

j)
```
    for(int i=1;i<=10;i=+1)
        System.out.println(i);
```

k)
```
int j=1;
                    for(int i=1;i<=10;i++);
                        System.out.println(j);//1 but cannot access i
```

l)
```
    int i=1;
                    int j=1;

                    for(;i<=10;i++);
                    {
                        System.out.println(j);//1
                        System.out.println(i);//11
                    }
```
i)
```
    for(int i=1;i<=10;i=+1)
                            System.out.println(i);
```

j) for(int i=1;i<=20;i++)
        System.out.println(i);
     // System.out.println(i);//compiler error -scope of is

            //limited to first line*/

24th Sep 2016

While loop:

Public class Demo24

{

        PSVM()

        {

                Int i=1;

                While(i<=20)

                {

                        SOP(i);

                        I+=2;

                }

        }

}

Class Demo25

{   PSVM();{

                Int i=1;

                While(i<=10)

                SOP(i++);   }

}

Note: While loop can be used with non numeric conditions also. Like checking a character in a variable or checking for a string value in a variable

Ex:

```
Char ch;
While(ch!='y')
{
------------- body
}
```

Ex for non numeric condition

```
Import java.util.Scanner;
Class WhileDemo{
Psvm()
{
Scanner input = new Scanner(System.in);
Char ch = 'n';
String ans = "";
While (ch!='y')
{ SOP("Will you lisyen to me?");
Ans = input.nextLine();
Ch = ans.charAt(0);// used to convert chart at '0' place to lowercase}}}
```

Example for comparing a String n a while

```
Import java.util.Scanner;

Class WhileDemo2{
PSVM(){
Scanner input = new Scanner(System.in);
String  ans="no";
While(!ans.equals("yes")){
SOP("Will you listen to me?");
Ans = input.nextLine();}}}
```

Do while:-

Do while loop works similar to while loop. But the difference is do while loop block executes at least once irrespective of whether the expression evaluates to true a false
(while loop, is an entry check loop and do while is an exit check loop)

What is the difference b/n for loop and while/ do while loop?

- For loop checks for numeric conditions and have definite iterations
- While or do while loop can have both numeric and non numeric conditions
- It has definite iterations with numeric condition and not finite iteration with non numeric conditions

```
Public class Demo26{
PSVM(){
Int i=1;
Do{
SOP(i++);} while(i<=20);}}
```

o/p – 1 to 20 in a column


```
Public class Demo27{
PSVM(){
Int i=1;
Do{
SOP(++i);} while(i<=20);}}
```

o/p – 2 to 20 in a column

```
class  Biscuit
{
  public static void main(String...arg)
  {

    int i='A';
    System.out.println(i);

    char ch='B';
    System.out.println(ch);

    ch=67;
   System.out.println(ch);
   System.out.println(++ch);

   System.out.println(ch + 1);

//   ch=ch+1;
    int k=ch +1;
   System.out.println(k);


   for(i='A';i<='Z';i++)
     System.out.print(i);



   for(char ch1=65;ch1<=90;ch1++)
    System.out.print(ch1);
```

```
    System.out.println("----------------");

    for(char ch1='Z';ch1>='A';ch1-=2)
      System.out.print(ch1);

  }
}
```

1. Write a program to print "I love Java" 10 times without writing anything in body

For (int i=1;i<=10;i++,SOP("I love java"));

2. Write a program "I love java & selenium", each word in a separate line using single SOP

SOP("I/nlove/njava/nand/nselenium");

25th Sep 2016

Very imp for selenium

Implementing the for each loop (enhanced for loop)

The for each loop is used to iterate over arrays and collections. The for each loop has only two parts, unlike the traditional which has 3 parts.

The two parts of the for each loop are variable and array collections

Syntax:

For (variable: collections or arrays)

{

        -----body

}

Class Orange(){

Class ForEachDemo{

```
PSVM(){

Int marks[] = {70, 45, 60, 78, 89}

For (int i=0; i<5;i++)

SOP(marks[i]);

Sop("---------");


Foat temps[] = {28.7f, 23.6f,33.5f,34.7f, 56.4f};

For (int i=0; i<4;i++)

{

SOP(temps[i]);

}

Sop("---------");


String names = {"sheela", "nikhitha", "amrith", "das", "lolly"};

For (int i=0;i<5;i++)

SOP(names[i]);

Sop("---------");


Orange o1= new Orange();

Orange oranges[] = {new Orange(),new Orange(),new Orange(),o1};

For (int i=0;i<4;i++)

SOP(oranges[i]);


Sop("---- Using enhanced loop -----");

For (int mark: marks)

SOP(mark);

SOP("----");

For (float temp: temps)

SOP(temp);

SOP("----");
```

For (String name: names)

SOP(name);

SOP("----");

For (Orange 0: Oranges)

SOP(o);

SOP("----");

}}

Very imp note: when we want to print multiple rows and column we have to use nested for loop.

Outer loop represents rows

Inner loop represents columns

Different examples to print the o/p in different formats


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

    /*for(int i=1;i<=10;i++)
      System.out.println(i);*/

    /*for(int i=1;i<=10;i++)
      System.out.print(i);*/

     /* for(int i=10;i<=100;i+=10)
       System.out.print(i);*/


     /*for(int i=1;i<=10;i++)
        System.out.print(1);*/

     /*for(int i=2;i<=100;i+=2)
       System.out.print(i);*/


     /*for(int i=1;i<=10;i++)
      {
       for(int j=1;j<=10;j++)
       {
        System.out.print(i);
       }
        System.out.println();
      }*/
```

```java
/* for(int i=1;i<=10;i++)
 {
   for(int j=1;j<=10;j++)
   {
     System.out.print(j);
    }
     System.out.println();
  }*/


  /*for(int i=1;i<=10;i++)
    System.out.println("5 x " + i + "=" + (i *5));*/


/*for(int i=1;i<=10;i++)
{
 for(int j=1;j<=10;j++)
    System.out.println(i+ " x " + j + "=" + (i *j));

}*/

/* int num=1;

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

   for(int j=1;j<=10;j++)
   {
    System.out.print(num++);
   }
   System.out.println();

}*/

/* int val=10;

 for(int i=1;i<=3;i++)
 {
   for(int j=1;j<=3;j++)
   {
    System.out.print(val);
    val+=10;
   }
    System.out.println();
}*/

/*int num=1;
for(int i=1;i<=4;i++)
{
  for(int j=1;j<=i;j++)
  {
   System.out.print(num++);
  }
   System.out.println();
}*/
```

```
/*for(int i=1;i<=5;i++)
{
  for(int j=1;j<=i;j++)
   {
      System.out.print(i);
   }

   System.out.println();

} */


/*  for(int i=1;i<=5;i++)
 {
   for(int j=1;j<=i;j++)
    {
       System.out.print(j);
    }

    System.out.println();

 }     */

/* for(int i=1;i<=5;i++)
 {
   for(int j=1;j<=5;j++)
   {
    if(i==j)
     System.out.print(1);
    else
     System.out.print(" ");
    }
   System.out.println();

 }*/

/* for(int i=1;i<=5;i++)
 {
   for(int j=1;j<=5;j++)
   {
    if(i==j)
     System.out.print(j);
    else
     System.out.print(" ");
    }
   System.out.println();

 }*/

/*for(int i=1;i<=5;i++)
 {
   for(int j=1;j<=5;j++)
   {
     if((i==j)||(i+j==6))
       System.out.print(j);
```

```
        else
          System.out.print(" " );
      }
    System.out.println();

  }*/


    for(int i=1;i<=10;i++)
    {
      for(int j=1;j<=10;j++)
      {
        if(i==1||i==10)
          System.out.print("*");
        else
            System.out.print(" ");

      }
    System.out.println();

  }}}




class ForDemo2
{
  public static void main(String...args)
  {


  /* for(int i=1;i<=10;i++)
    {
      for(int j=1;j<=10;j++)
      {
      if(i==1||i==10)
        System.out.print("-");
      else
          System.out.print(" ");
    }
    System.out.println();
  }*/

  /* for(int i=1;i<=10;i++)
    {
    if(i!=5)
    {
      for(int j=1;j<=10;j++)
      {
      if(j==1||j==10)
        System.out.print("|");
      else
        System.out.print(" ");
    }
    }
    else
    {
```

```java
    for(int k=1;k<=10;k++)
      System.out.print("-");

  }

    System.out.println();
  }*/

  /*int num=1;

  for(int i=1;i<=4;i++)
  {
   for(int k=1;k<=4-i;k++)
     System.out.print(" ");

   for(int j=1;j<=i;j++)
   {
      System.out.print(num++);
   }
    System.out.println();
  }*/


  /* int num=1;

  for(int i=1;i<=4;i++)
  {
   for(int k=1;k<=4-i;k++)
     System.out.print(" ");

   for(int j=1;j<=i;j++)
   {
      System.out.print(num++ + " ");
   }
    System.out.println();
  }*/

  /*int num=10;

  for(int i=1;i<=4;i++)
  {
   for(int k=1;k<=4-i;k++)
     System.out.print(" ");

   for(int j=1;j<=i;j++)
   {
      System.out.print(num-- + " ");
   }
    System.out.println();
  }*/

  /*int num=10;

  for(int i=4;i>=1;i--)
  {

   for(int k=0;k<=4-i;k++)
     System.out.print(" ");
```

```
    for(int j=1;j<=i;j++)
    {
       System.out.print(num-- + " " );
    }
    System.out.println();
  }*/

  //print 1-5 diagonally with single for loop
  /*String s="";
  for(int i=1;i<=5;i++)
  {
    System.out.println(s+i);
    s=s+" ";
  }*/

  //print 1-5 diagonally with out any loop
  System.out.print("1\n 2\n  3\n   4\n    5"); }
}
```

<mark>Sep 26th 2016</mark>

Implementing Jump statements

1. Break
2. Continue
3. Return

Break statement: helps to determine the loop and transfer the control in decisional and iteration statements

Class Demo28

{

      PSVM()

      {

            Int i=1;

            While (true)

            {

            SOP(i++);

            If (i==100)// prints 1 – 99

            Break;

            }

      }

}

In the above code control breaks out of the loop when decision "if" statement becomes true.

Break can be used in all loops.

Labelled break statement: It is used only in nested, decisional and iteration statements. A label in a labelled break statement I similar to an identifier which is used before the loop. The identifier is always followed by a colon (:).

Class Demo29

{

      PSVM()

      {

      Demo:// label for break

      For (int i=o; i<=3; i++)

      {

            For (int j=2; j<=4; j++)

            {

                  If (i==j)

                  Break demo;// labelled break

                  SOP(i+ " "+ j + "");

            }

            SOP();

      }

      }

}

If we use only break statement – o/p – exit inner loop

02	03	04
12	13	14
	23	24
32		34

If we use labelled break statement – o/p – exit outer loop

| 02 | 03 | 04 |
| 12 | 13 | 14 |

Continue Statement: In unlabelled statements, the continue statement causes the current iteration of a loop to stop and continue with the next iteration. However, in labelled statements the continue statements stops the current iteration of outer loop and continue with next iteration.

```
Public class Demo30
{
      PSVM()
      {
            For (int j=0; j<10; j++)
            {
                  If (j==6)
                  {
                  Continue;
                  }
            SOP(j);
            }
      }
}

o/p – 0 2 4 8
```

Using labelled continue

```
Public class Demo31
{
      PSVM()
      {
            Int j,k;
            Demo:
            For (j=0; j<5; j++)
            {
                  For (k=2;k<5;k++)
                  {
                        If (j==6)
```

```
                                        {
                                        Continue demo;
                                        }
                               SOP("j= " + j + "k= " + k);
                               }
                    }
          }
}
```

o/p -

02	03	04
12	13	14

32	
42	43

Return statement: cause the control of execution to return the caller of a method. While using the return statement, you must be careful about the return type of the method. If the method is of the void type you should not use return or use just return.

27th Sep 2016   **OOPS concept**

Class Mirchi

{

       Static int I;

           Int j;

           PSVM()

           {

           }

       Static void print()

       {

       }

       Void disp()

       {

       }

       Mirchi m1 = new Mirchi();

       Mirchi m2 = new Mirchi();

Mirchi m3 = new Mirchi();

}

- Designing a class means defining a class in simple words. Class contains variables which hold values. Methods which perform actions and (also can contain inner classes). Together we call it as Members
- You can declare the variable and methods in a class with the keyword static. Then these members will become static members or class members
- Variables and methods declared in a class without static keyword is called as Non static members or instance members or object members
- Static members belongs to class and instance members belongs to object
- Static members are only one per class whereas instance members each object has its own copy of non-static members or instance members

- In the above example, when we carte 3 objects Mirchi class, the sequence is as follows
- Execution using java Mirchi: where the class file is being executed by JVM
- When the class is loaded in the memory all the static members gets loaded
- Instance members or non-static members gets loaded only when the object is created. In this case, when the first object is created using Mirchi m1 = new Mirchi ();, then one copy of Non – static members are loaded in the object, then assigned to reference variables. Samething applies to m2 and m3 also
- Variables declared within a method or a block is called local variable
- Variable declared within a class and outside any method is called class level variable
- Static variable is also called class variable
- Non static variable also called as instance or object variable

Class VariableDemo

{

Static int a = 100;// static variable/ class variable

Int b = 200; // Non- static variable/ instance variable

PSVM()

{

Int i=0; // local variable

{ // statement block

Int j = 20; // local variable

}

}

Int add (int n1, int n2)// n1 and n2 are method parameters

```
                    {

                            Return n1+n2;

                    }

        }




Class Demo33

{

    Static int I = 10;

    PSVM()

    {

            SOP("i= " + i);

            I=20;

            SOP("i= " + i);

            I = 30;

            SOP("i= " + i);

            Int I = 400;

            SOP("i= " + i);

            I = 500;

            SOP("i= " + i);

            SOP("i= " + Demo33.i);

            Demo33.i = 40;

            SOP("i= " + Demo33.i);

            SOP("-------");

            { // statement block for organizing code

                    Int j = 100; // local variable

                    SOP("i= " + i);// local
```

```
                    SOP("j= " + j); // block level scope

                    SOP("i= " + Demo33.i); // static I which is class level

            }

            SOP(j); // out of scope and not accessible. Because 'j' is limited to only statement
    block

        }

    }
```

- If local variable name is not same as class variable, then you can directly access the class variable (static)
- Any executable statement should be part of a method and, in class we can only declare and initialize variables
- Whenever we use class variable (static) or method (static) always use Classname.membername. don't use directly

Interview question – what is the difference b/n static members and non-static/ instance members

Note 1: To access static members we use "class name and ● separator along member name"

Note2: To access non-static members (variables/ methods) we use "class name and ● separator along member name"

```
// using static variable an method
Class Demo34

{

        Static int I = 20;

        PSVM()

        {

                Demo34.test();

        }
```

```
        Static void test()

        {

                SOP("i= "+ Demo34.i);

        }

}
```

o/p – i=20

28th Sep 2016

Class Orange

{

      Int i=100;

      PSVM()

      {

            Orange o1 = new Orange (); // Orange object is created and o1 is referring to it

            O1.print(); // calling method print() of the object

            O1.i = 200; // changing print the value to 200

            O1.print();

            New Orange (). Print(); // one more new object is created and calling print () method. This will have its own copy of values which are initialized first
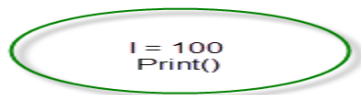
            New Orange (). I = 200; // one more new object is created and changing the value to 200. This will have its own copy of values which are initialized first
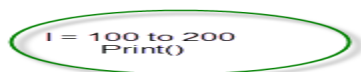
            New Orange (). Print(); // one more new object is created and calling print () method. This will have its own copy of values which are initialized first

I = 100
Print()

        }
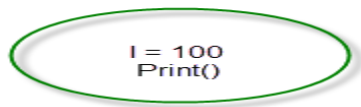
        Void print()

        {

                SOP("I= " +i);

        }

}

o/p – 100, 200, 100, 100

Note: **Static members can be accessed from anywhere (from static methods or non – static methods) directly (without using class name.**

Here members means variables and/ or methods

Class A

{

        Static int I = 10;

        Static void print ();

        {

                SOP(i);

        }

        PSVM()

        {

                Print();

        }

        Void disp()

        {

Print();

        }

}
Note 2: Non static members can be called from non-static method directly.

But non static members cannot be called from static methods directly. But we can call the members by creating an object and using reference variable dot (.)

Class B

{

        int I = 100; // non static variable

        void print (); // non static method

        {

                SOP(i);

        }

        PSVM()

        {

                B b1 = new B();

                B1.Print();

                SOP(b1.i);

        }

        Void disp()// non static method

        {

        Print(); // calling non static method

        }

}

Class A

{

        Int I;

}

Class Demo35

{

        PSVM()

        {

                A a1 = new a();

                a1.i = 10;

```
                A a2 = new A();

                a2.i = 20;

        }

}
```

Memory location
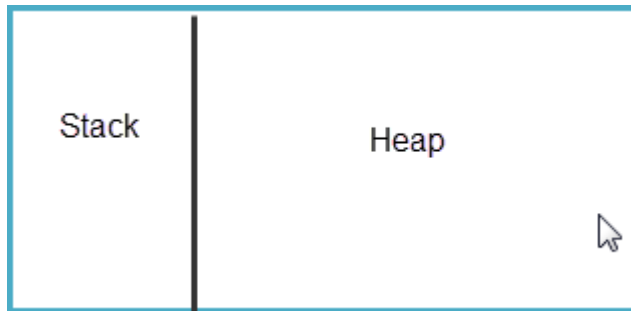
Stack – stack is for execution purpose

Program to link the necessary libraries and load classes

Heap – Heap is to store objects and used for storage purpose

A a1 = new A();

a1 points to class A object

Until we create object, we cannot access non static members

a1.i=10;
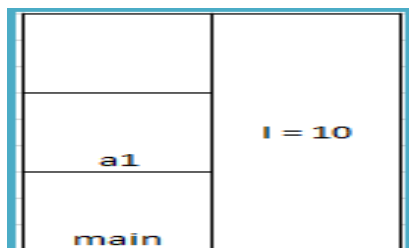
A a2 = new A();

a2.i = 20;

When main method is done, java (JVM) calls garbage collector before terminating to clean the memory

Note: Static members are also called as class members and non-static members are also called as reference or instance or object members

<mark>**VV. imp – class level variables are called as fields**</mark>

<mark>**29th Sep 2016**</mark>

Static block: It is a block (code) with keyword <u>static</u> which gets executed when the class is loaded (before the main method) and is used for initializing the static variables

<mark>VV imp – IF there is static block, java first executes the static block then the main method</mark>

Static
{

-----   Static block
}

You can have multiple blocks. Always it executes in top to bottom sequential order

Class Demo36
{
          Static int I = 10;

}
Static
{
        I=100;
        SOP(3);
}
PSVM()
{
        SOP(i);

```
}
Static
{
        I=200;
        SOP(1);
}
Static
{
        I=300;
        SOP(2);
}
}
```

o/p –
3 – First static block value
1 - Second static block value
2 - Third static block value
300 – Updated latest value of I in static blocks

```
Class Mango
{
        Static int I = 10;

        Static
        {
                SOP(i);
                Print(); // calling print method
                I = 555;
        }

        PSVM()
        {
                SOP(mango.i);
                SOP(kachamango.j);
        }
        Static
        {
                I = 6666;
        }
        Static void print()
        {
                SOP("Hello")
        }
        Class kachamango()
        {
                Static int j=100;
                Static
                {
                        J=1234;
```

```
                    }               }        }
```

When we execute the command, java Mango
1. First the class is loaded in the memory
2. During that, first the static variables are loaded and initialized to default or assigned values
3. Next the static methods are loaded
4. Next the static blocks are executed
5. Next the main method gets executed

Note: In main method if we access members of any other class like in the above example first the class gets loaded (Kachamango). During that
1. First the static variables are loaded and initialized to default or assigned values
2. Next the static methods are loaded
3. Next the static blocks are executed
4. If that class has main method it will not be executed (Imp)
5. Loading and executing happens only if they are present (variables, methods or static blocks (Imp)
6. Suppose if we wanted to access non static members then we would have to create object of that class during which the same sequence would happen (loading of class, loading of value and initialization, loading methods, execution of static blocks and creating objects

Why final variables are preferably static

We don't need to create final variables for each object separately, since the value do not change.

// Demo for initializing a blank final

Class BlankFinalDemo

{

        Static final double PI; // blank final

        Static

        {

                PI = 3.14// initialized in the static block

        }

        PSVM()

        {

                SOP(PI);

        }

}

Note: We can declare a final variable and initialize in static block

Blank final – a final variable which is not initialized to a value in the same line where we are declaring is called blank final

Non Static block:

It is a block (code) without keyword static and which gest executed every time an object of class containing non static blocks gets created

{

------------------   ┤   Non static block

}

You can have multiple non static blocks. Always it executes in top to bottom sequential order

```
Class Demo39
{
        Int I, j;
        PSVM()
        {
                Demo39 d1 = new Demo39();
                SOP(d1.i);
                SOP(d1.j);
                Demo39 d2 = new Demo39();
                SOP(d2.i);
                SOP(d2.j);
                SOP(new Demo39()); // executes IIB and prints address
                SOP(new Demo39().i); // new object
                SOP(new Demo39().j); // new object
                D2.i = 123;
                D2.j = 456;
                SOP(d1.i);
                SOP(d1.j);
                SOP(d2.i);
```

```
        SOP(d2.j);

    }

    { // non static block

        SOP("IIB1");

        I = 20: j=30;

    }

    { // non static block

        SOP("IIB1");

        I = 200: j=300;

        {i=222; j=333;}

    }

}
```

o/p –

IIB1	IIB1	IIB1	IIB1	IIB1	222	123
IIB2	IIB2	IIB2	IIB2	IIB2	333	456
222	222	Demo3944564654654	222	333		
333	333					

A class contains two types of members

1. Static members
2. Non static members

Static	Non static
static members are created using static keyword on variable and method	non static members are created without using static keyword on variable and method
static members belongs to class	Non static members belongs to object
static members are one per class	each object has its own copy of non-static members
static members are also called as class members	Non static members are also called as instance/reference/object members
All static members can be accessed by using dot separator on the class name Ex: classname.variablename	Non static members can be accessed by using dot separator with the reference variable name Ex: object/reference variablename. Variablename
Java provides separate block to initialize static variables of the program. This block is known as static initialization block (SIB)	We have block to initialize non static variables. This is known as instance initialization block (IIB).

Before executing main method, IIB will be executed first when the class is loaded *SIB is executed only once when the class is loaded*	*IIB gets executed when an object of the class is created* *IIB is called (multiple times) each time an object increased*
static variables are initialized when the class is loaded	*non static variables are initialized when the object is created*
We can have any no of static initialization blocks	*We can have any no of non-static/ instance initialization blocks*
The execution of block is sequential (top to bottom)	*The execution of block is sequential (top to bottom)*
These static variables will have the recent values initialized by last static initialization block	*These non-static variables will have the recent values initialized by last instance initialization block*

30th Sep 2016

- A java file can contain more than 1 class (definition blocks)
- If a java file has 1 public class, then that class name should be used as filename.java
- A java file cannot have more than 1 public class
- If the file contains only non-public classes, then any class name can be used to name the file
- Whenever a java file which contains multiple classes is compiled, separated class files will be created for each class
- Compiler can compile single or multiple java files at a time. But JVM executes only one class file, that too which has main method
- We can pass runtime arguments while executing class

Ex: java Demo1 I Love Java 123

Where Demo1 is the class with main method, rest are String arguments passed as a string array to the args variable in the main method

Note: generally the java file name will be of the class which has main method. But you can have the java file name as one of the class names (or any name not mentioned as class or only '.java' also (not compulsory))

Class Orange

{

       Void test()

       {

```
                SOP("I am Orange");

        }

}

Class A

{

        PSVM()

        {

                Orange o1= new Orange();

                O1.test();// 1st way to call method

                New orange().test();// 2nd way to call method

        }

}
```

You can save it as Orange.java or A.java--------------------------

```
Class A

{

        Static

        {

                SOP("SIB of A class);

        }

        {

                SOP("IIB of A class);

        }

}


Class Demo40

{

        Static

        {

                SOP("SIB of Demo40 class");

        }
```

```
        PSVM()

        {

                SOP("main starts");

                A a1= new A();

                A a2 = new A();

                SOP("main ends");

        }

}
```

Imp: Here when a1 is executed it will go to class A file and execute static block first. And non-static next

But when a2 is executed it will go to class A file again and execute only non-static block because static block will get executed only once when the first time class is loaded

Write a program to implements a non-static method which initializes variable with passed value and also returns the same value

```
Class B

{

        Int I;

        Int j;

        Int getival(int a)

        {

                I=a;

                Return I;

        }

        Int getjval(int b)

        {

                j=b;

                Return j;

        }
```

```
}
Class Demo41
{
        PSVM()
        {
                B b1 = new B1();

                SOP(b1.i);

                SOP(b1.getival(555));

                Int k = b1.getjval(666);

                SOP(k);
        }
}
```

Understanding of variables
-        Variable can be classified as primitive variables and reference variables
-        These variable declared within a method or a block is called local variables. These variables declared within a class is called class level variables.
-        In class level scope you can make them as static or non static.
-        If you add static keyword for declaration then these variable become static variables .If you dont use static keyword these variable become non static variables/instance variables.
-        You cannot make static or non static in local scope.
-        Local variable decalred in one method or a block cannot be accessed in another method or a block

```
class Apple{

}
class Orange{

    int i,j;
    String s;
    Apple a;


    static int x,y;
    static String s1;
    static Apple a1;
```

```java
/*  Orange() // this is constructor
 {
    i=100;j=200;
    s="Hello";
    a=new Apple();
 }*/

 {//we can use non static block or constructor to
  //initialize non static variables

    i=100;j=200;
    s="Hello";
    a=new Apple();
 }




 static{
  x=222;y=333;
  s1="Bangalore";
  a1=new Apple();

 }

public static void main(String...arg)
 {
    Orange o=new Orange();
    System.out.println(o.i);
    System.out.println(o.j);
    System.out.println(o.s);
    System.out.println(o.a);
    System.out.println("----------------");

    Orange o1=new Orange();
    System.out.println(o1.i);
    System.out.println(o1.j);
    System.out.println(o1.s);
    System.out.println(o1.a);
    System.out.println("----------------");

    System.out.println(Orange.x);
    System.out.println(Orange.y);
    System.out.println(Orange.s1);
    System.out.println(Orange.a1);
 }

}
```

1st Oct 2016

Constructors: these are definition blocks in java which is used to build or construct an object of a class

Every class in java should contain at least one constructor. If not compiler writes constructor in the class which is known as default constructor

B b1 = new B();

Here new keyword calls the constructor. Constructor creates the object. B1 is the reference variable of type B and the B class object is assigned to b1. If we print b1 (ref var), we get the address of the object to which it is referring. You cannot create an object without a class (new is a keyword).

Constructor is used to

- Create objects
- Initialize non static variables

Rules for defining a constructor as per jdk 1.7

- The constructor name should be same as class name
- It should not have return type
- It should not contain non access (specifies/ modifiers): final, static, abstract, synchronized
- It can have any of the access modifiers: private, public, protected and default (here default is not a keyword. It means without any)
- It can have parameters
- It can have throws clause i.e. we can throw exception from constructor
- It can contain all java legal statements except return statement. i.e. we cannot have return in constructor

```
Class c
{
        Int I; int j;
        C();
        {
                I=10;
                J=20;
        }
}
```

Class Demo43

{

    PSVM()

    {

        C c1 = new C();

        SOP("I= " + c1.i);

        SOP("j= " + c1.j);

    }

}

o/p –

i=10;

j = 20;

- Constructor does not have a return type. And we can have a method with same name as class

Ex:

    A() // constructor (do not have return type before A)

    {

    }

    Void A() // method (which has return type void)

    {

    }

▪▪

Class A

{

    A()

    {

        SOP(" constructor of class A");

    }

    Static

    {

        SOP("SIB of class A");

    }

```
        {

                SOP("IIB of class A");

        }

}

Class Demo44

{

        Static

        {

                SOP("SIB of class Demo44");

        }

        {

                SOP("IIB of class Demo44");

        }

        PSVM()

        {

                SOP("main starts");

                A a1 = new A();

                A a2 = new A();

                SOP(" main ends");

                New Demo44();

        }

}
```

First static block then IIB & then constructor part

Actually constructor call the IIB's then completes itself

```
Class D
{
        Int I,j;
        D (int a, int b)
        {
                SOP("inside class D constructor");
                I=a;
                J=b;
        }
}
Class Demo45
{
        PSVM()
        {
                D d1 = new D(100,200);
                SOP(d1.i);
                SOP(d2.j);


                D d2 = new D(300,400)
                {
                        SOP(d2.i);
                        SOP(d2.j);
                }
        }
}
```

3rd Oct 2016

Class C

{

      Int I;

      C (int a)

      {

            SOP("constructor of class C");

            I=a;

      }

}

Class Demo45

{

      PSVM()

      {

            C c1 = new C(100);

            SOP("i= " + c1.i);

            C c2 = new C(200);

            SOP("i= " + c2.i);

      }

}

o/p –

Constructor of class C

I = 100

Constructor of class C

I = 200

Note: whenever you write any constructor, the compiler will not create any constructor. If no constructor is created by user then compiler creates a default constructor

Ex: If we have created a constructor with single argument then we should use that constructor with single argument to construct the object. If we want constructor with No argument also, like what compiler does, then we should only create a NO argument constructor

Below is example:

```
Class D
{
        Int I;
D(int a) // int arg constructor
{
        I=a;
}
D()// NO argument constructor
{
I=123;
}
}
Class Demo46
{
        PSVM()
        {
                D d1 = new D(100); // calling int arg constructor
                SOP(d1.i);
                D d2 = new D();// calling NO arg constructor
                SOP(d2.i);
                D d3 = new D(200); // calling int arg constructor
                SOP(d3.i);
                D d4 = new D();// calling NO arg constructor
                SOP(d4.i);
                D2.i = 567;
                SOP(d2.i);
        }
}
```

                    _____******_____

Demo for 2 single arguments constructor like int and long, byte etc..

```
Class F
{
        Int I;
        Long L;
        Byte b;
        F (int a)
        {
                I=a;
        }
        F ()
        {
                I=123;
        }
        F (long L)
        {
                L=b;
        }
        F (byte c)
        {
                b=c;
        }
}
Class Demo 47
{
        PSVM()
        {
                F f1 = new F();
                SOP(f1.i);
                SOP(f1.L);
                SOP(f1.b);
```

```
            F f2 = new F(100);

            SOP(f2.i);

            SOP(f2.L);

            SOP(f2.b);

            F f3 = new F(200L);

            SOP(f3.i);

            SOP(f3.L);

            SOP(f3.b);

            F f4 = new F((byte)100);

            SOP(f4.i);

            SOP(f4.L);

            SOP(f4.b);

        }

}
```

Note:
- If we have int constructor and also we have constructor with byte argument or short argument we have to explicitly cast it ..like

    ((byte)100) add (byte) at left side within parenthesis
    ((short)200) add (byte) at left side within parenthesis

- For long argument we can pass it like 100L adding L at right side
- If we have double and float, compiler will take it as double by default, to make it into float. We have to add 'f' like '100.4f' (adding f at right side)

What is constructor overloading?
- When we have more than one constructor for a class with different signature is called constructor overloading

Note: different signature means, no of arguments should be different or types of arguments should be different or sequence of arguments should be different

```
Class G
{
        Int I,j;
        Long l;
        String a,b;
        G(int u, int v)
        {
                I=u;
                J=v;
        }
        G(int x, long y)
        {
                I=x;
                L=y;
        }
        G(String S, int p)
        {
                a=s;
                i=p;
        }
        G(int p, String s)
        {
                a=s;
                i=p;
        }

}

Class Demo48
{
        PSVM()
        {
        G g1 = new G(100,200); // calling int and int constructor
        SOP(g1.i);
        SOP(g1.j);
        SOP("----");

        G g2 = new G("Star", 20); // calling String and int constructor
        SOP(g2.i);
        SOP(g2.j);
        SOP(g2.a);
        SOP(g2.b);
        SOP(g2.l);

        G g3 = new G(30, "Super"); //// calling int and String constructor
        SOP(g3.i);
        SOP(g3.j);
        SOP(g3.a);
        SOP(g3.b);
        SOP(g3.l);          }           }
```

4th Oct 2016   Inheritance

Inheriting all the members of parent class to the child class other than private members is known as inheritance. The parent class is called a super class whereas the child class is known as subclass

- A sub class inherits all the members (both static and non-static) (fields, methods and nested classes) from its super class except private members. **Constructors are not members, so they are not inherited by subclasses, but the constructor of the super class can be invoked from the subclass**
- The inheritance can be achieved by using keyword **<u>extends.</u>** The subclass should extend the superclass
- With the reference of subclass object we can access both the inherited member as well as subclass member
- Multilevel inheritance are supported in java
- Multiple inheritance are not allowed in java through classes

Class A

{

        ------------------→ Super/ parent class

}

Class B extends A

{

        -------------------------------→sub/ child class

}

- Multiple classes can inherit same class

Class C extends A

{

}

Class D extends A

{

}

Multilevel inheritance

Class A

{

}

Class B extends A

{

}

 Class C extends B

{

}

Multiple inheritance is not possible

Class extends A, B

{

}

Above one is not possible

****

Class A

{

      Int i=100;

      Static int j = 200;

      Private int k=300; // does not get inherited, because it is private

      Void print()

      {

            SOP("i= " + i);

      }

      Static void disp()

      {

            SOP("j= " + j);

      }

}

```
Class B extends A
{
}
Class InheritDemo
{
        PSVM()
        {
                B b1 = new B();
                B1.print();
                SOP(b1.i);
                SOP("----");
                B.disp();
                SOP(B.j);
                SOP("------");
                B1.disp();// compiler will replace b1 with class B.
                SOP(b1.i); // compiler will replace b1 with class B
                // SOP(b1.k); // private members are not inherited
        }
}
```

Note: 1 – we can access static members through a reference variable. Compiler replaces the reference variable with its type like

Compiler does the follwoing

b1.j – B.j

2 – Non static members cannot be accessed through a class name

Question – What will the compiler do?

```
Class X
{
}
Class Y extends X
{
}
```

Answer: Compiler does the following

Class X

{

      X()

      {   Compiler will put this default constructor

      }

}

Class Y extends X

{

      Y()

      {

            Super();   Compiler will put this default super () class

      }

}

Note: When compiler creates a constructor its capacity is only

1 – To create a default constructor i.e. constructor without argument

2 – To put "Super" without any arguments in the constructor

Class X

{

      X(String s)

      {

      }

}

Class Y extends X

{

      Y()

      {

            Super("Fellow");

      }

}

Class Demo49

{

    PSVM()

    {

        X x1 = new X("Hello");

        Y y1 = new Y();

    }

}

Notes: Super class has user defined constructor with 1 or more arguments, then we have to create the subclass constructor compulsorily and call super (argument) as expected by super class constructor

Class G

{

}

Class H extends G

{

    H (int a, int b, int c);

    {

        Super (); // this will be put by compiler by default. Since super class does not have a constructor, it will first create a default constructor in the super class and in sub class uses super ();

    }

}

Class Demo 50

{

    PSVM()

    {

        G g1 = new G();

        H h1 = new H(100, 200, 300);

    }

}

Very Imp:

1 – We cannot have 2 variables with the same name – one static and another non-static

2 – We cannot have 2 variables with the same name – one local and another method parameter

3 – We can have 2 variables with the same name – one static and another local

4 – We can have 2 variables with the same name – one non-static and another local

5 – We cannot have 2 variables with the same name – one local and another within static block

```java
class C{

     int i,j;

     C(int i,int j)
     {
        this.i=i;
        this.j=j;
     }


     void print()
     {
        System.out.println("i="+i);
        System.out.println("j="+j);
     }
}
class D extends C{

      int a,b;

    D(int a,int b)
    {
       super(a,b);
       this.a=a;
       this.b=b;

    }
     void disp()
     {
        System.out.println("a="+a);
        System.out.println("b="+b);
     }

}

class E extends D{

    E()
    {
        super(555,666);
    }

    E(int x,int y)
    {
       super(x,y);
    }

}
class Demo51{

    public static void main(String...args)
    {
       C c1=new C(100,200);
        c1.print();
```

```
        System.out.println("---------");
        D d1=new D(300,400);
         d1.disp();
         d1.print();
System.out.println("---------");
      E e1=new E();
         e1.disp();
         e1.print();
System.out.println("---------");
        E e2=new E(500,600);
         e2.disp();
         e2.print();

     }

}
```

Interview question: what is the difference b/n parameter and argument?

Parameters refers to the definition of a method or constructor or exception to which we intend to pass values or variables

Ex:

Static int add(int n1, int n2)

{

}

Here n1 and n2 are parameters

Arguments refers to values or variables which we actually pass when calling the method or a constructor

Ex: add (100,200)

Here 100 and 200 are arguments

• •

<mark>5th Oct 2016</mark>

Class A

{

       Int I;

       A()

       {

             I = 123;

       }

       A(int i)

       {

             this.i = I; we will this when two variable name are same to defer the scope

       }

```
        A(String s)
        {
                SOP(s);
        }
}
Class B extends A
{
        B();
        {
                Super();
        }
        B(int x);
        {
                Super(x);
        }
        B(int a, int b, String s1, String s2, char ch)
        {
                Super(a);
        }
}
Class Demo52
{
        PSVM()
        {
                A a1 = new A (); // no argument constructor called
                New A (10); // int argument constructor called
                A a2 = new A ("Hello"); // String argument constructor called
                B b1 = new B ();
                B b2 = new B (100);
                B b3 = new B (300, 400, "Start", "open", 'a');
                }        }
```

1 – If a class contains more than one constructor we can call any one of the constructor to create an object

2 – If super class contains more than one constructor we can call any one of the super class constructor available in the super class from subclass

3 – When we use 'Super ()' to call super class constructor it should be the first statements in the subclass constructor

Why do we need to create our own constructor?

-   If we need to create an object of a class based on specific criteria then we need to create our own constructors
    Ex: If we need to create object of student class which takes name and student id or name and mail id then create our own constructor
    Class A
    {
        A(int a)
        {
        }
        A()
        {
        }
    }
    Class B extends A
    {
        PSVM()
        {
        }
    }

    // compiler automatically put B () constructor and add super (). This will work as the super class has a no argument constructor.

    Write a count the objects created for a class

```
Class Apple

{

        Static int cntr = 0;
        Apple()
        {
                Cntr++;
        }

}

Class objectcount

{

        PSVM ()
```

```
Apple a1 = new Apple ();

Apple a2 = new Apple();

        New Apple();

Apple a3 = new Apple();

SOP(Apple. Cntr);

}

}
```

Write a program to count total objects created (or) program to show subclass constructor calls super class constructor

```
Class Apple
{
    Static int cntr = 0;
    Apple()
    {
            Cntr++;
    }
}
Class OotyApple
{
    Static int cntr = 0;
    OotyApple()
    {
            Cntr++;
    }
}

Class objectcount

{

    PSVM ()

    {

    Apple a1 = new Apple ();

    Apple a2 = new Apple();

            New Apple();

    Apple a3 = new Apple();

    SOP(Apple. Cntr); // o/p is 4

    OotyApple a1 = new OotyApple ();

    OotyApple a2 = new OotyApple();
```

OotyApple o3 = New OotyApple();

OotyApple a4 = new OotyApple();

SOP(OotyApple.cntr); // o/p is 4

SOP(Apple. Cntr); // o/p is 8

}

}

6th Oct 2016

Notes:

- Every class we create in java will be sub class to the object. Object class is the super most class in java. If any class is not extending superclass, by default will always extend object class
- When subclass object is created, chaining constructors will be involved. The constructor of sub class calls constructor of the superclass. The constructor of the super class in turn calls constructor of its super class. This is known as chain of constructor.
- Java compiler makes an implicit call to the super class constructor using super() from subclass constructor
- In super class, if default/ no argument constructor is not accessible, then the constructor should be called explicitly using super statement along with argument(s) as expected from superclass
- The Super statement should be always the first statement in the constructor
- Using super statement we can call any of the superclass constructors
- Whenever the java compiler makes a call to the super class, when super() is not mentioned, it writes super() statement implicitly

Java supports two types of relationship
1. IS A relationship
2. HAS A relationship

The IS A relationship defines the inheritance of a superclass to the subclass where HAS A relationship defines composition of a class

ENCAPSULATION

Encapsulation is the packing of data and functions into a single component and hiding the internal implementation

It is achieved by hiding the data by making the variables private and showing the functionalities through getter and setter methods

Ex: setage

getage

For Boolean value use 'is'. Don't use get. It should be meaning full.

Isvalid

Iscopied

```
Assignment:

Create Student class with following attributes.
Student_ID, Student_Name, Student_PhoneNumber, Student_MailId, Student_Address,
Degree & YOP (year of passing).
•        Class should allow to create a student with
1.       Student_Id ,Student_Name & Student_PhoneNumber
2.       Student_Id,Student_Name & Student_MailId
•        Class should have methods to print every attribute values
•        The student class should provide an option to modify phoneno, MailId &
Address

•        Provide method to print details of the student


1.       Student Class
2.       Attributes-Id, StudentName,
StudentNumber,StudentMailId,StudentAddress,Degree & YOP
3.       Constructors
a)       StudentId,StudentName & StudentPhoneNumber
b)       StudentId,StudentName & StudentMailId
4.       Method to get each & every attribute values
5.       Modify StudentPhoneNo,MailId & Address
```

```java
class Student
{
        //attributes
    private   int studentId;
    private   String studentName=null;
    private   long studentPhoneNumber;
    private   String studentAddress=null;
    private   String studentMailId=null;
    private   String studentDegree=null;
    private   int YOP;
```

```java
//Constructors


    Student(int studentId,String studentName,long
studentPhoneNumber)
    {
        this.studentId=studentId;
       this.studentName=studentName;
       this.studentPhoneNumber=studentPhoneNumber;
    }

   Student(int studentId,String studentName,String StudentMailId)
   {
     this.studentId=studentId;
    this.studentName=studentName;
    this.studentMailId=studentMailId;
    }




   int getStudentId()
  {
    return studentId;
  }

 String getStudentName()
{
   return studentName;
}
long getStudentPhoneNumber()
{
       return studentPhoneNumber;
}

String getStudentMailId()
{
       return studentMailId;
}

String getStudentAddress()
{
   return studentAddress;
}
String getStudentDegree()
{
   return studentDegree;
}
int getStudentYOP()
{
      return YOP;
}

void setStudentPhoneNumber(long studentPhoneNumber)
{
   this.studentPhoneNumber=studentPhoneNumber;
}



void setStudentMailId(String studentMailId)
{

this.studentMailId=studentMailId;
}

void setStudentAddress(String studentAddress)
```

```java
{
   this.studentAddress=studentAddress;
}


void printStudentDetails()
{
      System.out.println("Student Id:" + studentId);
      System.out.println("Student Name:" + studentName);
       System.out.println("Student PhoneNo:" + studentPhoneNumber);
       System.out.println("Student MailId:" + studentMailId);
       System.out.println("Student Address:" + studentAddress);
       System.out.println("Student Degree:" + studentDegree);
       System.out.println("Student YOP:" + YOP);
    }
}




class  CreateStudent
{
    public static void main(String[] args)
    {
        Student s1=new Student(1,"YoyoHoneySingh",9876543211);



        Student s2=new Student(2,"AamirKhan","aamir@pk.com");



         System.out.println(s1.getStudentName());
         s1.setStudentPhoneNumber(80000000);

         System.out.println(s1.getStudentPhoneNumber());
         s2.printStudentDetails();
    }
}
```

7th Oct 2016   Packages

S package is a grouping (or folder structure) of related classes/ interfaces/ enumerations or annotation types providing access protection and name space management

This helps in –

- Grouping of related types like classes and/ or interfaces
- Avoiding name conflicts with classes created in same or other projects in the same company
- You can allow types within the package to have unrestricted access to one another. yet still restrict access for types outside the package

Public class A
{
    Int I;

    Public void print()
    {
    SOP("i= "+i);
    }

    PSVM()
    {
    SOP("class A of package com.qspiders.pack1");
    }
    }

For compilation (use hyphen (\) and folder path like below
Javac –d ..\bin com\qspiders\pack1\A.java

For execution (use dot (.) and folder path like below
Java com.qspiders.pack1.A

// separate java file B.java
Package com.qspiders.pack1;
Public class B
{
        PSVM()
        {
                SOP(" class B of package com.qspiders.pack1");
                A a1 = new A();// this is from another java file
                A1.print(); // calling print method from another java file
                B b1 = new B()// we just created and not using
        }
    }
    o/p - class B of package com.qspiders.pack1
        i=0

Access specifiers/ modifiers

The access specifies in java indicates visibility of the members of the class. Java supports following 4 specifier levels

Access modifiers/ specifiers (visibility) levels

1. Private
2. Default (without any keyword)
3. Protected
4. Public

Note: specifiers and sopecifier levels, both are different

When it comes specifiers, they are three

1. Private
2. Protected
3. Public

When it comes specifier levels, they are four

1. Private
2. Default (without any keyword)
3. Protected
4. Public

1. You can apply the access specifiers to classes, variables, methods & constructors
2. Please note below
   o Package declaration should be the first statement.
   o If you are importing any class from another package, then this should be the second - statement. You can import as many classes as you want from any other packages
   o Third should be class statement

Package com.qspiders.app1;

Class A

{

     Private int i=10;

     Void print()

     {

          SOP("i= " + i);

     }

```
        PSVM()

        {

                A a1 = new A();

                a1.print();

                a1.i=20;

                a1.print();

        }

}

I=10;

I=20;
```

- **Private members are accessible only with in the same class**

```
Package com.qspiders.app1;
Class Run1
{
        PSVM()
        {
                A a1 = NEW A();
                a1.print();
                a1.i=100; // CTE (compile time error. Because variable I is private to previous
program
                a1.print();
        }
}
```

                        *****

```
Package com.qspiders.app1;

Public class B

{

        Int i=100;

        Private B (); // private constructor

        {

        }

        Public void print ()

        {

                SOP("i= " + i);

        }
```

PSVM()

{

B b1 = new B();

B1.print();

}

}

I=100;

******

Package com.qspiders.app1;

Class Run2

{

PSVM()

{

B b1 = new B();// can't create object as constructor is private (CTE – compile time error)

b1.print();

}

}

o/p – CTE (compile time error)

Note:

- Private members have visibility with in the same class
- Private members cannot get access outside the class
- Private can be used for variable/method/constructor
- If constructor is declared as private an instance of the class cannot be created using private constructor in another class
- Class cannot be declared private or protected class can be public or default (without any keyword)

While developing any application, classes related to similar tasks/ operations are stored in a package. The java packages consists of classes which can be used in other packages

- The java package naming is always done with the company domain name (reverse URL)
- In one package duplicate classes are not allowed whereas same class name com in multiple packages
- Fully qualified class name consists of package name & the class name
- A class in one package can access class from other package by using either fully qualified class name or import statement
- When we write a class or an interface without a package, Java will consider it as default package
- Java lang package gets automatically imported (which contains String class)

8th Oct 2016

Assume that class A file is in com.qspiders.app1 folder

Assume that other class A and class B files are in com.qspiders.app2 folder

Below is the example to demonstrate accessing of class A

******

Package com.qspiders.app1; // this is app1 folder class A

Public class A

{

    Int i=100;

    Void print ();

    {

        SOP("i= " + i);

    }

}

Package com.qspiders.app2; // this is app2 folder class A

Class A

{

    Int j=200;

    Void disp()

    {

        SOP("j= " +j);

    }

}

Scenario 1: Accessing class A of same package where in class B is also in same package

```
Package com.qspidders.app2;

Class B

{

   PSVM()
   {
        A a1 = new A();
        A1.disp();
   }
}
```

Scenario 2: Accessing of class A of other package (folder) in class B
Solution a:

```
Package com.qspiders.app2;
Import com.qspiders.app1.A; // using import for class A
Class B
{
   PSVM()
   {
        A a1 = new A();
        A1.print();
   }
}
```

Solution b: using fully qualified class name (here we need not to use import because we aree using fully qualified class name)
```
Package com.qspiders.apppp2;
Class B
{
   PSVM();
   {
        Com.qspiders.app1.A a1 = new com.qspiders.app1.A();
        a1.pprint();
   }
}
```

Scenario 3: Accessing class A from same and different folder at a time simultaneously

Sol a:

Package com.qspiders.app2;

Class B

{

      PSVM()

      {

            com.qspiders.app1. A a1 = new com.qspiders.appp1.A();

            A1.print();

            A a2 = new A(); // here this is from same folder. Hence we need not to define the package name

            A2.disp();

      }

}

Sol b:

Package com.qspiders.app2;

Class B

{

      PSVM()

      {

            com.qspiders.app1. A a1 = new com.qspiders.appp1.A();

            A1.print();

            Com.qspiders.app2.A a2 = new Com.qspiders.app2A(); // here this is from same folder. Hence we need not to define the package name. but to be more clear we defined the package name

            A2.disp();

      }

}

** Demo for using default access specifier

Package com.qspiders.pack1;

Class C

{

       Int i=100;

       Void print();

       {

              SOP("i= " +I);

       }

}

Class Run1

{

       PSVM()

       {

              C c1 = new C();

              C1.print();

              C1.i = 200;

              C1.print();

       }

}

If we are writing both classes in same java file, save the above code with public classname.java

Ex: c.java (if any of the class is public)

Else if we are writing both in separate java files then write the "package com.qspiders.pack1" in the first line

Package com.qspiders.pack2;

Import com.qspiders.pack1.c // default class cannot be imported. For C class there is no public

 Class Run2

{

     PSVM()

     {

         C c1 = new C();

         C1.print();   all the members should be public to access. These

         C1.i = 200;   are all default in class C.

         C1.print();

     }

}

Assume that class A and class Run1 files are there in com.qspiders.pack1 folder

Assume that Run2 file is in com.qspiders.pack2 folder

- If class is defined as default then these will have visibility inside package
- If we want to access classes in another package, we should use public class and members should also be public in order to access them
- Constructor access depends on the class access specifier
  If the class is public, constructor that compiler create will be public
  If we write our own constructor, then we have to give public access to create the object outside the package

```
//demo for accessing class from another package
//here class should not only be public, if we have user defined constructor then it should also be
//public and if we want to access the members ,that should also be public


package com.qspiders.pack1;
public class  D
{
    public  int  j=100;

    public D()
    {
        System.out.println("running constructor");
    }

     public void print()
     {
```

```
    System.out.println("j="+j);
   }
}
```

Note:Save it as public classname.java

```
package com.qspiders.pack2;
import com.qspiders.pack1.D;

class Run3
{
  public static void main(String[] args)
  {
    D d1=new D();
   d1.print();
   d1.j=200;
   d1.print();
  }
}
```

•default members are declared without a keyword and has a visibility within the class and same package.
•The public members have a visibility within the package as well as outside the package (another package)
•Whenever compiler creates its own constructor then then access specifier of the constructor will always will go with the class access specifier
•If the class is public and access specifier is default (for constructor) we cannot create an instance of the class outside the package.

public class   constructor (default access specifier)

```
public class  D
{
    D()
    {
       System.out.println("running constructor");
    }
    public int j=100;
}
```

//you can access the class in another package with //static members (which is public)
//you cannot create object in another package

//protected members demo

```java
package com.qspiders.pack1;
public class F
{    public  int  j=0;
    protected int  i=100;

        protected void print()
        {
                System.out.println("i="+i);
        }
}


class Run3
{

    public static void main(String[] args)
        {
                    F f1=new F();
                    f1.print();
                    f1.i=200;
                    f1.print();
        }
}
```

// Save this as F.java
//Compile & run , the above program Run3.class will work //because protected members can be accessed from Same //package.It behaves like default

```java
package com.qspiders.pack2;
import com.qspiders.pack1.F;
class  Run4
{
        public static void main(String[] args)
        {
        F f1=new F();
    f1.print(); // protected members can't be
         //accessed
    f1.i=200;  // without inheritance from other
        //package
         f1.print();
        }
}
```

//to correctly access protected members in another package

 package com.qspiders.pack2;
 import com.qspiders.pack1.F;

class Run5 extends F
{
   public static void main(String[] args)
  {
     Run5 r1=new Run5();
     r1.print();//print() and i are
            //protected members of
     r1.i=200; // class F, can be accessed only
            //through
     r1.print(); //inheritance, becomes
             //private to this class
 }

}

• 	The protected members have the visibility outside the class and also outside the package. But protected members can be accessed by other package only through inheritance.
• 	After inheriting protected members behaves as a private member to the inherited class

12ᵗʰ Oct 2016

Method overloading:

Developing multiple methods with same name but different method signatures is known as Method overloading. The signature should change either in terms of no of arguments or its data types or sequence

Imp:

1. **Method overloading applies to both static and non-static methods**
2. **Method can be overloaded in the same class (usually) or in subclasses**

```
Package com.qspiders.pack1;

Class Demo49

{

        Void print()

        {

                SOP("print() method (no org)");

        }

        /*boolen print()

        {

                SOP("print() method (no org) Boolean type");

                Return false;

        }*/

        // you cannot have more than one method with same name with no arguments/ (same
signature in same sequence)

}

String print (int i)

{

        SOP(" print() method (one arg)");

        Return "success";

}

String print (int I, int j)

{

        SOP(" print() method (two args)");

        Return "success 2";

}

String print (long l, long n)

{

        SOP(" print() method (one arg)");

        Return "success3";

}
```

```
String print (String s, int i)

{

        SOP(" print() method (two args)");

        Return "success 4";

}

String print (int I, String s)

{

        SOP(" print() method (two args)");

        Return "success 5";

}

        PSVM()

        {

                String res;

                Demo49 d = new Demo49();

                d.print();

                res = d.print(555);

                SOP(res);

                res = d.print(123, 456);

                SOP(res);

                res = d.print(100l, 200l);

                SOP(res);

                res = d.print(10, "Hello");

                SOP(res);

                res = d.print("Hello", 10);

                SOP(res);

        }

}
```

Method overriding:

Sub class providing a new implementation of already present method (non static) in super class

Rules for method overriding

1. There should be inheritance or 'IS A' relationship
2. Method should be non-static
3. Method must have same name as in the superclass with same signature and in same order
4. Return type should also be same

Imp: Static methods are not overridden in the sub class but it is hidden.
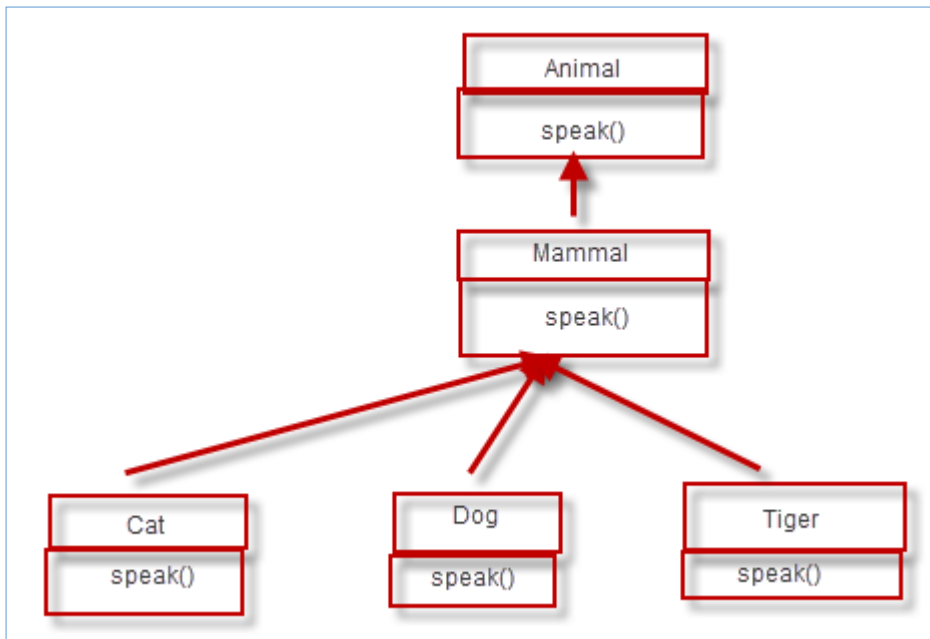

```java
package com.qspiders.pack1;
class Animal
{
    void speak()
    {
      System.out.println("Animal speaking");
    }
}

class Mammal extends Animal
{
    void speak()
    {
      System.out.println("Mammal speaking");
    }
}

class Dog extends Mammal
{
  void speak()
  {
```

```
    System.out.println("Bowbow");
  }

}


class Demo51
{
  public static void main(String args[])
 {

  Mammal m1=new Mammal();
  m1.speak();

  Animal a1=new Animal();
   a1.speak();

  Dog d1=new Dog();
  d1.speak();

 }

}
```

13th Oct 2016

Package com.qspiders.pack1

Class A

{

       Void print()

       {

              SOP ("Hiya");

       }

       Static void test ()

       {

              SOP ("static method");

       }

}

```
Class B extends class A

{

        Void print ()// method overriding

        {

                SOP ("Huya");

        }

        Static void test ()// hiding (when we do for a static, we will call Hiding)

        {

                SOP ("Hiding");

        }

}
```

Notes:

1. We cannot change from static method to non-static in subclass and same vice versa
2. We usually overload methods in the same class, but method can also be overloaded in the sub class. If at all it is done, will be for non-static methods
3. Overriding applies only to non-static methods
4. Overloading can be done either for static or non-static methods

```
Package com.qspiders.pack1;

Class Animal

{

        Void speak ()

        {

                SOP("Animal speaking");

        }

        Void disp ( String name, int age)

        {

                SOP("name = " + name + "Age = " + age);

        }

        Void disp (int age, String name)

        {

                SOP("name = " + name + "Age = " + age);

        }       }
```

```
Class Mammal extends Animal
{
        Void speak () // overriding
        {
                SOP ("Mammal speaking");
        }
}
Class Dog extends Mammal
{
        Void speak () // overriding
        {
                SOP(" Dog speaking");
        }
        Void disp ()
        {
        SOP("Scooby bow bow");
        }
}
Class Demo 52
{
        PSVM()
        {
                Dog d = new Dog ();
                d.speak();
                d.disp(2, "Scooby");
                d.disp("Snoopy", 8);
                d.disp();
        }
}
```

****

Package com.qspiders.pack1;

Class MummyMD

{

    void createMD()

    {

        SOP("Heat Tava");

        SOP("Put oil");

        SOP("once gold colour comes, fold that");

        SOP("serve this");

    }

```
        Void createIdly()

        {

                SOP("White Idly");

        }

}

Class MyMd extends MummyMD

{

        Void createMD ()

        {

                Super.createMD();

                SOP(" put ghee on dosa");

                SOP("Put chetny in cup");

        }

        Void createIdly ()

        {

                Super.createIdly();

                SOP(" prepare masala idly");

        }

        Void MixedIdly ()

        {

                Super.createIdly();

                SOP(" put this on left side of plate"); // Whiteidly

                this.createIdly();

                SOP("put this on right side of plate"); // Masalaidly

        }

}

Class Demo53

{

        PSVM()

        {

                MyMD m1 = new MymD ();
```

```
                M1.createMD();

                SOP("----next week-----");

                M1.createIdly();

                SOP("----next week-----");

                M1.MixedIdly();

        }

}
```

```
Package com.qspiders.pack1;

Class O

{

}

Class A extends O

{

Int I,j;

A();

{

        this(100, 200);

        //super (); // we cannot have 'this' and 'super' class constructor together

}

Private A (int I, int j)

{

        Super (); // automatically compiler put by default

        This.i=I;

        This.j=j;

}
```

```
Void disp()

{

        This.print();

}

Void print()

{

        SOP("print() method");

}

}

Class Demo54

{

        PSVM()

        {

                A a1 = new A ();

                SOP("I = " + a1.i + "j= " + a1.j);

                a1.disp();

        }

}
```

What is 'this' and 'super'?

- 'this' and 'super' are keywords which can be used along with dot (.) or ().
- 'this' refers to current object and can be used within a instance (non-static) method or a constructor
- We can refer to any members of the current object using 'this'. We can call current class constructor from another constructor of the same class using 'this ()'.
- 'super' refers to super class non-static members. We can call super class instance members (non static) from sub class instance method using 'super.membername' or we can call super class constructor from sub class constructor using super ();
- Super () and this () can be used only in constructors. 'Super.' And 'this.' Can be used only in instance methods. Super () and this () cannot be used together.

Can we override static method?

No, static method cannot be overridden.They belong to class ,though it gets inherited ,if we redefine ,the original method gets hidden.

****************object creation scenarios*************

1st----------

```
class Apple
{
  Apple()
  {

  }
}

class Demo
{
  public static void main(String[] args)
  {
    Apple a1=new Apple();
  }

}
```

2nd********************

```
class Apple
{
  private Apple()
  {

  }

  static Apple getApple()//Factory method
  {
    return new Apple();
  }

}

class Demo
{
  public static void main(String[] args)
  {
    // Apple a1=new Apple(); //not possible since //cons is private
      Apple a1=Apple.getApple();

  }

}
```

***************3rd**********************

```
class Apple
```

```
{
  private Apple()
  {

  }

  static Apple getApple()
  {
    Apple a=new Apple();

    return a;

  }

}
```

****************4th way***********************

```
//simple way to create a singleton class
package com.qspiders.pack1;

class Apple
{
  private static  Apple a1;

  private Apple()
  {

  }

  static Apple getApple()//Factory method
  {
    if(a1==null)
      a1=new Apple();

    return a1;
  }

}

class Demo
{
  public static void main(String[] args)
  {
    // Apple a1=new Apple(); //not possible since //cons is private
      Apple a1=Apple.getApple();

  }
}
```

```
*************5th***************
package com.qspiders.pack1;

class Apple
{

   Apple getApple()
  {
     return new Apple();//not meaningful
                //because constructor is
                 //not private
  }

}

class Demo
{
  public static void main(String[] args)
  {
     Apple a1=new Apple();
     Apple a2=a1.getApple();//not meaningful
  }
}
```

```
***********5th to return current reference********
package com.qspiders.pack1;

class Apple
{
   Apple getApple()
  {
     return this;
  }
}


class Demo
{
  public static void main(String[] args)
  {
     Apple a1=new Apple();
     Apple a2=a1.getApple();
     System.out.println(a1==a2);
  }

}
```

14th Oct 2016

***********6th ********

Example for covariant return type

Package com.qspiders.pack1;
Class Apple
{
        Apple get()
        {
                Return this;
        }
}
Class ootyApple() extends Apple
{
        OotyApple get();
        {
                Return this;
        }
        Void print()
        {
                SOP("covariant return type");
        }
}

Note:
- Covariant applies for non-primitive and non-final return types
- We can change to bugger access specifier in the subclass. We cannot change from bigger to smaller access specifier in the subclass. **Please dine below example program for this**

Class A

{

    Void print()

    {

            SOP("print method called");

    }

    Private void disp()// doesn't get inherited. Because it is private

    {

            SOP("disp method called");

    }

    Public void paint()

    {

```
                    SOP("Paint method called");

        }

}

Class B extends A

{

        Void print() // same access specifier as in super class for print

        {

                SOP("print overridden method called");

        }

        Private void disp() // we are not overriding since it is not inherited (private). This method is

                        // new method for this class.

        {

                SOP("disp method of current B class);

        }

        Void print() // this will not work. Cannot change to small access specifier. In super class, the

                // access specifier is public. Here we are changing the access to default which is
                // small compared to it in super class which we is not possible

        {

                SOP("paint method called");

        }

}
```

Abstraction

Class types

1 – Abstract class

2 – Concrete class

Abstract method – A method without body is called abstract method. We will define this method with abstract

Ex: abstract print();

Concrete method – A method with body is called Concrete Method.

Ex: Void print()

         {

         }

Abstract class D

{

         Void test1 ()

         {

                 // concrete method

         }

         Abstract void test (); // abstract

}

               *****

```
Package com.qspiders.pack1;
Abstract class B // class is abstract, because it has abstract method.
{
        abstract void print(); // this has no BODY. Hence abstract method
        // abstract static void disp(); // we cannot have static and abstract

        Static void disp()// we can have static and concrete
        {
                SOP ("disp method");
        }
}
```

```
Class C extends B
{
        Void print()
        {
                ("print() implemented in class C");
        }
}
Class Demo55
{
        PSVM()
        {
                C c1 = new C();
                C1.print();
                C1.disp(); // compiler replaces ref var name c1 with class name like C.disp().
                B.disp(); // calling static concrete method of an abstract class
        }
}
```

Note:
- To declare an abstract method we near to use abstract keyword i.e. abstract method has no body or block & it is incomplete
- If a class has an abstract method then class should be declared as abstract
- Abstract class does not mean it should compulsorily have abstract methods, but the opposite is not true
- Abstract class cannot be used to create objects
- Abstract methods should only be non-static
- Concrete methods can be either static or non-static

```java
package com.qspiders.pack1;

abstract class C{

    static int i;
    int j;
    static final double PI=3.14;


    static void print()
    {
     System.out.println("i="+ i);
    }


    void disp()
    {
     System.out.println("j=" + j);
    }
    //abstract static void print1(); cannot have
    //static method as abstract
}
```

```
class D extends C{

}
class Demo56{

    public static void main(String...arg)
    {
       D d1=new D();
        d1.disp();
       D.print();

     System.out.println("--------");
     System.out.println(d1.j);//non static
                 //variable
   System.out.println(d1.i);//static variable
    System.out.println(D.i);

    System.out.println("--------");
    C.print();//calling a static method of an
          //abstract class
     D.print();//calling a inherited static method
     }
}
```

15th Oct 2016

//Demo for using different access specifiers

Package com.qspiders.pack1;

Abstract class A

{

Abstract void test1();

Abstract public void test2();

// abstract private void test 3(); // abstract and private methods are not inherited and no question of overriding

Abstract protected void test 4();

}

Class B extends A

{

Public void test1 ()

{

SOP("test1() of implemented in class B");

```
        }

        Public void test2 ()

        {

                SOP("test2() of implemented in class B");

        }

        Private void test3 () // this private non static method belongs to class B and not inherited.

        {       Because private method cannot get inherited from A

                SOP("test3() of class B");

        }

        protected void test4 ()

        {

                SOP("test4() of implemented in class B");

        }

}
```

--Demo for abstract class having main method---

```
Package com.qspiders.pack1;

Abstract class A

{

        Static int i=55;

        Static void print()

        {

                SOP("i= " + i);

        }

        Abstract void disp();

        PSVM() // we can have main method in abstract class

        {

                A.print();

        }

}
```

--We can call abstract class constructors from only concrete sub class—

Package com.qspiders.pack1;

Abstract class A

{

     Int I;

     A(int i)// we can have constructor in abstract class

     {

          SOP("hello");

          This.i = I;

     }

}

Class B extends A

{

     B()

     {

          Super(100);

     }

}

Class Demo 58

{

     PSVM()

     {

          //A a1 = new A(); // A is abstract class, cannot be instantiated. Means we cannot create an object

          B b1 = new B();

          SOP(b1.i);

     }

}

o/p – Hello

     100

****

Package com.qspiders.pack1;

Abstract class W

{

       Abstract void test1 ();

       Abstract void test2 ();

       Abstract void test3 ();

}

Abstract class X extends W

{

       Void test1 ()

       {

              SOP("test1() implemented in class X");

       }

}

Abstract class Y extends X

{

       Void test2 ()

       {

              SOP("test2() implemented in class Y");

       }

}

Abstract class Z extends Y

{

       Void test3 ()

       {

              SOP("test3() implemented in class Z");

       }

}

```
Class Demo59
{
        PSVM()
        {
                Z z1 = new Z();
                Z1.test1();
                Z1.test2();
                Z1.test3();
        }
}
```

What is abstraction?

Abstraction represents the essential features of a system without/ partially involving in complexity.

In java abstraction is implemented using abstract class and interfaces. Implementing classes define the actual working

Abstract class: a method without body is known as abstract method. The abstract method should be declared with the keyword abstract.

Abstract class is a class which can contain abstract methods and concrete methods

Concrete class: a method which contain both declaration and method body is known as concrete method or complete method

If a class contains only concrete methods, then such class is known as concrete class

- An abstract class can have any number of abstract method as well as concrete methods or only concrete methods or only abstract method
- If a class contains at least one abstract method, then that class should be declared as abstract
- An abstract class cannot be instantiated. It means we cannot create an object for abstract class
- An abstract class should be extended to the subclass. The sub class should implement all methods (abstract) or else the subclass also should be declared as abstract class
- Whenever we use abstract methods the abstract methods should be completed/overridden in the subclass
- It can contain only non-static abstract methods
- It can contain both static and non-static concrete methods
- It can contain both static and non-static variables
- It can have constructor which will be invoked when concrete subclass object is created
- It can have static and non-static final variables

Note: When we add abstract methods to the super class then all the subclasses are affected.

VV IMP

Interface

Interface interfacename

{

     -----   interface body

}

Interface is a type/definition block which contains only abstract methods

It is like 100 % abstract class

- An interface type starts with keyword interface
- A java file can contain only interface definition block
  After compilation of interface definition block compiler generated class file for the interface
- Java file can have both class definition block and interface definition block
- Inside interface definition block, concrete methods are not allowed.
- The interfaces should contain only abstract methods (that too non static)
- By default the methods in interface are abstract and public
- Other than public access specifier we cannot use other access specifier
- All variables in an interface must be public, static and final. i.e. it can declare only constants and it should be initialized at the declaration time
- A method cannot be declared as static in interface
- A class cannot extend more than I class (java doesn't support multiple inheritance) but a class implement more than one interface
- We cannot instantiate an interface
- An interface can extend one or more other interfaces
- An interface cannot extend anything other than interface
- An interface cannot implement another interface or class

****

Package com.qspiders.pack1;

Abstract interface Idemo1 // abstract is optional as interface class is by default abstract

{

       Void test 1(); //**public & abstract is optional as the method in interface is by default public & abstract**

       Public abstract void test2 ();

}

Class A implements Idemo1

{

    Public void test 1(); **// we have to put public compulsorily as it is by default public in interface**

    {

        SOP("test1()implemented in class A");

    }

    Public void test2()

    {

        SOP("test2() implemented in class A");

    }

    Void disp()

    {

        Test2(); // we can call test2()directly as it is in same class

    }

}

Class Demo70

{

    PSVM()

    {

        A a1 = new A();

        A1.test1();

        A1.test2();

    }

}

When we compile we will get three class files

- Ideom.class
- A.class
- Demo70.class

****

```java
Package com.qspiders.pack1;

Interface Idemo2
{
        Void test1();

        Void test2();
}

Abstract class B implements Idemo2
{
        Public void test1()
        {
                SOP("test1() implemented in class B");
        }
}

class c extends B
{
        Public void test2()
        {
                SOP("test2() implemented in class C");
        }
}

Class demo71
{
        PSVM()
        {
                C c1 = new C();
                C1.test1();
                C1.test2();
        }
}
```

```
Package com.qspiders.pack1;

Interface Idemo3

{

        Void test1();

}

Interface Idemo4

{

        Void test2();

}

Class D implements Idemo3

{

        Public void test1()

        {

                SOP("test1() implemented in class D");

        }

}

Class E extends D implements Ideom4

{

        Public void test2()

        {

                SOP("test2() implemented in class E");

        }

}

Class Demo72{

        PSVM() {

                E e1 = new E();

                E1.test1();

                E1.test2();

        }

}
```

Note: extend and interface

****

```
Package com.qspiders.pack1;

Interface Idemo5
{
        Void test1();
}

Interface Idemo6
{
        Void test2();
}

Class F implements Idemo5, Ideom6
{
        Public void test1()
        {
                SOP("test1() implemented in class F");
        }
        Public void test2()
        {
                SOP("test2() implemented in class F");
        }
}

Class Demo73
{
        PSVM()
        {
                F f1 = new F();
                f1.test1();
                f1.test2();
        }
}
```

153

****

Package com.qspiders.pack1;

Interface Idemo7

{

    Void test1();

}

Interface Idemo8

{

    Void test2();

}

Class G implements Idemo7, Ideom8

{

    Public void test1()

    {

        SOP("test1() implemented in class G");

    }

}

Class H extends G

{

    Public void test2()

    {

        SOP("test2() implemented in class H");

    }

}

Class Demo74{

    PSVM() {

        H h1 = new H();

        h1.test1();

        h1.test2();

    }

}

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

Package com.qspiders.pack1;

Interface Idemo9

{

        Void test1();

}

Interface Idemo10

{

        Void test2();

}

Interface Idemo11 extends Idemo9, Idemo10

{

        Void test3();

}

Class I implements Idemo11

{

        Public void test1()

        {

                SOP("test1() implemented in class I");

        }

        Public void test2()

        {

                SOP("test2() implemented in class I");

        }

        Public void test3()

        {

                SOP("test3() implemented in class I");

        }

}

Class Demo75

{
```

```
PSVM()
{
        I i1 = new I();
        I1.test1();
        I1.test2();
        I1.test3();
}
}
```

Note: Interface can extend only interface and can extend more than one interface

Class can extend only class and implements interface, it can extend only one class and implements more than one interface

Abstract class is a class definition block which can contain abstract methods and concrete methods.

Interface is a type which contains only abstract methods.

Difference between abstract Class & Interface

• An abstract class should be declared with the keyword abstract.

interface starts with keyword interface and by default is abstract

• The abstract class can contain both concrete and abstract methods.

The interface contains only abstract methods.

• In abstract class the method should be explicitly declared as abstract

In interface by default all the methods are abstract

• In abstract class we can use all the access specifier except private.

In interface we can use only(default) public access specifier

• An abstract class abstract methods are non static and concrete methods can contain static & non static

Interface should have only non static methods

• A class can extend an abstract class, but a class should implement an interface

• A class cannot extend two abstract class at a time but an interface can extend one or more interfaces

• A class can extend only one class implement multiple interfaces

• In abstract class we can have static,non static and final variables and in interface we can have only final variables(public ,static and final)

• In an abstract class if a concrete method is added, the subclasses will not be affected and will be affected only when you add an abstract method

In interface if you add a new method then all the classes which implements the interface will break(fail)

17th Oct 2016

When do we go with abstract class and when with interfaces?

- When some of the functionalities are common among implementing classes and some are class specific then we can go with **abstract classes**
- When all the functionalities are common to the implementing classes and implementation are class specific (dependant) then we go with **Interfaces**

Ex: for abstract class

Abstract TV

{

      Abstract void display()

      Void volup()

      {

      }

      Void voldown()

      {

      }

      Void switchon()

      {

      }

      Void switchoff()

      {

      }

}

Class Sony extends TV

{

```
        Void display()

        {

        }

}

Class Samsung extends TV

{

        Void display()

        {

        }

}
```

Ex: for interface

```
Interface browser

{

        Void openhtml();

        Void forward();

        Void backward();

        Void refresh();

}

Class IE implements Browser

{

}

Class Chrome implements Browser

{

}

Class Firefox implements Browser

{

}

Class Safari implements Browser

{

}
```

Difference b/n constructor and method

Constructors	Methods
Constructors are blocks used to build the object of the class	methods are blocks used for code reusing and perform actions
the constructor name should be same as class name	method names are identifiers which describe the behavior of the class
the constructors do not have return types	methods have return types
return statements are not used in constructors	In method return statements are used
constructors cannot be declared as static or non-static	methods can be declared as static or non-static
constructors can be overloaded but cannot be overridden	methods can be overloaded and overridden
constructors cannot be declared as abstract	methods can be declared as abstract
constructors cannot be final	methods can be final
Recursive calls to the constructors are not allowed	Recursive calls to methods are allowed

Final keyword:

Whenever you declare final variable it should be initialized in the same line in global scope and in local scope it can be done in same line or later. Once a value is assigned, you cannot change it.

Final keyword can be applied for a variable, method or a class.

Package com.qspiders.pack1;

Class orange

{

      Final int NO_WHEELS = 4;

}

Class Demo76

{

      Final static double PI=3.14;

      Final static Orange o1 = new orange();

      PSVM()

      {

            SOP(Demo76.PI);

            // Demo76.PI = 12; // we cannot reinitialize final variable

            SOP(Demo76.o1);

            // o1 = new orange (); // cannot be reassigned as its final

            Print (100, new Orange ());

```
                }

                Static void print (final int I, Orange o1)

                {

                        //I = 120; // final method parameter cannot be reassigned

                        O1 = new Orange ();

                        //o1.No_WHEELS = 6; // final variable of Orange class cannot be changed

                }

}
```

Final method:

Final method cannot be overridden

Package com.qspiders.pack1;

Class B

```
{

        Final void test1 ()

        {

                SOP("test1() of class B");

        }

        Static final void disp ()

        {

                SOP("disp() of class B");

        }

}
```

Class C extends B

```
{

        Void test1 ()

        {

                SOP("test1() of class C");

        }

        static void disp ()

        {

                SOP("disp() of class C");
```

```
        }

}
```

o/p – gives you compile time error, because we cannot override the final methods test1() and disp() of class B in class C.

test1() in C cannot override test1() in B. Overridden method is final

disp() in C cannot override disp() in B. Overridden method is static and final

final class – example

```
Package com.qspiders.pack1;

Final class D

{

        Void test1()

        {

                SOP("test1() of class D");

        }

}

Class E extends D ---→ this line gives error as the above class D is final

{

        Void test2()

        {

                SOP("test2() of class E");

        }

}
```

o/p – gives you compile time error

Note: final class cannot be inherited or extended

Final keyword:

- If a variable is declared as final such variable (primitive and reference variable) cannot be modifier
- A method (static and non-static) can be declared as final but cannot be overridden/ redefined in the subclass
- A class can be declared as final. Final class cannot be inherited to the subclass
- Constructor cannot be declared as final
- Example of final classes are string class, string buffer, string builder and all wrapper classes
- Final and abstract cannot be used together

JAVA questions

- What is java? – Java is high level object oriented programming language
- Who is the creator of Java? - JAMES GOSLING
- Which company is the official implementer of java? - **Oracle**
- What software is needed to work on java? - **JDK**
- What does jdk stand for? – **Java development Kit**
- What is the latest version of jdk? – **1.8**
- How do you install jdk? –

  https://www.oracle.com/index.html - Downloads - Popular Downloads – Java SE – JDK Download-Windows x64
  Install latest version

- java or javac is not recognised as internal or external command why?
  If java is not installed at all
  If java is installed and not defined the path

- Why do you need to set the path of java commands?
  To compile the java files and execute the class files

- How do you set the path?
  My computer – properties – Advanced system settings – Environemntal Variables – System Variables – Paths – paste the path that copied for bin folder (i.e. C:\Program Files\Java\jdk1.8.0_102\bin) by separating with semicolon (;)

- Assuming Projectname with src and bin folders
  what is the command for compilation of Apple.java?

  javac –d ../bin Apple.java

- What file do you get when you compile?
  Class file
- What is the command for execution of the class file?
  Java classfilename
- What format it is in?
  Class file will be in bytecode format
- What is a class?
  Class can be defined as a blue print/ template to create an object. A class specifies the design of an object. It states what data an object can hold and the way it can behave
- What are the members of a class?
  Variables, methods and inner classes

- What is a variable?
  Variable is named memory location which can hold value and we can change value any no of times during execution. It is also an identifier in java.

- What is a method?
  A method describes the behaviour or actions that can be performed by an object or a class.

- What is an object?
  Object is an instance of a class

- Create an Orange object using a ref variable and one object without it

```
Orange o1 = new Orange();
new Orange();
```

- What is a type?

Type describes the type of data which a variable can hold.

- Classify diff types and its sub types?

Primitive type -
        byte, short, int, long, float, double, char, boolean

Non Primitive type –
        Class, interface and arrays

- What kinds of variables are there?

There are 7 kinds of variables (as per jls7). Kinds determine the scope and behaviour of a variable.

They are
1. **Local variable**
2. **Static variable**
3. **Non static (instance) variable**
4. **Method parameter**
5. **Constructor parameter**
6. **Exception parameter**
7. **Array components**

- What are operators and classify based on operations and operands?

An operator is a symbol or character that allows a programmer to perform certain operations like (Arithmetic and logical) on data and variables (Operands)

There are six types of operators are there

1. **Arithmetic (+, -, *, /, %, ++, --)**
2. **Assignment (=, +=, -=, *=, /=, %=)**
3. **Relational (==, !=, <, >, <=, >=)**
4. **Logical (&, |, ^, !, &&, ||)**
5. **Conditional (?, ;)**
6. **Instanceof (instanceof)**

Depending on the number of operators, operators can be of following three types.

Unary Operator: It takes operand such as ++X, Y--. Here X and Y are variables where ++ and – are operators.
Binary Operators: It takes 2 operands such as X+Y, X-Y, X>Y etc. Here X and Y are variables where +, - and > are operators
Ternary Operators: It takes 3 operands such as Z=X>Y?X:Y. here X, Y and Z are operands where ?, :, > are operators

- What is == used for?

  <mark>It is a relational operator which is used to compare two primitive variables where it will compare values.
  It can compare two reference variables where it will compare addresses</mark>

- What is instance of? And when do you use it?

The instanceof is a binary operator that checks whether an object is of a particular type (here type can be class, interface or any array). **It is used for object or reference variables only**

18th Oct 2016

Casting (Type Casting) (VV IMP)

Usually we will call casting as **type casting** in java

There are 2 types of castings

1 – Primitive casting

  a. Auto Widening
  b. Explicit narrowing

2 – Non Primitive casting (derived casting)

  a. Auto up cast
  b. Explicit downcast
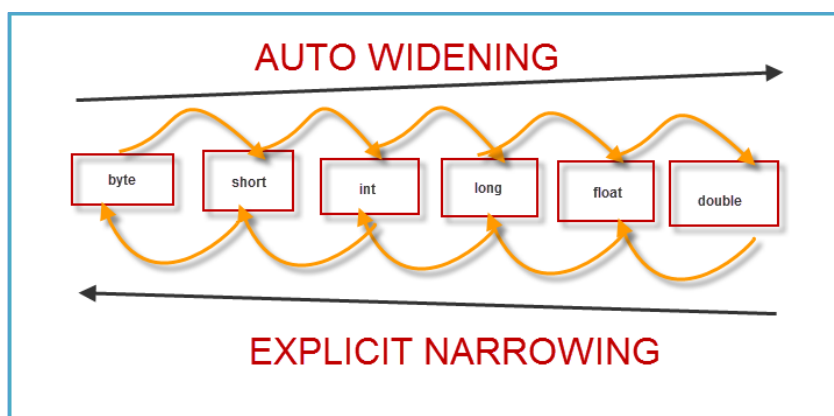
Int I = 10; ------- here the left side and right side values are of same type

Double d = 10.20;

String str = "Rama";

If lest side and right side are of the same type then it is homogeneous statements

Int j = 10.25 --------- here the left side and right side values are of different type.

String str = 10;

If the left side and right side are different then these are called heterogeneous statements

Auto widening ⎯⎯⎯⎯⎯→ compiler does it

Explicit narrowing ⎯⎯⎯⎯⎯→ programmer should explicitly tell the compiler

Double d = 10.20;

Int k = (int) d;

⎯⎯⎯⎯⎯→ Cast operator

Package com.qspiders.pack1;

Class Demo77

{

      PSVM()

      {

            int I = 100;

            long l = I; //auto widening

            SOP(l);

            byte b;

            b = (byte)l; // explicit narrowing

            SOP(b);

            l = 130;

            b= (byte)l; // -126 is o/p. because the range of byte is -127 - +128. Here the value of l is 130. So when it is more than 128 it will round again and gives the o/p based upon the extra number. Here 130 is 2 more than 128. So it will turn in clock wise and gives -126. Please see below picture for more details

            SOP(b);

            l=260;

            b= (byte)l; // o/p is 4 . Please see below picture for more details

            SOP(b);

            Double d = 12.44;

            Int k = (int)d; // explicit narrowing

            SOP(k);

```
            Char ch =25; // assigning char equal to Unicode no 25

            SOP(ch);

            Int i1 = 65;

            Char ch1 = (char) i1;

            SOP(ch1);

            double d1 = 65.43;

            Char ch2 = (char) d1;

            SOP(ch2);

            Long l1 = 100;

            Char ch3 = l1; // cannot be done as char expects only characters or numbers

            Double dd = 100;

            SOP(dd);

            Int i1 = (int) 100.0;

            SOP(i1);

    }

}
```
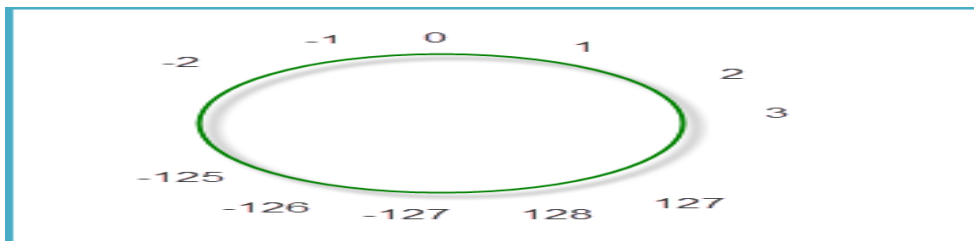

o/p –

100

100

-127

-26

12

↓

A

A

100.0

100

Interview question: write a program to print all the Unicode characters using casting

```
for (int i=0; i<=128; i++);

{

        Char ch = (char)I;

        SOP(ch); // applying casting and printing

}

                (Or)

for (int i=0; i<=128; i++);

{

        SOP((char)I); // directly printing

}

Long l1 = (long) (float) (long) (int) (short) (byte) f; // this is correct

int i1 = (int) (double) (float) (long) (int) (short) (byte) f; // this is correct

int d1 = (double) (float) (long) (int) (short) (byte) f; // this is incorrect
```

Note: in the above, we can convert or cast from one type to any other possible type any no of times in a statement but when the value is being assigned it should be cast or narrowed to the type expected on the left hand side

If left hand side is int, right hand side the first word should be of same type (i.e. int);

If left hand side is long, right hand side the first word should be of same type (i.e. long);

Derived casting

Derived casting is also called as non-primitive type casting and it is of 2 types

1 – Downcast (explicit)

2 – Up cast (auto)
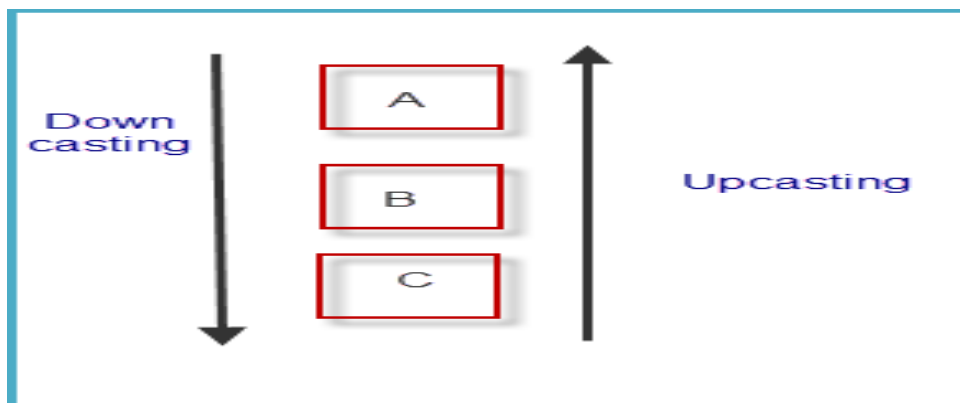
Derived casting – converting an object to behave like another possible type is called derived casting

Up cast (auto cast): converting a subclass object to behave like a super type is called up casting

```java
Package com.qspiders.pack1;

Class Animal
{
        Void eat()
        {
                SOP("eating");
        }
        Void sleep()
        {
                SOP("sleeping");
        }
}
Class Dog extends Animal
{
        Void bark()
        {
                SOP("bow bow");
        }
        Void wagtail()
        {
                SOP("wag tailing");
        }
}
Class Demo78
{
        PSVM()
        {
                Dog d1 = new Dog();
                d1.bark;
                d1.wagtail;
```

```
            d1.eat; // can access its own and inherited methods

            d1.sleep;

            Animal a1 = new Dog();

            a1.eat;

            a1.sleep;

            a1.bark;// error. Cannot access its own methods

            a1.wagtail; // error. Cannot access its own methods

        }

}
```

Down casting (explicit casting): converting an upcasted object to behave like a sub type is called down casting

<mark>**Note: Down casting will be done only after up casting. It means to do down casting, first we need to do up casting**</mark>

```
        Animal a1 = new Dog(); // first we are doing up casting

        a1.eat();

        a1.sleep();

        Dog d = new Animal(); // direct down casting is not possible

        // Dog d = (Dog) new Animal(); // explicit down casting, compiler will compile. But JVM will throw cast exception

        Dog d = (Dog) a1; // up casted object being down casted.

        d.eat();

        d.sleep();

        d.bark();

        d.wagtail();

        }

}
```

19th Oct 2016

Package com.qspiders.pack1;

Class A

{

}

Class B extends A

{

}

Class C extends B

{

}

Class Demo79

{

    PSVM()

    {

        A a1 = new A(); //normal object creation

        B b1 = new B();//normal object creation

        C c1 = new C();//normal object creation

        A a2 = a1; // no abject creation, but a2 refers to same object as a1

        B b2 = b1; // no abject creation, but b2 refers to same object as b1

        C c2 = c1; // no abject creation, but c2 refers to same object as c1

        A a3 = new B(); // auto upcasting

        B b3 = new C(); // auto upcasting

        A a4 = new C(); // auto upcasting

        B b4 = new A(); // direct down casting of super class object is not possible. Gives Compile time error - incompatible types

        C c4 = new B(); // direct down casting of super class object is not possible. Gives Compile time error - incompatible types

B b4 = (B) new A(); // explicit casting of a super object. This will compile but it will not execute and gives 'class cast exception'

C c4 = (C) new B(); // explicit casting of a super object. This will compile but it will not execute and gives 'class cast exception'

C c5 = (C) new A(); // explicit casting of a super object. This will compile but it will not execute and gives 'class cast exception'

B b5 = (B) a3; // down casting of any updated object

C c5 = (C) b3; // down casting of any updated object

B b6 = (B) a4; // down casting of any updated object

C c6 = (C) a4; // down casting of any updated object

        }

}

What is the use of inheritance?

Reusability, extendibility and modularity

Class Animal

{

}

Class Mammal extends Animal

{

}

Class Monkey extends Mammal

{

}

Upcast –

Animal a1 = new Mammal();

Mammal m1 = new Monkey();

Animal a2 = new Monkey();

Downcast –

Mammal m2 = (Mammal) a1;

Monkey M3 = (Monkey) m1;

Monkey M4 = (Monkey) a2;

Mammal m5 = (Mammal) a2;

****

Class Animal

{

}

Class Dog extends Animal

{

}

Class Bulldog extends Dog

{

}

Upcast –

Animal a1 = new DOg();

Dog d1 = new Bulldog();

Animal a2 = new Bulldog();

Downcast –

Dog d2 = (Mammal) a1;

Bulldog bd1 = (Bulldog) d1;

Bulldog bd2 = (Bulldog) a2;

Dog d3 = (Dog) a2;

****

Package com.qspiders.pack1;

Class D

{

      Void test 1()

      {

            SOP("test1() of class D")

      }

}

Class E extends D

{

```
        Void test 2()

        {

                SOP("test2() of class E")

        }

}

Class F extends E        {

        Void test 3()

        {

                SOP("test3() of class F")

        }

}

Class Demo80

{

        PSVM()

        {

                D d1 = new E();

                d1.test1();

                d1.test2(); // not works. can't access its own class members of upcasting

                SOP("-----");

                // D d2= new D ();

                // E e1 = (E) d2; // not possible. We will get class cast exception

                // E e1 = (E) new D();// not possible. We will get class cast exception

                // E e1 = (E) new d1;// Down casting of an updated object.

                e1.test1();

                e1.test2();

        }

}
```

1. E e1 = new D() // direct down casting will not work and gives CTE – 'incompatible types'
2. Explicit down casting of a super class object to sub class results in 'class cast exception'
   a. E e2 = (E) new D(); // class cast exception while runtime
   b. D d2 = new D();
      E e3 = (E) d2; // class cast exception

Note:

1. (a) and (b) are same, in (a) we are passing D object and in (b) we have reference variable
2. Whenever you do up cast, subclass members can't be accessed

Imp Notes:

-   Subclass features (members) will not be in super class.
-   Super class features (members) will be in subclass.
-   When an object is up casted the objects own members becomes invisible

Type casting – converting one type to another type is called type casting

Java supports two types of type casting

1. Primitive type casting
2. Derived casting

Primitive type casting: converting one primitive type to another primitive type is known as primitive type casting

1. Auto widening: converting a smaller primitive type to any of the bigger primitive type is known as widening. Since widening is done automatically by compiler it is also called as <u>auto widening</u>
2. Explicit widening: converting the bigger primitive type to the smaller primitive type is known as <u>explicit narrowing.</u> It should be explicitly specified in the program using () cast operator by the user

Derived casting: converting an object to behave like another possible type is known as derived casting. In order to convert an object to another type the class should have IS A relationship

1. Up casting: converting an object of subclass to behave like any of the super type is called <u>up casting</u>. It is automatically done by compiler hence called auto up cast. It is automatically alone by compiler hence called auto up cast
2. Down casting: converting an up casted object to behave like subclass type. It should be explicitly specified in the program by the user, Down casting cannot be directly done only an up casted object can be down casted. An explicit down cast using () of a super class object to any of the sub class type compiler. But throws runtime exception 'class cast exception'.

20th Oct 2016

Package com.qspiders.pack1;

Class Fruit

{

    Void test()

    {

        SOP("test() of class Fruit");

    }

}

Class Apple extends Fruit

{

    Void test()

    {

        SOP("test() of class Apple");

    }

}

Class OotyApple extends Apple

{

    Void test()

    {

        SOP("test() of class OotyApple");

    }

}

Class Execution

{

    Static void (Fruit f)

    {

        If (f instanceof Fruit) // or we can use If (f != null)

        f.test();

        else

        SOP("null value passed");

```
        }

}

Class Demo81

{

        PSVM()

        {

                Fruit f1 = new Fruit();

                Execution.start(f1);

                Execution.start(new Fruit());

                Execution.start(null);// generates 'null exception' and it can be handled using
instance of operator. Please see yellow highlighted line above for this


                Apple a1 = new Apple();

                Execution.start(a1);

                Execution.start(new Apple());

                Execution.start(null);


                OotyApple a1 = new OotyApple();

                Execution.start(a1);

                Execution.start(new OotyApple());

                Execution.start(null);

        }

}
```

Very imp Note: When we up cast, we will get overridden methods output

When do you get null pointer exception?

- **When we try to access a member of an object by a reference variable and that variable is not pointing towards any object, then we get null pointer exception**

Marker Interfaces:

These interfaces do not have any methods and gives special instruction to the JVM.

Ex: cloneable, serializable, eventlistener

```
Package com.qspiders.pack1;

Class Apple implements cloneable

{

        PSVM()

        Clone Notsupported rxception

        {

                Apple a1 = new Apple();

                Apple a2 = new (Apple)a1.clone();

                SOP("a1==a2"); false

        }

}
```

// reference variable of an abstract class can point towards object of a concrete subclass that has implemented it

```
Package com.qspiders.pack1;

abstract class AbDemo

{

        Void test1()

        {

                SOP("test1() of class AbDemo");

        }

        Abstract void test2();

}

class ConDemo extends AbDemo

{

        Void test2()

        {
```

```
                SOP("test2() implemented in ConDemo");

        }

        Void test3()

        {

                SOP("test3() of class ConDemo");

        }

}

Class Demo83

{

        PSVM()

        {

                AbDemo a = new Condemo();

                a.test1();

                a.test2();

                a.test3();// not accessible. Because its own members cannot be accessible

        }

}
```

                    ****

```
Package com.qspiders.pack1;

abstract class AbDemo1

{

        Void test1()

        {

                SOP("test1() of class AbDemo");

        }

        Abstract void test2();

}

abstract class AbDemo2 extends AbDemo1

{
```

```java
        Void test2()

        {

                SOP("test2() implemented in AbDemo2");

        }

        Abstract void test3();

}


class ConDemo extends AbDemo2

{

        Void test3()

        {

                SOP("test3() implemented in ConDemo");

        }

}

Class Demo84

{

        PSVM()

        {

                AbDemo2 a2 = new Condemo();

                AbDemo1 a1 = new Condemo();

                a2.test1();

                a2.test2();

                a2.test3();


                a1.test1();

                a1.test2();

        }

}
```

179

21ˢᵗ Oct 2016

Static method upcasting

Package com.qspiders.pack1;

Abstract class AbDemo1

{

      Static void paint()

      {

            SOP("paint() of AbDemo1");

      }

}

Class ConDemo extends AbDemo1

{

      Static void paint()

      {

            SOP("paint() of ConDemo");

      }

}

Class Demo85

{

      PSVM()

      {

            AbDemo1 a1 = new ConDemo();

            A1.paint(); // super class static method is called through reference variable. Infact Compiler replaces a1 with AbDemo1

      }

}

- Reference variable of an interface can point towards an object of a class which has implemented it

Package com.qspiders.pack1;

Abstract class IDemo1

{

    Abstract void test();

}

Class ConDemo implements IDemo1

{

    public void test()

    {

        SOP("test() implemented in ConDemo");

    }

}

Class Demo86

{

    PSVM()

    {

        IDemo1 i1 = new ConDemo();

        i1.test();

    }

}

****

Package com.qspiders.pack1;

interface IDemo1

{

    void test1();

}

interface IDemo2 extends IDemo1

{

```
        void test2();

}

Class ConDemo implements IDemo2

{

        void test1()

        {

                SOP("test1() implemented in ConDemo");

        }

        void test2()

        {

                SOP("test2() implemented in ConDemo");

        }


}

Class Demo87

{

        PSVM()

        {

                IDemo2 i2 = new ConDemo();

                I2.test1();

                I2.test2();


                IDemo1 i1 = new ConDemo();

                i1.test1();

                //i1.test2();  //not correct and gives error


        }

}
```

182

```
*************
Package com.qspiders.pakc1;

Class A
{
        Public void test()
        {
                SOP("test() implemented in A");
        }
        Public void print()
        {
                SOP("print() implemented in A");
        }
}
Class B extends A
{
        Public void test()
        {
                SOP("test() overridden in A");
        }
        Public void disp()
        {
                SOP("disp() implemented in B");
        }
}
Class Demo88
{
        PSVM()
        {
                A a1 = new B();
                A1.test();
                A1.print();
```

```
                A1.disp(); // not possible - error

        }

}
```

****

<mark>Below program is very important program with so many logics. Please read very carefully and understand the logics</mark>

// Demo for concrete class reference variables acting (casting) to behave like super type with different scenarios

```
Package com.qspiders.pack1;

Interface Idemo1

{

        Void test1();

}

Interface Idemo2 extends Idemo1

{

        Void test2();

}

Class B implements Idemo2

{

        Public void test1()

        {

                SOP("test1() implemented in class B");

        }

        Public void test2()

        {

                SOP("test2() implemented in class B");

        }

        Void print()

        {

                SOP("print() of class B");

        }

}
```

```
Class Demo89
{
        PSVM()
        {
                B b1 = new B();//normal object creation

                b1.test1();

                b1.test2();

                b1.print();

                SOP("--------");


                ((Idemo2)b1).test1(); // asking b1 to behave like Idemo2. Here Ideom2 methods are
                accessible

                ((Idemo2)b1).test2();

                //((Idemo2)b1).print();// not belongs to Idemo2

                SOP("--------");


                ((Idemo1)b1).test1(); // asking b1 to behave like Idemo1. Here Ideom1 methods are
                accessible

                //((Idemo1)b1).test2();

                //((Idemo1)b1).print();

                SOP("--------");


                Idemo1 i1 = new B(); // upcasting

                i1.test1();

                //i1.test2();// not possible


                ((Idemo2)i1).test1(); // asking i1 to behave like Idemo2. Here Ideom2 methods are
                accessible

                ((Idemo2)i1).test2();

                SOP("--------");
```

```
            Idemo2 i2 = new B(); // upcasting

            I2.test1();

            I2.test2();


            ((Idemo1)i2).test1(); // asking i2 to behave like Idemo1. Here Ideom2 methods are
            accessible

            //((Idemo1)i2).test2();// not possible

            SOP("--------");


            ((B)i2).test1();// asking i2 to behave like 'B'

            ((B)i2).test2();

            ((B)i2).print();

            SOP("----very long casting----");

            ((Idemo1) (Idemo2)(B) (Idemo1)(B)i1).test1(); // extreme left hand side mentioned
Idemo1 methods are accessible.

      }

}
```

Inheritance fails when two methods which are inherited having same method name and different return type

Purely for selenium purpose

Web driver structure
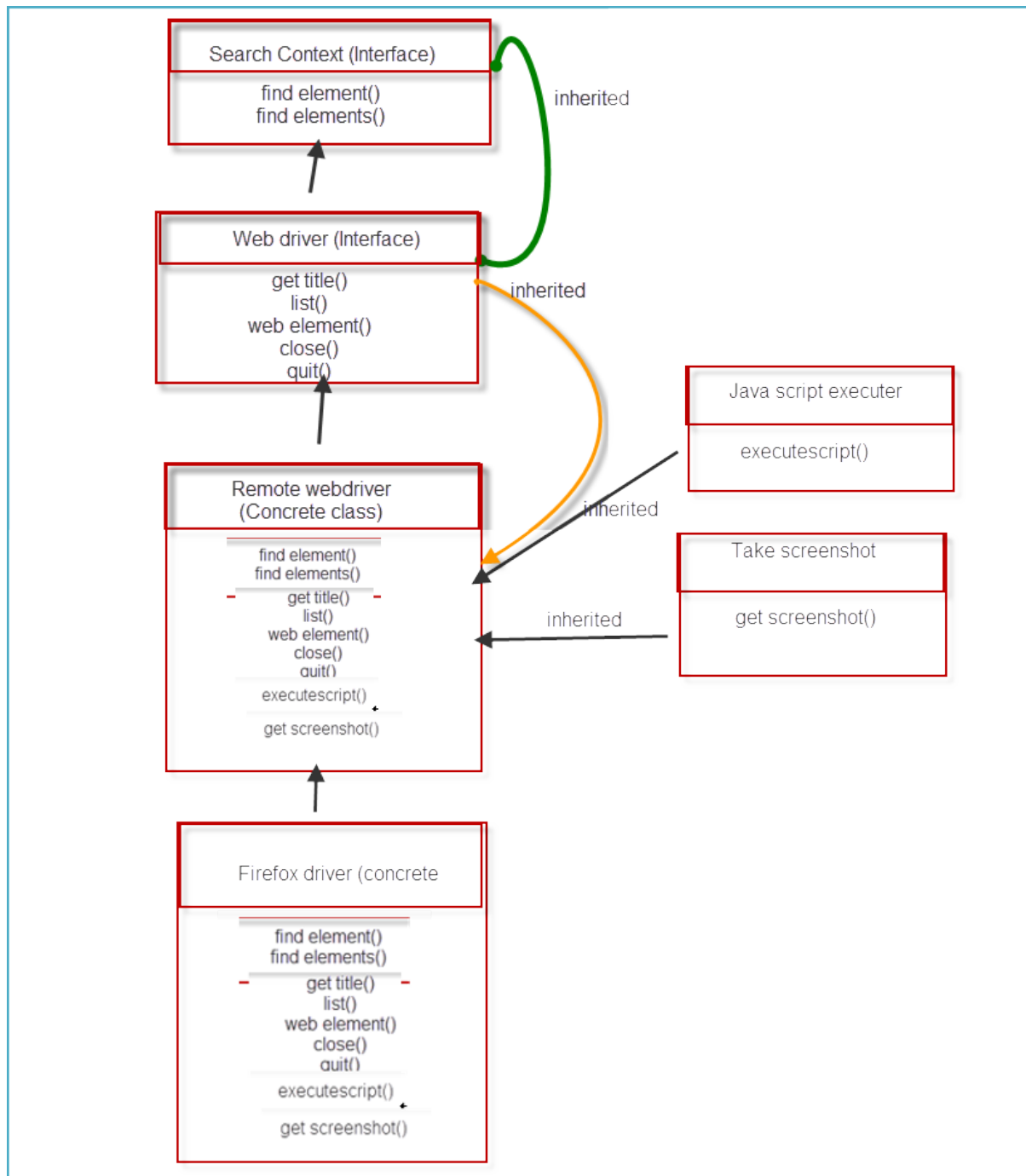
Example for upcasting:

Webdriver driver = new FirefoxDriver();

//(javascript executer)driver.executescript (paramString paramArrayof Object)

//(takescreenshot)driver.getScreenshotAs(paramoutput Type)

Driver.get("http://www.google.com");

Polymorphism

polymorhism implies one method performing same/similar

task based on the different context(as in method overloading)ex: draw(3) and draw(3,"red")

or performing redefined tasks or enhanced task(as in

method overriding).

 This can be classified into

1.static /compile time polymorphism -Method overloading

2.Dynamic /run time polymorphism -Method overriding

   (Dynamic method dispatch)-Upcasting

 In DMDispatch , when overridden method is called through the reference of base class then jvm determines which method is to be invoked.

static binding- when a method is associated to the object/class during compilation it is called static binding

ex: private,final and static methods

dynamic binding- when a method is associtated to the object during execution it is called dynamic binding.

ex: overridden methods and upcasting (calling overridden method)

23ʳᵈ Oct 2016

Java IDE

JDeveloper
NetBeans
Eclipse
IntelliJ

Eclipse is a tool which supports multiple languages i.e provides IDEs for different languages including Java .

Downloading Eclipse: Latest version of Eclipse can be downloaded f rom http://www.eclipse.org/downloads

Workspace:
Workspace is the **main directory** under which you create all your projects(Eclipse projects).Each
Project is given its own folder in Eclipse's workspace.

Workbench :
Workbench is the GUI on which you work in the Eclipse. It is built using Eclipse's own Standard Widget Toolkit(SWT) and JFace.

Perspectives:

A perspective is a group of views and editors in the Workbench window. One or more perspectives can exist in a single Workbench window. Each perspective contains one or more views and editors. Within a window, each perspective may have a different set of views but all perspectives share the same set of editors.

Ex:

Perl, Java, Python, C++,J2EE-**Perspective,**
Debug perpective

Views:
Different internal windows you see in the workbench is called
Views.

Ex:
Project Explorer,Console,Properties etc..

You create a project (java) under which you have the src folder .
You create a pacakage and create classes under those packages.
Else it will be under default package(not recommended)
You will be writing your code in Editor window.

Eclipse:

Private	default	protected	public
🟥	🔺	🔶	🟢
Red	Blue	Yellow	Green
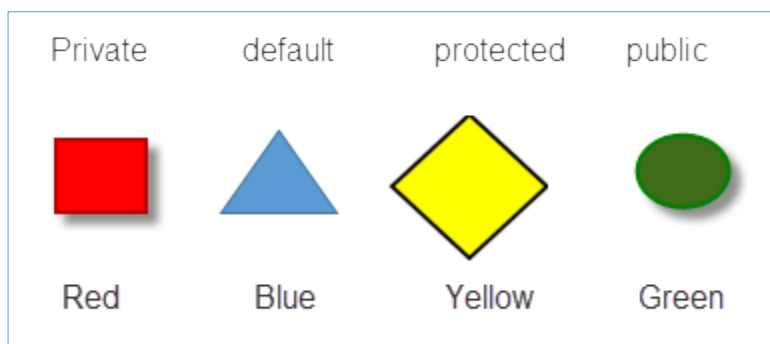
Fields: halos will be there

If we have S on top – static public method

If we have S and F on top – static and final field

Object class – VVVVVVVVVVVVVVV IMP

Object class is the super most class of the all the objects.it contains eleven (11) methods which are non-static. Out of 11, 2 are protected and 9 are public.

2 Protected methods are –

    1. clone ()
    2. finalize()

9 public methods are –

    1. toString()
    2. equals()   first 3 important methods
    3. hashcode()

    4. getclass()
    5. notify()   second 3 important methods
    6. notifyall()

    7. wait()
    8. wait(long timeout)   third 3 important methods
    9. wait(long timeout, int nanos)

****

public class A extends java.lang.object -----→ this will be done by default

{

    A()

    {

        Super(); // compiler will declare on our behalf

    }

}

****

Package com.qspiders.pack1;

Class Orange()

{

}

Public class Demo1

{

    PSVM()

```
        {
                Orange o1 = new Orange();

                SOP(o1);// prints address

                SOP(o1.toString());// prints address


                String s1 = new Stinr("Hello");

                SOP(s1);// prints value

                SOP(s1.toString());// prints value

        }

}
```

24ᵗʰ Oct 2016

Package com.qspiders.pack1;

Class A

```
{
        Int i=100;

        Public String toString()

        {
                Return "i= " + I;

        }

}
```

Public class Demo 87

```
{
        PSVM()

        {
                A a1 = new A();

                SOP(a1); // this will implicitly calls toString method

                SOP(a1.toString()); // this is explicitly calling toString method

        }

}
```

o/p – I = 100

I = 100;

****

Package com.qspiders.pack1;

Class B

{

      Int i=100, j=200;

      Public String toString()

      {

            Return "i= " + I + "j= " + j;

      }

}

Public class Demo 88

{

      PSVM()

      {

          B a1 = new B();

          SOP(b1); // this will implicitly calls toString method

          SOP(b1.toString()); // this is explicitly calling toString method

          SOP(new B();//this will implicitly calls toString method – prints value

          SOP(new Demo88());// prints address

      }

}

o/p – I = 100

j = 200;

****

Package com.qspiders.objectDemo;

Class C

{

}

Public class Demo89

{

      PSVM()

```
        {
                C c1 = new C();

                C c2 = new C();

                C c3 = new C();

                SOP(c1);

                SOP(c2);

                SOP(c3);

                SOP(c1==c3);

                SOP(c1.equals(c3)); // equals method not overridden hence will compare addresses


                C1=c3;

                SOP(c1==c3);

                SOP(c1.equals(c3)); // equals method not overridden hence will compare addresses

                SOP(c1);

                SOP(c2);

                SOP(c3);

                // for understanding

                String s1 = new String("java");

                String s2 = new String("java");

                SOP(s1==s2);// false

                SOP(s1.equals(s2));// true

        }

}
```

****

//overriding equals method – simple way

Package com.qspiders.pack1;

Class Apple

{

       Int wt;

       Apple(int wt)

       {

```java
        This.wt = wt;

    }

    Public String toString()

    {

        Return "wt = " + wt;

    }

    Public boolean equals(Object O)

    {

        Return this.wt == ((Apple)O).wt;

    }

}

Public class Demo90

{

    PSVM()

    {

        Apple a1 = new Apple(100);

        Apple a2 = new Apple(120);

        Apple a3 = new Apple(100);

        SOP(a1); // prints address

        SOP(a2);// prints address

        SOP(a3); // prints address

        SOP(a1==a2); // false – comparing 2 addresses

        SOP(a1==a3); // false – comparing 2 addresses

        SOP("----------";

        SOP(a1.equals(a2)); // false – comparing 2 addresses as we have not overridden
        equals method

        SOP(a1.equals(a3)); // false – comparing 2 addresses as we have not overridden
        equals method

    }

}
```

****

//overriding equals method – simple way

```
Package com.qspiders.pack1;

Class Apple

{

        Int wt;

        Apple(int wt)

        {

                This.wt = wt;

        }

        Public String toString()

        {

                Return "wt = " + wt;

        }

        Public boolean equals(Object O)

        {

                If (O==null)

                Return false;

                If (this==null)

                Return true;

                If(O instanceof Apple)

                {

                        If (this.wt == ((Apple)O).wt)

                        Return true;

                }

                Return false;

        }

}

Public class Demo91

{

        PSVM()
```

```
{
        Apple a1 = new Apple(100);

        Apple a2 = new Apple(120);

        Apple a3 = new Apple(100);

        SOP(a1); // prints address

        SOP(a2);// prints address

        SOP(a3); // prints address

        SOP(a1==a2); // false – comparing 2 addresses

        SOP(a1==a3); // false – comparing 2 addresses

        SOP("----------";

        SOP(a1.equals(null));// false

        SOP(a1.equals(a1)); // true – comparing 2 values (wt) as we have overridden equals
        method

        SOP(a1.equals(a3)); // // true – comparing 2 values (wt) as we have overridden
        equals method

        SOP(a1.equals(a2)); // // false – comparing 2 values (wt) as we have overridden
        equals method

        SOP(a1.equals("Hello")); // // false – different Object handled

        }
}
```

25th Oct 2016

Difference between == and .equals()

- == is a relational equality operator which is used
  1. To compare two primitive variable where it compare the value like x==y or x==10
  2. To compare two reference variable where is compares the address of the object

- Equals () is non-static method of object class which by default compares the address. If we override we can compare the value
  Ex: String class, wrapper classes.

// Program to generate address of an object

```
Package com.qspiders.pack1;
Class Biscuit
{
}
Public class AddressGenerateDemo
{
        PSVM()
        {
                Biscuit b1 = new Biscuit();
                SOP(b1);
                SOP(b1.hashcode());
                Class cls = b1.getclass();
                SOP(cls.getname() + "@" + b1.hashcode());
                SOP(cls.getname()+"@"+Integer.toHexString(b1.hashcode());

        }
}
```

                        ****
```
Package com.qspiders.pack1;
Class Orange
{
        Int wt;

        Orange (int wt)
        {
                This.wt = wt;
        }

        Public String toString()
        {
                Return "wt= " + wt;
        }

        Public bolen equals(Object o)
        {
                If(o==null)
                        Return false;
                If(this==o)
                        Return true;
                Is(this.wt == ((Orange)o.wt)
                        Return true;
                        Return false;
        }
        Public int hashCode()
        {
                Return wt*7*11*5;
```

```
                }
        }
        Public class Demo3
        {
                PSVM()
                {
                        Orange o1 = new Orange(100);
                        Orange o2 = new Orange(120);
                        Orange o3 = new Orange(100);

                        SOP(o1);
                        SOP(o2);
                        SOP(o3);

                        SOP(o1==o3);
                        SOP(o1.equals());
                        SOP(o1.hashCode());
                        SOP(o2.hashCode());
                        SOP(o3.hashCode());
                }
        }
```

Note: reference variable o1 and o3 will have same hashcode after overriding hashcode(). That does not mean both refer to same object

Note: When two objects (of same type) are equal, their hashcode should be same. But when hashcode of two objects (different) are same then the objects need not be equal. In otherwords tow different Objects can be in a same bucket or locker (can have same hashcode).
i.e when you override equals method of a class, we should also override hashcode method

26th Oct 2016

Wrapper classes: To convert primitive data type into an object, we use Byte, Short, Integer, Long, Float, Double, Character, Boolean.

Since they wrap around primitive it is called wrapper classes

- Converting primitive data type into an object is called as auto boxing (compiler does it)
- Converting an object (which is converted from auto boxing) to primitive type is called as unboxing. (developer does it)

In both the cases we use wrapper class. For every primitive type, respective wrapper class is given by JDK. Every wrapper class will have overloaded constructor (except character class). They do not use default constructor in wrapper class.

199

Note:

1. Generally whenever a reference variable is printed, the address of the object will be printed. The address is usually represented with fully qualified class name – fullyqualifiedclassname@hexadecimal address of the object

2. Whenever a reference variable of any wrapper class is used or printed then it prints the primitive data of the wrapper class instead of address. Because in every wrapper class toString() method of the object class has been overridden to display or return the primitive data instead of address

```
Public lass Demo91
{
        PSVM()
        {
                Int I = 10;
                SOP(i);
                Integer intobj1 = new Integer(i); // boxing operation
                SOP(intobj1);
                Integer intobj2 = new Integer(100); // boxing operation
                SOP(intobj2);
                Integer intobj3 = new Integer("200"); // boxing operation
                SOP(intobj3);
                Integer intobj4 = 300;
                SOP(intobj4);
        }
}
```

o/p –   10

         10

         100

         200

         300

// Integer intobj2 = 100; (this is correct)

// Integer intobj2 = i; (this is correct) – Auto boxing from JDK 1.5 onwards
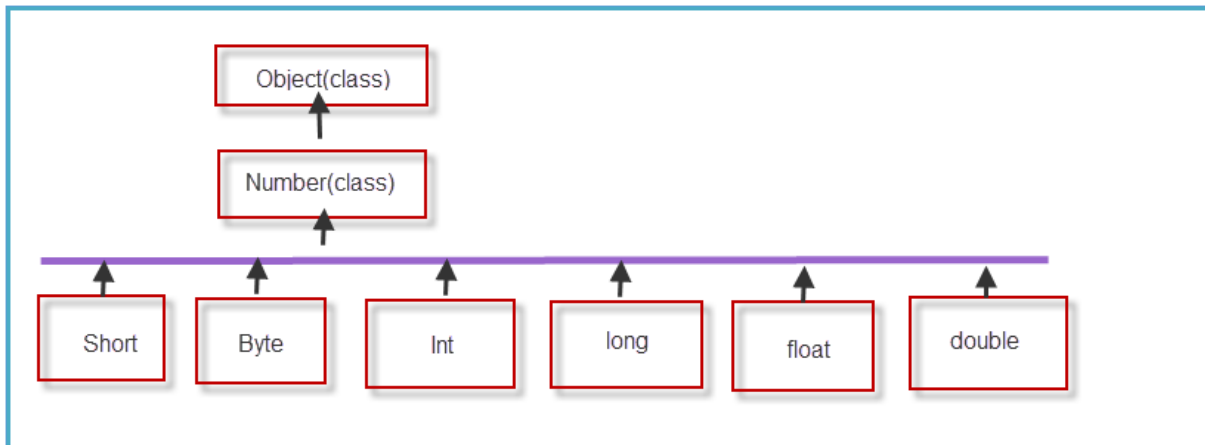
Wrapper class:

- In java.lang
- Two constructors for each wrapper class except character class (only one constructor)
- toString(), equals and hashCode() method is overridden

Every number related wrapper class inherit from super **class Number** (Number class is on abstract class). It is inherited from object class. The **Number class** methods are overridden in every wrapper class which deals with numbers

Public class Demo92

{

    PSVM()

    {

        Int i=100;

        Integer intobj = new Integer(i); // boxing operation

        SOP(intobj);

        Int j = intobj.intvalue();// unboxing operation

        SOP(j);

    }

}

Boxing operation: by passing primitive value to wrapper class constructor of respective type

Unboxing: by calling non static method of boxed variable

```java
public class Demo93 {

    public static void main(String[] args) {

        int i=100;

        Integer intObj=new Integer(i); // boxing operation

        System.out.println(intObj);

        int j=intObj.intValue(); //unboxing operation

        System.out.println(j);

        double d=intObj.doubleValue();//unboxing and autowidening

        System.out.println(d);

        //1st way

        Double dd=new Double(d);//boxing

        System.out.println(dd);

        //2nd way

        Double dd1=new Double("12.56");//boxing

        System.out.println(dd1);

        //3rd way

        Double dd1=d;//boxing

        System.out.println(dd1);

        System.out.println("unboxing");

        int i1=dd1.intValue();//unboxing and
```

```java
//narrowing
    System.out.println(i1);


    System.out.println("unboxing and explicit narrowing");
    int i2=(int)dd1.doubleValue();
    System.out.println(i2);




   byte b=123;
                Long ll=new Long(i);//i autowidening
                Long ll1=new Long(b);//b autowidening
                System.out.println(ll);
                System.out.println(ll1);




                long l=100;
                Byte bb=new Byte((byte)l);
                System.out.println(bb);




    System.out.println("converting object to String");
    String s1=ll.toString();
    System.out.println(s1);
    System.out.println("----");


   Double dd=new Double(100);
        String s=dd.toString();//object to String
                Double.toString(10.22);//primitive to String
                Long.toString(100l);//ditto
```

```java
        Byte.toString((byte)100);


    //String s2=ll; cannot convert from Long to

       //  String, assigning ll address to s2


    //Add two string containing numbers and put in another string



    String s11="123";
    String s22="456";


    String s33=Integer.toString(Integer.parseInt(s11)+Integer.parseInt(s22));


    System.out.println(s33);




    Integer ii=new Integer(123);
    Integer ii2=ii;//assigning the address
    System.out.println(ii==ii2);//true
    System.out.println(ii2);


    Integer ii3=Integer.valueOf(ii);//returning an new  Integer object
    System.out.println(ii==ii3);//false
        }
}
```

27th Oct 2016

Int to string convertion:

Integer.toString(10); - - here toString() method is static and not the one which belongs to Obejct class. The one which belongs to Object class is non static

o/p – it converts 10 to "10"

String to int (primitive) type:

Integer.parseint("10");

o/p – it sonverst "10" to 10;

Object to String:

S1 = o.toString();

Interview questions:

1. how do you convert int to a String
2. how do you convert String to int
3. how do you convert primitive to a object
4. how do you convert object to primitive
5. how do you convert object to a String
6. how do you convert String to char – we can do by using (reference variable.charat[])
7. how do you convert char to String – we can do by using Character.toString()

****

Preference execution of arguments

Package com.qspiders.pack1;

Class C

{

    Void test(double d)

    {

        SOP("double argument method");

    }

    Void test(Integer intobj)

    {

        SOP("Integer argument method");

    }

```
Void test(Number num)

{

        SOP("Number argument method");

}

Void test(Object obj)

{

        SOP("object argument method");

}

Void test(int i)

{

        SOP("double argument method");

}

}

Class Demo95

{

        PSVM()

        {

                C c1 = new C();

                C1.test(100);

        }

}
```

o/p – int argument method – try this program by commenting one one method every time

here the preference of execution is ---same type, auto widening, wrapper classes, Number class nd Object class

Object class

Number class

Integer

double

int

String class - VVVVVVV IMP

// basically values can be assigned in two ways

Package com.qspiders.StringDemo;

Public class Demo95

{

      PSVM()

      {

            String s1 = "Hello"; - 1st way

            String s2 = new String ("Bangalore"); - 2nd way

            SOP(s1 + " " + s2);

      }

}

               *****

Packge com.qspiders.StringDemo;

Public class Demo96

{

      PSVM()

      {

            String s1 = "java";

            String s2 = new String("Developer");

            String s3 = "java";

            String s4 = new String("Developer");

            SOP(s1==s3);// true

            SOP(s2==s4);// false

            String s5 = new String("java");

            SOP(s1==s5);// - false

      }

}

o/p – true

        false

        false

Please see below diagram for explanation.
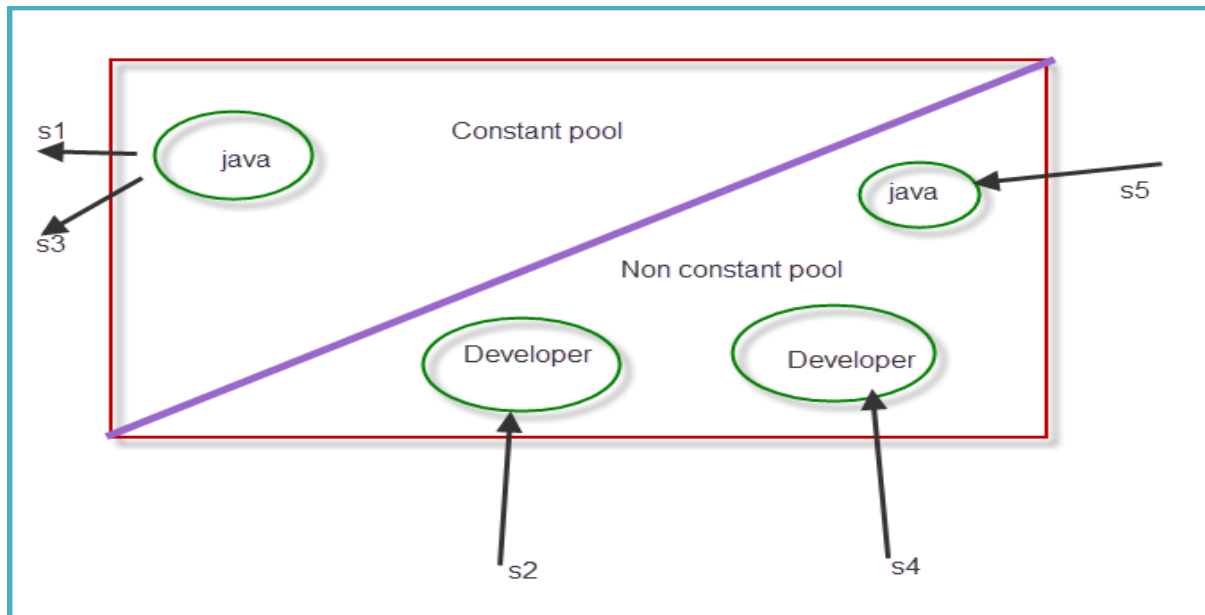
Direct assignments are stored in constant pool.

When we use 'new' keyword new object will be created and stored in non-constant pool

In constant pool duplicates are not allowed. When s1 is created "java" object is created. Again when we do s3="java" this will not create new object instead refers to same object where s1 is pointing.

It means s1 and s3 are referring to same object

String objects are created in 2 ways

1. new operator
2. assignment
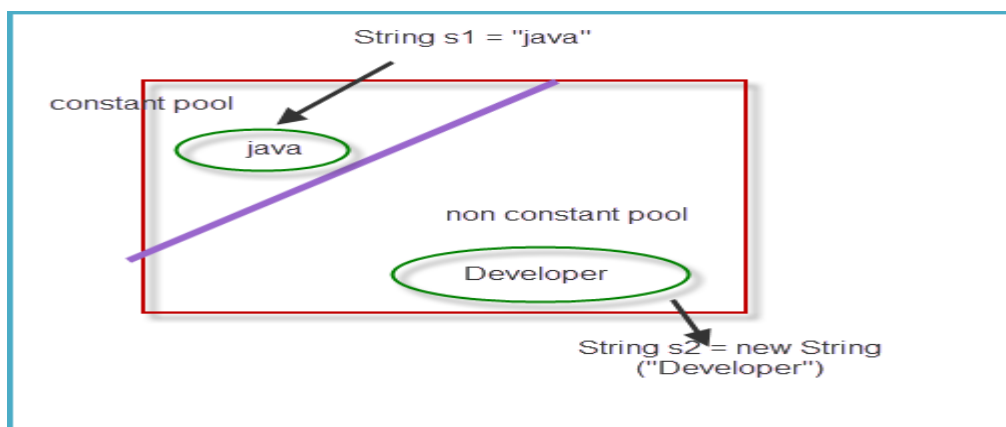
String class object created without new operator will be created in constant pool. (this happens only for String class and wrapper class)

If we have one more object with same value without new operator then the variable will point towards the already existing object.(In constant pool you can have only constant object)

String s1 = "java";

String s2 = new String("developer");

Like String s3 = "java" // not allowed to create a separate (duplicate) object. It refers to current object if already existing

But   String s4 = new String ("Developer");

        String s5 = new String ("java");
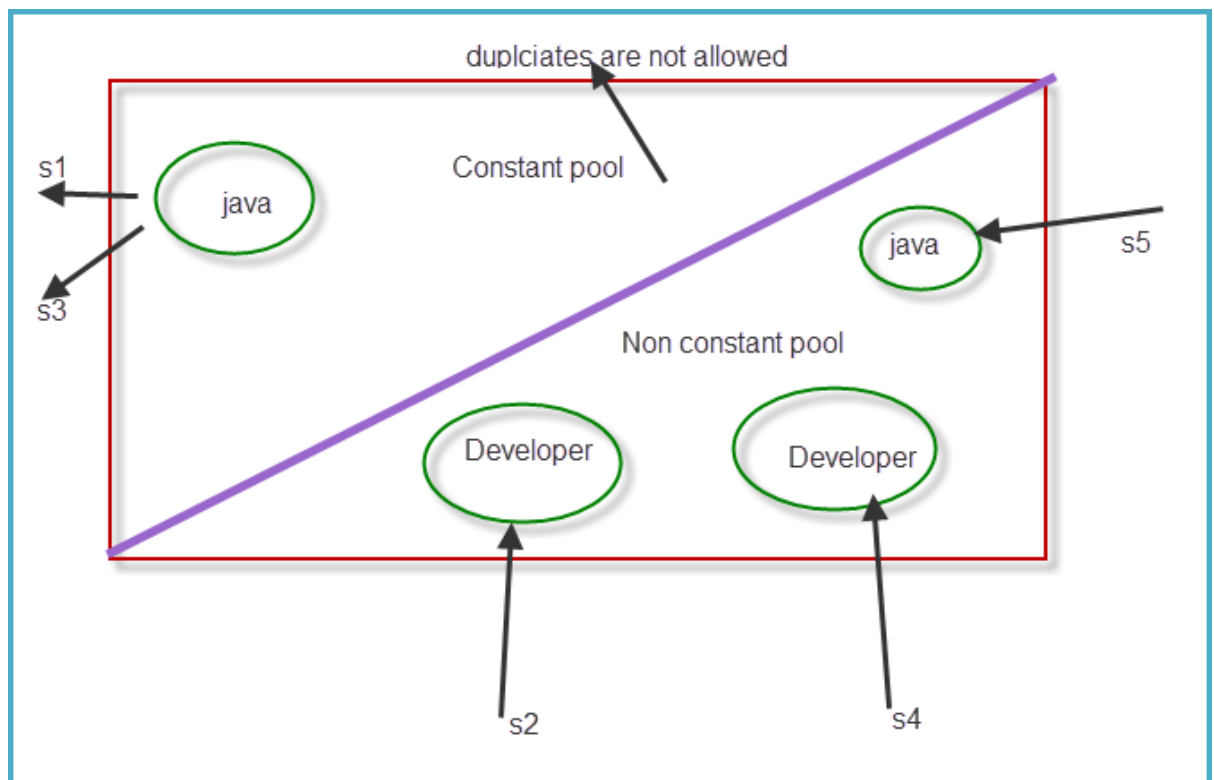
S1==s3; true

S2==s4; false

S1==s5 false

S3.equals(s5) // to compare values in the object, use this.

Note: To compare value of Strings we use equals method

S1==s3 // address comparison

S1.equals(s3) // value comparison

Package com.qspiders.StringDemo;

Public class Demo97

{

    PSVM()

    {

        String s1 = "java";

        String s2 = new String ("Developer");

        String s1 = "java";

        String s2 = new String ("Developer");

        String s1 = new String ("java");

        SOP(s1.equals(s3)); // comparing values - true

        SOP(s2.equals(s4)); // comparing values - true

        SOP(s1.equals(s5)); // comparing values - true

        SOP(s1.equals(s2)); // comparing values - false

    }

}

Imp: equals method is overridden in the String class to compare values in Eclipse

s1 + s2

s1 + "Hello"   **non constant pool – these will creates new objects**

"Hello" + s2

"Hello" + "Fellow" --→ constant pool

```
Package com.qspiders.StringclassDemo

Public class Demo98

{

        PSVM()

        {

                String s1 = "java";

                String s2 = "Developer";

                String s1 = s1 + s2;

                SOP("s3= " + s3);

                String s4 = "javaDeveloper";

                SOP("s4= " + s4);

                SOP(s3==s4);// false

                String s5 = "java" + "Developer";

                SOP(s3==s5); false

                SOP(s4==s5); true

                String s6 = s1 + "Developer";

                SOP(s3==s6);

                SOP(s4==s6);

                SOP(s4.quals(s6));

        }

}
```
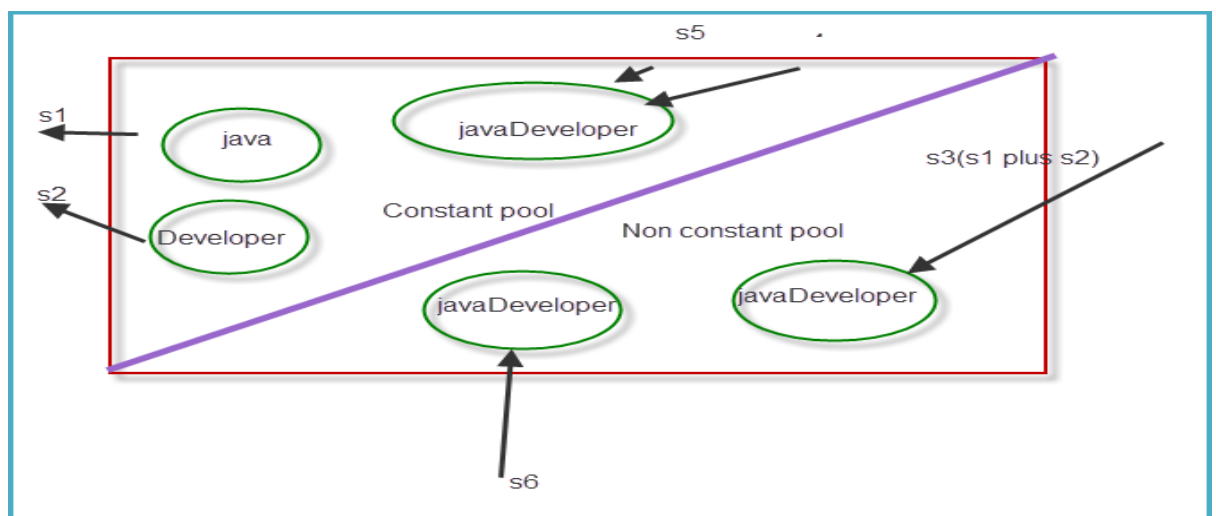
- String class is anon mutable class. It means if we try to update it will not update old object. it will create a new object and assign the new value

```java
public class Demo99
{
        PSVM()
        {
                String s1 = "Hellollo";
                SOP (s1.lenght());
                SOP (s1.concat("Hello"));
                SOP(s1);

                String s2 = new String();
                SOP(s2.lenght);
                SOP(s2.isEmpty);

                String s3 = new String(""); // string
                SOP(s3.lenght);
                SOP(s3.isEmpty);

                String s4 = new String(" ");// space character is there in parenthesis
                SOP(s3.lenght);
                SOP(s3.isEmpty);

                S2=s2.cancat("Bangalore");
                SOP(s2.isEmpty);
                SOP(s1.contains("Dev");

                S1.equals(s2);
                S1.equals(null);
                S1.equals("Hello");
                S1.equals(new String("Hello"));
                S1.equals(s1);
                SOP("Hello".equals("Hello");
                SOP(new String("Hello").equals(new String("Hello"));
                SOP("".equals(""));
                S1="Hello";
                SOP(s1.equalsIgonrecase("heLLO");

                S1 = "javaDeveloper"
                SOP(s1.indexof('D');
                SOP(s1.indexof('Deve');
                SOP(s1.indexof('Z');
                SOP(s1.indexof('e');
                SOP(s1.indexof('e', 0);
```

```
SOP(s1.indexof('e', 6);
SOP(s1.indexof('e', 8);

SOP(s1.replace("e", "E");
SOP(s1.replace("e", "java");
SOP(s1.replace("java", "eg");
SOP(s1.replace("ex", "x");
```

replace all occurrences

```
SOP(s1.replaceAll("l", "L")); // we use this for regular expession

SOP(s1.charAt(0));

For(int i=0;i<s1.length(); i++);

SOP(s1.charAt(i));
```

Printing s String in reverse order – VVVVVVVVVVV IMP

```
For(int i=s1.length(); i>=0; i--);

SOP(s1.charAt(i));

SOP(s1.charAt(0));
SOP(s1.charAt(s1.length()-1));
Char ch = s1.charAt(1);
SOP(ch);
S1= "Jackand Jill";
For (int i=0; i<s1.lenght();i++)
SOP(s1.charAt(i));
SOP("--------");

For (int i<s1.length()-1;i>=0;i--)
SOP(s1.charAt(i));
SOP("--------");
```

Reversing a String– VVVVVVVVVVVVVVVVVVVV IMP (1st method)

```
String s3 = "";

For (int i=s1.length()-1;i>0;i--)

S3 = s3 + s1.charAt(i);

SOP(s3);
```

```
SOP(s1.startswith("java"));

SOP(s1.startswith("dev"));

SOP(s1.endswith("per"));


SOP("DA".compareTo("DA"));
SOP("DA".compareTo("GA"));// when not same prints int value with the
difference of their Unicode values

SOP(s1.toUpperCase());
SOP(s1.toLowerCase());

SOP("sum of 4 and 5 is " + 4 + 5);
SOP("sum of 4 and 5 is " + (4 + 5));
SOP(4 + 5 + "is sum of 4 and 5");
```

Reversing a String– VVVVVVVVVVVVVVVVVVVV IMP (2nd method)

```
Char chars[] = s1.toCharArray();
Strign s3 = "";
For(int I = chars.length-1;i>=0;i--)
S3 = s3 + chars[i];
SOP(s3);

S3 = "JackandJill";
Char arr[] = s3.tocharArray();
For (int i=0; i<arr.length;i++)
{

        //prints char in array
        SOP(arr[i] + "\t");
        // print the Unicode value of the char
        SOP((int)arr[i] + "\t");
        // print the next char of the present char. It means if A is there print
        B. if J is there print K.
        SOP((char) (arr[i]+1) + "\t");
        SOP((++arr[i]));
}
```

1st Nov 2016
```
            SOP(new String("Hello").length());
            SOP("Bangalore".length());
            String s4 = "      Hello        ";
            SOP(s4.length());
            S4 = s4.trim();
            SOP(s4.length());
```

```
//Alternate way

SOP(s4.trim().length());
S1 = "javaDeveloper';
SOP(s1.subString(4));
SOP(s1.subString(4, s1.length())); //
```

SOP(4, 11);
here 4 is index (index numbering starts from 0) and 11 is the position
(position numbering starts from 1)

```
String str = "welcome subhankar to stackoverflow.com";

SOP(str.subString(8, str.indexof(" ", 8);


String s5 = "This is java class");
String arr1[] = s5.split(" ");

For(int i=arr1.length-1;i>=0;i--");
        {
                SOP(arr1[i] + " ");
        }
    }
}
}
```

2nd Nov 2016

What is StringBuffer? – StringBuffer is a mutable class. It means we can update the value of object

What is the difference b/w StringBuffer and StringBuilder? – StringBuffer and StringBuilder both are identical except that StringBuffer is synchronized (thread safe) and StringBuilder is not synchronized (not threadsafe)

What is the difference b/w String and StringBuffer? – String is a non-mutable class. It means we cannot update the value of object. When we try to assign new value every time new object will gets created. StringBuffer is a mutable class. It means we can update the value of object

```
package com.qspiders.stringdemo;

public class Demo101 {

        public static void main(String[] args) {

                StringBuffer sb1=new StringBuffer();

                //sb1="Hello";// not possible since "" double quotes belong to //String class
```

```java
//i.e like assigning String class object to StringBuffer

sb1.append("Hello");

StringBuffer sb2=new StringBuffer("Bangalore");

System.out.println(sb1 + " " + sb2);

System.out.println(sb1.reverse());//updates the existing object
System.out.println(sb1);//same as above

System.out.println(sb1.reverse());

StringBuffer sb3=sb2.reverse() ;

System.out.println(sb1);
System.out.println(sb2);
System.out.println(sb3);
System.out.println(sb2==sb3);//true

sb1.setLength(0);
//sb1.delete(0, sb1.length());//alternate
System.out.println(sb1.length());
System.out.println("--------");

StringBuffer sb4=new StringBuffer();
System.out.println(sb4.length());
System.out.println("--------");

sb4.append("JavaDeveloper");
sb4.insert(4, "haha");
```

```java
        System.out.println(sb4);//JavahahaDeveloper

                        sb4.delete(4, 6);

                        System.out.println(sb4);//JavahaDeveloper

                        System.out.println(sb4.deleteCharAt(4));//5th char 'h' is deleted


        StringBuilder sb5=new StringBuilder("Selenium");

                        System.out.println(sb5);

                        StringBuilder sbl;

        System.out.println(sbl=sb5.insert(4, "Haha"));

                        System.out.println(sb5);


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

                        System.out.println(sb5==sbl);

                        System.out.println(sb5.equals(sbl));//true

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

                        //equals method not overridden
    //compares address

                        System.out.println(sb5.reverse());

                        sb5.reverse();

                        System.out.println(sb5.delete(3, 7));


                        //String s=sb1//not possible since they are of diff class
                    //assiging StringBuffer content to String class

                        String s=sb1.toString();

                        //assiging String to StringBuffer class

                        StringBuffer sb6=new StringBuffer("Hello");//1st way

                        sb6.append("Bangalore");//2nd way

    }
}
```

String class of java.lang(package) is used to store the String object.

- A string object can be created either by using new keyword or direct assignment of the string literal
- Whenever a string object is created with help of string literal the objects are created in the constant pool . The constant pool doesn't allow duplicate objects.
- If the objects are created with new keyword then the objects will be created in non constant pool. This pool allows duplicate objects
- If same object is being created in the constant pool the object will be created only once and all the reference variables will be pointing to the same object. However in non constant pools the reference variable will be pointing to different object(new object) of the same string.
- The string object is a immutable object since we cannot change the object's value
- In the String class toString() method and equals() method are overridden.
- The comparison of two String object is done based on value of the object.(First address is compared if it returns false then value is compared)
- In order to compare wrt String values use equals() method
- From JDK1.5 onwards two additional classes called StringBuilder & StringBuffer class have been introduced. This class can be used to create String object. Using any of these classes we can create object with the help of new keyword.
- Both these Classes add an additional buffer of 16 characters to the String object
- The methods of StringBuffer classes give synchronized methods
- reverse ,insert,delete methods are available in StringBuilder & StringBuffer class

==Reversing a String– VVVVVVVVVVVVVVVVVVVVV IMP (3rd method)==

String s1 = "Hello"

StringBuffer sb1 = new StringBuffer(s1);

Sb1.return();

String s1 = sb1.toString();

==Reversing a String– VVVVVVVVVVVVVVVVVVVVV IMP (4th method)==

S1 = new StringBuffer(s1).reverse().toString();

Interview questions

1. Reverse a String
2. Reverse a String wihout using inbuilt functions (use 1st and 2nd)
3. Reverse a number. please see below programs

Reversing a number– VVVVVVVVVVVVVVVVVVVV IMP (1st method)

Int I = 1234;

String s = Integer.toString();

StringBuffer sb1 = new StringBuffer(s);

       Sb1 = sb1.reverse90;   Multiple lines program

       S = sb1.toString();

       I = Integer.parseInt(s);

I = Integer.parseInt(new StringBuffer(Integer.toString(i)).reverse().toString(); - single line code

Reversing a number– VVVVVVVVVVVVVVVVVVVV IMP (2nd method)

```
Public class reverseNUmberwithoutfunction

{

        PSVM()

        {

                SOP(reverse(1234));

        }

        Static int reverse (int i)

        {

                Int reverse = 0;

                While (i>0)

                {

                        Int remainder = i%10;

                        Reverse = remainder + reverse*10;

                        I = i/10;

                }

                Return reverse;

        }

}
```

Test 2 questions.

what are the difference b/w static and instance members?Mention 3 differences

what variables are called as fields?

what variables should be compulsorily initialized?

what variables will be initialized to default values if not initalized?

what is constant in java?

what is blank final and should it be initialized, if so, where?

wap to print

  o/p should be print:"World's End"

wap to print

  o/p \n

what are operators and diff kind of operators based on operation?

what is the o/p of

  System.out.println('A'+'B');

  System.out.println('A'+ "B");

  System.out.println('A'+ 100);

  System.out.println("A"+ 100);

what happens ?

    boolean res=x!=y;

    a)System.out.println('res');

    b)System.out.println(res);

    c)System.out.println("res");

    d)System.out.println(TRUE);

    e)System.out.println(true);

what is ==?explain fully

what happens?

```
Orange o1=new Orange();

Orange o2=new Orange();
```

```
a) System.out.println(o2);

b) System.out.println(o1==o2);

c) System.out.println(o1!=o2);

d) System.out.println(o1>=o2);
```

what is the o/p?

```
int x=10,y=8;
```

```
System.out.println(x & y);

System.out.println(x | y);

System.out.println(x ^ y);
```

wap using nested conditional operator.

Take marks as input .classify as fail(0-<50),pass(50-60),fc(61-74)

distinction(75-100)and invalid

3rd Nov 2016

Arrays:

Arrays are objects in java that stores multiple values of same type. Arrays can hold either primitive or object references. But the array itself is an object even if the array is declared to hold primitive elements

1. arraytype[] arrayName = new arraytype[size];
2. int arrayName[]; // here you shouldn't mention the size
3. int array[] = {value 1, value 2, value 3};

   or

   int array[] = new int[]{value 1, value 2, value 3};

   Examples:

   a. int age[] = new int[3];
   b. int[] marks = new int[4];
   c. int scores[];

      scores = new int[5];

   d. int[] marks = {60, 50, 40, 90};

Note: If you are declaring array & constructing the array in some line you should declare the size

Index in an array starts with zero (0). If the size is 3 means the indexes are 0, 1 and 2.

Array is fixed in size. Though you can create array object dynamically, once an array object is created, we cannot increase or decrease the size

Empty array : array with zero (0) size.

Ex: int arr[] = new int[0];

    Or

    Int arr[] = {};

        ****

Public class Demo12

{

    PSVM()

    {

        Int arr[] = new int[3];

        Arr[0]=10;

        Arr[1]=20;

```
            Arr[2] = 30;

            For (int i=0; i<arr.lenght;i++)

            {

                    SOP(arr[i]);

            }

            String arr1[] = new String [3];

            Arr[0]="I";

            Arr[1]="Love";

            Arr[2] = "java";

            For (String s: arr1) // for eachloop or enhanced for loop

            {

                    SOP(s);

            }

    }

}
```

Arrays can be classified for our convenience as primitive array and non-primitive array.
Ex: primitive array
Int arr[] = new int[5];

Non primitive array
Apple apples[] = new Apple[5];
Default values in array will be same as the default of the type declared.

i.e. if int arr[] then arr[0]. Will be zero(0) as default for int

if Apple apples[] then apples[0] will be null as default for Apple is null.

```
Public class Demo103
{
    PSVM()
    {
            //1st way to assign values to array and print it

            Int arr[] = new int[5];
            Arr[0] = 10;
            Arr[1] = 20;
            Arr[2] = 30;
            Arr[3] = 40;
            Arr[4] = 50;
```

```
                    For (int i=0; i<arr.lenght; i++)
                    {
                            SOP(arr[i]);
                    }

                    //2nd way to assign values to array and print it

                    Int arr1[] = new int[5];
                    Int val = 10;
                    For (int j=0; j<arr1.lenght;j++)
                    {
                            Arr1[J] = val;
                            Val = val+10;
                    }

                    For (int j=0; j<arr1.lenght;j++)
                    {
                            SOP(arr[j]);
                    }


                    //3rd  way to assign values to array and print it
                    Int arr2[] = {10,20,30,40,50};
                    For(int a; arr2)
                    {
                            SOP(a);
                    }
            }
        }
```

VV IMP note: in Arrays, length is a field where in String length is the method

Array object

- Int i[] = new int[5];

Array variable

Int i1[] = new int[0];

- Int b[] = {};
  Int b1[] = {10, 20, 30, 40};
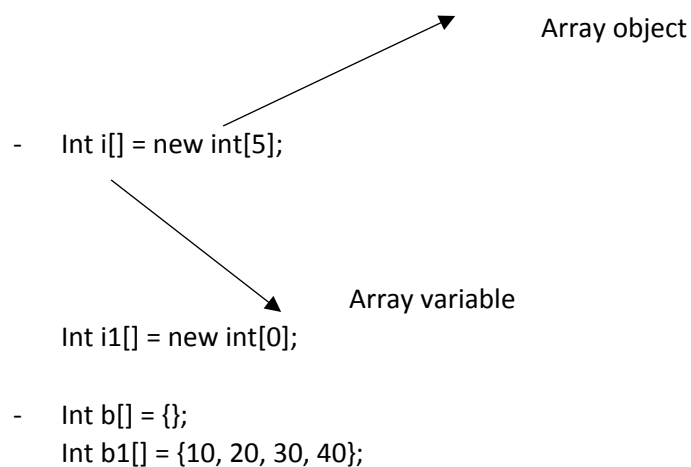
Note: If we access element where index is not there, it will give "ArrayIndexoutofBoundsException".

```
Public class Demo104
{
        PSVM()
        {
                Int arr[] = {};
                Char cArr[] = {}:
                double dArr[] = {}:
                String sArr[] = {}:


                SOP(arr);// address
                SOP(cArr);// nothing it will print
                SOP(dArr);// address
                SOP(sArr);// address
        }
}
```

All the types prints the reference variables and addresses except char. Char will not print anything

Array's don't override toString(), and so you see the result of the default object.toString implementation.

Int arr[] = {};

SOP(arr); // print the address of array object as we are printing reference variable

1.what is difference b/w JDK,JRE and JVM?

2.what is a variable?

3.what is a method?

4.difference b/w method and constructor?

5.what is constant variable?

6.what is difference b/w static and non static members?

7.what is package?

8.what are the access specifiers?

9.what are the non access specifiers?

10.what are the primitive datatypes?

11.what is abstraction?

12.what is difference b/w abstract class and interface?

13.whar is polymorphism?

14.what is difference b/w method overloading and overriding?

15.what is upcasting and downcasting?

16.what is encapulation?

17.what is inheritance?

18.what is difference b/w static binding and dynamic binding?

   programs

1.WAP to print fibonacci series?

2.WAP to print factorial of a number?

3.WAP to print factorial of a number using recursion?

4.WAP to reverse a number?

5.WAP to reverse a string?

6.WAP to print division of two numbers without using divide(/) operator?

7.WAP to print Multiplication of two numbers without using divide(*) operator?

8.WAP to find the area of a circle,square and rectangle?

9.WAP to reverse an string without using inbuild function?

10.WAP to reverse numbers without using inbuild function?

<mark>4th Nov 2016</mark>

// In array if you don't assign any value, will contain default value of the type declared

Class Apple

{

}

Public class Demo105

{

    PSVM()

    {

        Int arr[] = new int[3];

        Arr[0] = 10;

        For(int i: arr)

        {

            SOP(i);

        }

        String arr1[] = new String[3];

        Arr1[0] = "Hello";
        For(String s: arr1)

        {

            SOP(s);

        }

        // Apple array

        Apple arr2[] = new Apple[3];

        Apple a2 = new Apple(); // Apple object referred by a2 reference variable

        Arr2[0] = new Apple(); // assigning Apple object

        Arr2[1] = a2; // assigning Apple reference variable

        For(Apple a: arr2)

        {

            SOP(a);

        }

    }   }

****

Public class Demo106

{

    PSVM()

    {

        Int arr[] = {12, 21, (int) 10.78, (int) 12.45}; // 10.78 , 12.45 – explicit narrowing

        For (int val:arr)

        {

            SOP(val);

        }

        SOP("-------");

        Double arr1[] = {12.66, 7, 8, 6, 7}; // 7,8,6,7 – auto widening

        For (double val : arr1)

        {

            SOP(val);

        }

    }

}

        ****

Public class Demo107

{

    PSVM()

    {

        Int arr1[] = {10, 'a', 'v', 4,6,7,8,'B',6,'Z'};

        For(int i: arr1)

        {

            If((i>65&&i<=90)||(i>=97&&i<=122)

            {

                SOP((char)I + "" + i);

            }

        }   }   }

- 'Arrays' is an utility class

How to swap 2 numbers

1- Using temp variable

X= 10; y = 20;

Temp = x;

X=y;

Y=temp;

2 – without using temp

x= x+y;

y = x – y;

x = x-y

****

imprt jav.util.Arrays; - // 'Arrays' is an utility class

Public Class Demo108

{

  PSVM()

  {

    Int a[] = {10,3,47,04,8};

    SOP("Array element before sort");

    // 1st way to print the values

    For(int x:a)

    {

      SOP(a);

    }

    // 2dn way

    SOP(Arrays.tostring(a));

    SOP("Array element after sort");

    Arrays.sort(a);// sorting of array

    // 1st way to print the values – after sort

```
            For(int x:a)

            {

                    SOP(a);

            }

            // 2dn way – after sort

            SOP(Arrays.tostring(a));

    }

}
```

Note: Java is pass by value. Even when you pass reference variable address gets copied

// Demo for pass by value and demo for using ellipses

Package com.qspiders.arrayDemo;

Class Orange

{

        Int i=123;

}

Public class Demo

{

        PSVM()

        {

                Int x = 100; y = 200;

                SOP(add(x,y)); // values of x and y are passed to n1 & n2

                Print(new Ornage()); // object is assigned

                Orange o1 = new Orange();

                Print(o1); // address of o1 is copied to 'o' in method definition

                Int marks[] = {70,80,90};

                SOP(add(marks)); // address of marks (reference variable) is passed method parameter passing array

                SOP(add1(marks)); // passing array

                SOP(add(70,80,90)); // passing variable no of arguments (var args)

                }

```
Static int add (int n1, int n2)

{

        n1 = n1+n2;

        return n1;

}

Static void print (Orange o)

{

        SOP(o.i);

}

Static int add(int nos[])

{

        Int sum = 0;

        For (int n:nos)

        Sum = sum+n;

        Return sum;

}

Static int add1(int…nos)

{

        Int sum = 0;

        For (int n:nos)

        Sum = sum+n;

        Return sum;

}

}
```

5th Nov 2016

2 dimensional array (2-D-array):

Two dimensional array: It is an array of arrays

Ex:

10	20	30
40	50	60
70	80	90

Public class Demo110

{

    PSVM()

    {

        Int arr[][] = new int[4][3];

        SOP(arr.length); // gives no of rows (length)

        SOP(arr[0].length); // gives columns of row1;

        SOP(arr[1].length); // gives columns of row2;

        SOP(arr[2].length); // gives columns of row3;

        SOP(arr[3].length); // gives columns of row4;

    }

}

Interview question: How do you find size of array?

Ans: create an array and reference variable.length will fives the size of the array

               ****

Public class Demo111

{

    PSVM()

    {

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

        Int val = 10;

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

        {

```
                        For (int j=0; j<arr[i].length;j++)

                        {

                                SOprint(arr[i][j]); // if we print only SOP(arr[i]); - it will give address
because it is representing an array. There is no column. Only it row it referes(array)

                        }

                        SOP();

                }

        }

}
```

- What is jagged array: it is an array of arrays where the size of each sub array can differ

(Or)

In simple words, each row can contain different no of columns

****

Assigning & printing values in jagged array

```
Public class Demo112

{

        PSVM()

        {

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

                        (or)

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

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

                {

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

                        {

                                SOprint(arr[i][j]);

                        }

                        SOP();

                }
```

```
            SOP("----------------enhanced loop----------------");

            For(int a[]:arr)

            {

                    For(int i:a)

                    {

                            SOprint(i);

                    }

                    SOP();

            }

    }

}
```

****

```
Public class Demo113 // creating jagged array in another way

{

    PSVM()

    {

            Int arr[][] = new int [4][];

            Arr[0] = new int [4];

            Arr[1] = new int [2];

            Arr[2] = new int [4];

            Arr[3] = new int [1];

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

            {

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

                    {

                            SOprint(arr[i][j]);

                    }

                    SOP();

            }
```

```
Int arr1[][] = new int [4][];

Arr[0] = new int [] {3,5,7,9};

Arr[1] = new int [] {4,2};

Arr[2] = new int [] {5,7,8,6};

Arr[3] = new int []{6} ;

For (int i=0; i<arr1.length; i++)

{

        For(int j=0; j<arr1[i].length;j++)

        {

                SOprint(arr1[i][j]);

        }

        SOP();

}

}
}
```

Questions:

1 – How do you remove a value from an array? – by setting it to default value

2 – Sort an array using Bubble sort

3 – Write a program to reverse an array

4 – Write a program to reverse first half and second half of an array

5 - Write a program to transpose a Matrix

6 – Print values of an array at odd indexes

7 – Print values of an array at even indexes

8 - Write a program to add 2 arrays

9 - Write a program to subtract 2 arrays

10 - Write a program to demonstrate array multiplication

11 – Compare 2 arrays and print if they are different (both single and 2 –Dimensional arrays)

Exceptions:

Exception is an event that gets triggered when JVM is not able to execute a statement. It can handled using try – catch block

```
Package com.qspiders.exceptionDemo;

Public class Demo114
{
    PSVM()
    {
        Int I = 10;
        Int j;


        Try
        {
            SOP("inside try block");
            J = i/10;
            SOP("existing try block");
        }
        Catch (ArithmeticException e)
        {
            // e.printstackTrace(); // this is automatic method
            SOP("inside catch block");
        }
    }
}
```

7th Nov 2016

In the below program JVM searches for the catch block which matches with the exception and executes the corresponding catch block

Package com.qspiders.pack1;

Public class Demo115

{

    PSVM()

    {

        Int i=10;

        Int j;

        Try

        {

            SOP("inside try block");

            J=i/0;

            // int k = Integer.parseInt("test");

        }

        Catch (NumberFormatException exp)

        {

            SOP("inside Number Format Exception catch block);

            // exp.printStackTrace();

        }

        Catch (ArithmeticException exp)

        {

            SOP("Arithmetic Exception catch block);

            // exp.printStackTrace();

        }

        SOP("I = " + i);

    }

}

Above program also shows that for each statement that can generate exception, we need separate try – catch block

// handling multiple statements that can generate exception use separate try-catch block

Handling multiple statements that can generate exception use separate try-catch block

```java
Package com.qspiders.pack1;

Public class Demo115
{
    PSVM()
    {
        Int i=10;
        Int j;
        Try
        {
            SOP("inside try block");
            J=i/0;
        }
        Catch (ArithmeticException exp)
        {
            SOP("Arithmetic Exception catch block);
            // exp.printStackTrace();
        }
        Try
        {
            SOP("inside Number Format Exception try block);
            int k = Integer.parseInt("Hundred");
        }
        Catch (NumberFormatException exp)
        {
            SOP("inside Number Format Exception catch block);
            // exp.printStackTrace();
        }
        SOP("I = " + i);
    }       }
```

****

Alternately multiple statements can be handled using nested try-catch. First statement should be inside the inner try long with corresponding catch block and the second statement should be inside outer try.

```
Package com.qspiders.pack1;

Public class Demo115
{
        PSVM()
        {
                Int i=10;
                Int j;
                Try
                {
                        SOP("inside try block");
                        Try
                        {
                                SOP("inside inner try block);
                                J=i/0;
                        }
                        Catch (ArithmeticException exp)
                        {
                                SOP("Arithmetic Exception catch block);
                                // exp.printStackTrace();
                        }
                        int k = Integer.parseInt("Hundred");
                }
                Catch (NumberFormatException exp)
                {
                        SOP("inside Number Format Exception catch block);
                        // exp.printStackTrace();
                }
                        SOP("I = " + i);           }           }
```

*****

'finally' block is used along with try catch mostly or can be used along with try alone (try – finally). This block gets executed irrespective of a statement generating an exception or not

We can have try-catch or try-catch-finally or try-finally

Following is the example

```
Package com.qspiders.pack1;

Public class Demo118
{
      PSVM()
      {
            Int I = 10;

            Int j;

            Try
            {
                  SOP("inside try block");

                  J=i/0;

                  SOP("exiting try block");
            }

            Catch (AithmeticException exp)
            {
                  SOP("inside catch block");
            }

            Finally // to mandatorily execute block of code
            {
                  SOP("inside finally block");
            }

            SOP("i= " + i);
      }
}
```

****

Package com.qspiders.pack1;

Public class Demo118

{

    PSVM()

    {

        SOP("main method starts");

        SOP(test());

        SOP("main method ends");

    }

    Static String test()

    {

        Int I = 10;

        Int j;

        Try

        {

            SOP("inside try block");

            J=i/0;

        }

        Catch (AithmeticException exp)

        {

            SOP("inside catch block");

        }

        Finally // to mandatorily execute block of code

        {

            SOP("inside finally block");

        }

        Return "from outside block";

    }

}

```java
//single try block can have multiple catch blocks
//if 2 exceptions class in catch block are subclass and super
//first more specific and then super class
public class MulitpleCatchDemo {

    public static void main(String[] args) {

    //try with "" or "hundred" or "12.56" or close
    //Scanner like sc.close()

    String str="12.56";

        Scanner sc=new Scanner(str);

    int i=0;

    try{
                i=sc.nextInt();
        }

        catch(InputMismatchException imexp)
            {
        System.out.println(imexp);
            }
            catch(NoSuchElementException nsexp)
            {
                    System.out.println(nsexp);
            }
```

```
            catch(IllegalStateException iaexp)

            {

                    System.out.println(iaexp);

            }



            System.out.println(i);

  }

}
```

Test 3 questions:

```
what are unary operators?

what are increment and decrement operators?

what is the difference b/w pre increment and post increment?

what are the o/p of following programs?

class IncrementDemo1{

 public static void main(String [] args){

    int a=20;
    a= ++a + ++a;

    System.out.println(a);

  }
}

class IncrementDemo2{

public static void main(String [] args){

    int a=20;
    a= a++ + a++;

    System.out.println(a);

}
}
class IncrementDemo3{

public static void main(String [] args){

    int a=20;
    a= a++ + ++a;

    System.out.println(a);

}
}

class IncrementDemo4{

public static void main(String [] args){

    int a, b;
```

```
    a=10;
    b=--a;
   System.out.println(b);


}
}
```

what are methods?What is method syntax?
Write a method which can return two int values?
what can methods return?
write a method to find the area of a circle?
write a method to find the area of a square?
write a method to find the volume of a cube?
write a method to find the volume of a cone?
write a method to find the volumen of a cylinder?
write a method to find the total amount to be paid
prinipal,rateof interest and time -all user input
hint-ptr/100
write a method to return Orange object?
 write a method to return marks of 4 subject

Test 4 questions:

Questions

1.The most basic control flow statement supported by the Java programming language
is the ___ statement.
2.The ___ statement allows for any number of possible execution paths.
3.The ___ statement is similar to the while statement, but evaluates its expression
at the ___ of the loop.
4.How do you write an infinite loop using the for statement?
5.How do you write an infinite loop using the while statement?

Program
    Consider the following code snippet.

    if (aNumber >= 0)
        if (aNumber == 0)
            System.out.println("first string");
    else System.out.println("second string");
    System.out.println("third string");

        What output do you think the code will produce if aNumber is 3?
        Write a test program containing the previous code snippet; make aNumber 3.
What is the output of the program? Is it what you predicted? Explain why the output
is what it is; in other words, what is the control flow for the code snippet?
        Using only spaces and line breaks, reformat the code snippet to make the
control flow easier to understand.
        Use braces, { and }, to further clarify the code.

1 The term "instance variable" is another name for ___.
2 The term "class variable" is another name for ___.
3 A local variable stores temporary state; it is declared inside a ___.
4 A variable declared within the opening and closing parenthesis of a method
signature is called a ____.
5 What are the eight primitive data types supported by the Java programming
language?
6 Character strings are represented by the class ___.
7 An ___ is a container object that holds a fixed number of values of a single
type.

Excercise

244

```
Create a small program that defines some fields. Try creating some illegal field
names
 and see what kind of error the compiler produces.
Use the naming rules and conventions as a guide.

In the program you created in Exercise 1, try leaving the fields uninitialized and
print out their values. Try the same with a local variable and see what kind of
compiler errors you can produce. Becoming familiar with common compiler errors will
make it easier to recognize bugs in your code.
```

8th Nov 2016

// when you want to catch an exception and throw another exception, you can chain the first exception to the throwing exception or can return same exception

Package com.qspiders.pack1;

Public class ChainedExceptionDemo

{

      Public static void print()

      {

            Try

            {

                  String str[] = {"Hello"};

                  SOP(str);

            }

            Catch (ArrayIndexOutofBoundException aiob)

            {

                  Throw new RuntimeException(aiob);

            }

      }

      PSVM()

      {

            Try

            {

                  Print();

            }

            Catch(RuntimeException re)

            {

245

```
                Sop(re.getclass());

                Sop(re.getcause());

        }

    }

}
```

                                    *****

Package com.qspiders.smaple1;

Import java.util.Scanner;

Public class TryWithResourceDemo

```
{

    PSVM()

    {

        Try(Scanner sc = new Scanner(System.in));

        {

            Int I = sc.nextint();

        }

        Catch(exception e)

        {

        }

    }

}
```

// we have resource inside the parenthesis of try block. Compiler will internally converts try with resources to try finally block.

If catch is present it will be returned

Once work is done resources are automatically closed. Don't close it explicitly

Package com.qspiders.pack1;

Class Demo120

```
{

    PSVM()

    {

        SOP("main starts");
```

```
                SOP(test());

                SOP("main ends");

        }

        Static String test()

        {

                Int i=10;

                Int j;

                Try

                {

                        J=i/2; // put j=i/0 and try the program

                        Return "pass";

                }

                Catch(Arithmetic Exception exp)

                {

                        Exp.printStackTrace();

                        Return "fail";

                }

        }

}
```

o/p – when j=i/2   **o/p – when j=i/2**

main starts   main starts

pass   exception

main ends   fail

                                        main ends

****

```
Public class Demo121

{

        PSVM()

        {

                SOP("main starts");

                SOP(test());
```

```
                SOP("main ends");

        }

        Static String test()

        {

                Int i=10;

                Int j;

                Try

                {

                        J=i/2;

                        Return "pass";

                }

                Catch(Arithmetic Exception exp)

                {

                        Exp.printStackTrace();

                        Return "fail";

                }

                Finally

                {

                        Return "passfail";

                }

                /*Finally

                {

                        SOP("Hello Hello Dirty Fellow");

                } */

        }

}
```

When we have return statement in finally block the o/p is –

main starts

passfail

main ends

When we don't have return statement and have print statement in finally block (commented one in above program) the o/p is –

main starts

Hello Hello Dirty Fellow

pass

main ends

you can have return statement in try and catch or in finally or only outside or in try-catch –finally (in all) or in catch-finally or catch and outside. If you have return in finally and outside finally block or in try-catch and outside catch block then it became unreachable code and we shouldn't write the code like this

Return statement in try-catch block

Sr no	Inside try block	Inside catch block	Inside finally block	Outside finally block	Result
1	yes	No	No	No	CTE(error)
2	No	Yes	No	No	CTE(error)
3	yes	Yes	No	No	Success
4	No	No	Yes	No	Success
5	No	No	No	Yes	Success
6	yes	Yes	No	Yes	CTE(error)
7	No	No	Yes	Yes	CTE(error)
8	yes	No	Yes	No	Success
9	yes	Yes	Yes	No	Success
10	No	Yes	Yes	Yes	CTE(error)

```
Exception Handling:
   • An exception is an event triggered when jvm is not able to execute a
     statement
Handling the event is known as exception handling
   • Whenever exception occurs jvm terminates the execution by throwing the
     exception message
   • In order to complete the execution the exception should be handled by using
     try..catch block
   • Always the try ..catch block should be written in sequence.
   • try  block alone is not allowed
   • catch should take an argument of type exception
   • try block should contain the statements which generates exception. Once an
     exception has occurred jvm will go to catch block , remaining portion of the
     try block will never be executed
   • In the catch body we can print the object or we can print the entire
     exception trace. After executing the catch body the execution continues from
     the remaining program
   • In between try-catch block no other executable statements are allowed
```

9th Nov 2016

```
Public class Demo122
{
        PSVM()
        {
                Demo122 r = new Demo122();
                r.test1();
                void test1()
                {
                        Test2();
                }
                void test2()
                {
                        Test3();
                }
                void test3()
                {
                        Test4();
                }
                void test4()
                {
                        Int i=10/0;
                }
        }
}
```

o/p – exception in thread "main" java.lang.ArtihmeticException:// by zero

----

Interview question: when does stack unwinding happen?

While executing any program the methods of the classes will be loaded into stack for the execution purpose. Stack will be loaded with the methods in the order the way it is called. If the last entered method generates any exception then JVM looks for the handler in the same method. If no handler is found in the method then the exception propagates to the caller method. In the called method if exception is not handled, the exception will propagates to its called method. Finally the exception reaches main method.

If exception is not handled in main method, the JVM forcibly removes all the methods from the stack and terminates the program execution. This is known as stack unwinding

How exception is generated?

Class A

{

    Int i=10/0;// exception occurs at this line

}

 Collection of exception classes (JVM searches for the related class)

1

2

3   → new ArithmeticException();

Arithmetic Exception

|

|

N exception class

Generation of Exception: during execution of any program if java(JVM) finds any abnormal statement or any statement which generates exception, then it collects the information and looks for the type (class) matching the exception. Once the match is found JVM creates an object of the class with the details and throws the object. In the catch block the argument type should be same type of object raised or its super class type.

Note: when an exception is thrown it can be caught by reference variable of the same class or its super class or its super class... in the catch block
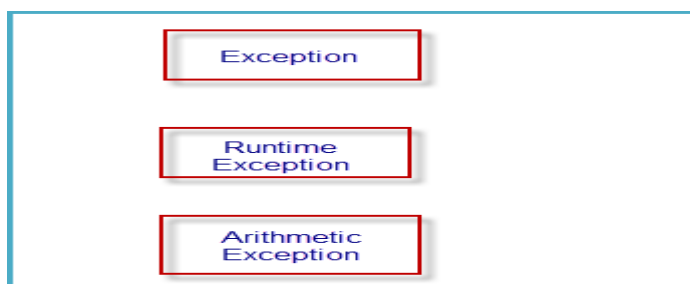
10th Nov 2016

Types of exceptions:

1. Checked exception
2. Unchecked exception

Note: throws keyword should be used only with checked exceptions

Package com.qspiders.pack1;

Public class Demo123

{

       PSVM() throws ClassNotFOUNDException

       {

              Class c1 = class.forName("com.qspiders.pack1.Demo123");

       }

}

Checked exception: an exception which the compiler can check at the time of compiler is known as checked exception. Checked exception can be handled using try-catch block or using <u>throws</u> keyword along with the method through exception Name. it should be written in the method name after the closing parenthesis

Ex: returntype method name(paramlist) throws exception
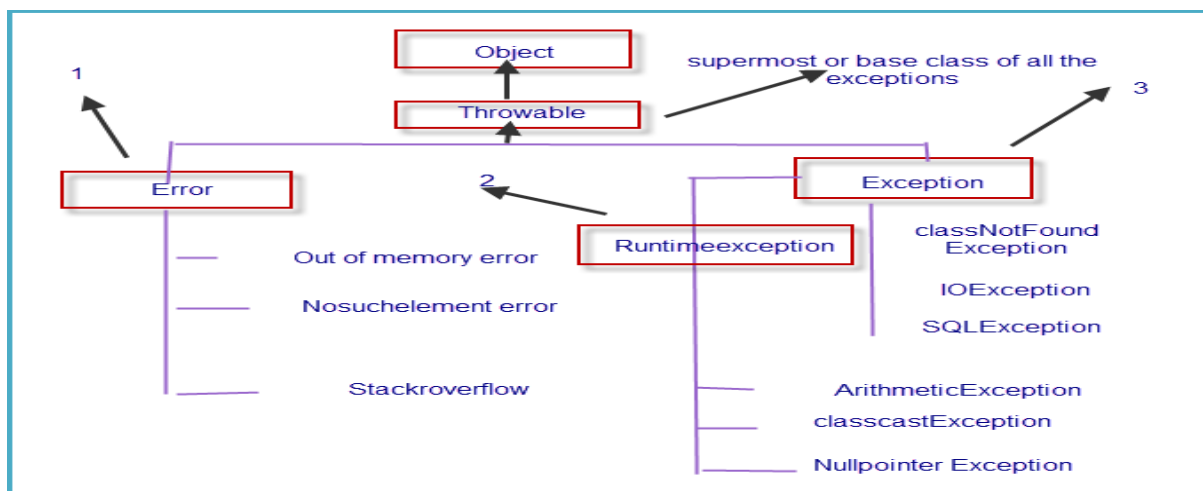
1 and 3 are unchecked exceptions

2 is checked exception

Ex: 1. If we get IOException we can catch argument of same class or its super class exception

2. for NullPointerException use same method or RunTimeException or Exception or Throwable

Interview questions:

1. Difference between throw, throws and throwable
2. Difference between final, finally and finalize
3. What is an error?
   Error is a subclass of Throwable class that indicates abnormal conditions which can cause serious problems. It should not be handled.
   Ex: stackoverflow error, when we use method recursion

Imp: Throwable is the base class of all exception classes. It is super most class of all exception classes

Creating your own exception:

Package com.qspiders.exceptionDemo;

Import java.util.scanner;

Class InvalidAgeException extends RuntimeException

{

      InvalidAgeException(String str)

      {

            SOP(str);

      }

}

Public class Demo124

{

      PSVM()

      {

            Scanner sc = new Scanner(System.in);

            SOP("Enter your age");

            Int age = sc.nectInt();

            Try

            {

                  If (Age>=60 || age<=0)

                  {

                        // SOP("Invalid Age: please enter age b/n 1-60")

                        Throw new InvalidAgeException("Invalid Age: please enter age b/n 1-60");

```
                    }

            }

            Catch (InvalidAgeException iae)

            {

                    SOP("Age Entered is" + age);

            }

        }

}
```

If we write our own exception class put it under any class RunTimeException or Exception. Then it becomes checked exception or unchecked exception based on the superclass the user class extends

Exception in Java are classified into 2 types

1.Checked Exception

2.Unchecked Exception

Checked Exception:

An Exception where the compiler can check at the time of compilation is known as checked Exception

🔲   Checked Exception should be handled in order to compile the program successfully.

🔲   Checked Exception can be handled in 2 ways

1)Surrounding try-catch block

2)By using throws declaration statement

 The throws declaration should be done in the method signature, the throws keyword should be used only for checked exceptions.

Unchecked Exception: Exceptions which are not been able to be identified by the compiler at the time of compilation is known as unchecked exceptions.

throws keyword cannot be used for unchecked exception and it should be handled only through try,catch block.

▢   throw keyword is used to generate or throw an exception(existing exception class or user defined exception) in the program.

▢   We can develop our own exception class by inheriting one of exception classes.

Garbage Collection

The garbage collector is under the control of JVM. The JVM decides when to run the garbage collector. From the Java program we can tell JVM to run garbage collector. We can write code to make objects eligible for garbage collection.

Setting reference variable to null

```java
public class Demo125 {


    public static void main(String[] args) {


    StringBuilder sb=new StringBuilder(▢Java▢);

    System.out.println(sb);

    sb=null; //eligible for collection


     String s1=new String(▢Hello▢);

        String s2=new String(▢Java▢);

      System.out.println(s1);


    s1=s2;

    //Hello object is eligible for collection


    System.out.println(new String("Banglore").length());

    //new object is eligible for GC


      }
}


class GarbageCollection
{
```

```java
  public static void main(String s[]) throws Exception
  {
    Runtime rs =  Runtime.getRuntime();

    System.out.println("Free memory in JVM before Garbage
    Collection = "+rs.freeMemory());
    rs.gc();
    System.out.println("Free memory in JVM after Garbage
    Collection = "+rs.freeMemory());

  }
}
```

finalize() method

This is a protected method of Object class thats run before the object is deleted by the garbage collector.

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

        }

        protected void finalize() throws Throwable
        {
                super.finalize();
                System.out.println("our cleaning");
        }
}
```