

# Syllabus

Selenium Webdriver:

ECLIPSE:

Java Topics

1 \* Basic Programming of Java: [08-01-2013 Tuesday]

1.1 \* How to install jdk Pg 2

1.2 \* Compilation & execution Pg 4

1.3 \* Write a Sample Program Pg 5

1.4 \* Variable, methods () Pg 7

1.5 \* Conditional Statements Pg 8

1.6 \* Loops Statements Pg 9

1.7 \* Methods () Pg 10

2 \* Object Oriented Programming:-

2.1 \* Class Members [21-01-2013 MONDAY] Pg 17

2.2 \* Object [22-01-2013 TUESDAY] Pg 20

2.3 \* Inheritance [30-01-2013 WEDNESDAY] Pg 34

2.4 \* Abstract [04-02-2013 MONDAY] Pg 40

2.5 \* Interface [05-02-2013 Tuesday] Pg 43

2.6 \* Typecast [06-02-2013 WEDNESDAY] Pg 46

2.7 \* Polymorphism [09-02-2013 / Saturday] Pg 52

2.8 \* Packages [11-02-2013 / MONDAY] Pg 54

2.9 \* Encapsulation

30 - 32  
classes

3 \* Object Class Pg 57

4 \* String Class Pg 61

5 \* Array Class Pg 64

6 \* Collections [Data Structures] Pg

7 \* Exception Handling Pg

8 \* File Handling Pg

9 \* Threads Pg

## API Classes

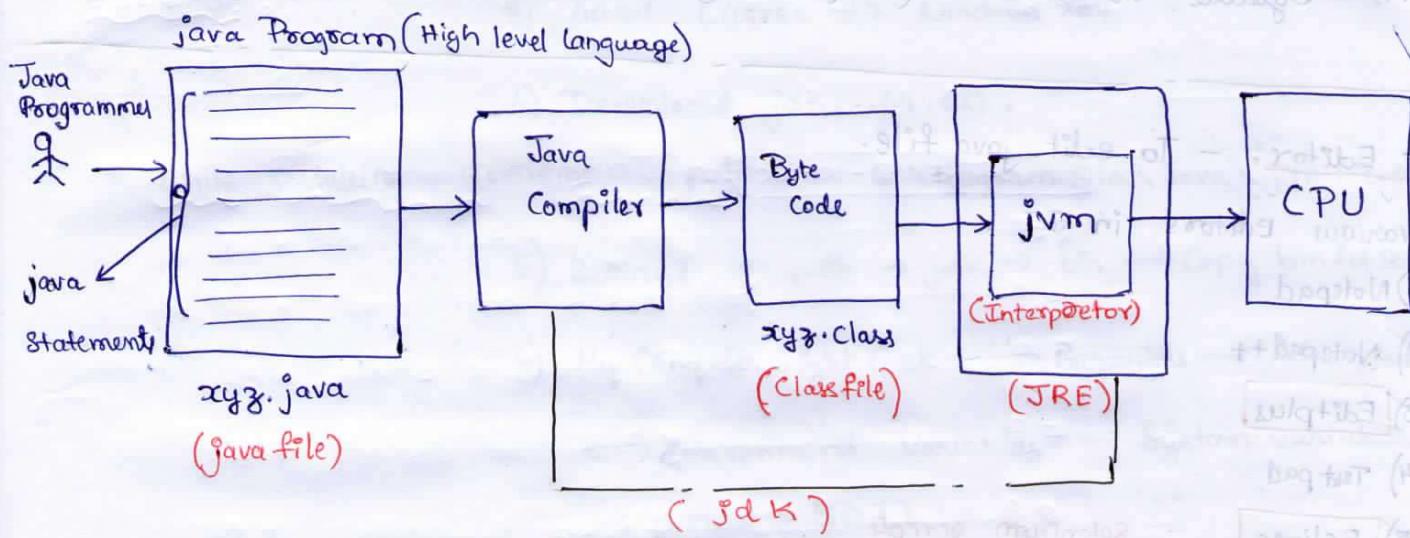
- 1\* JUnit / TestNG
- 2\* Ant Tool
- 3\* Log4j
- 4\* Html/CSS

5 - 6 classes

## SELENIUM

- 1\* Introduction to Selenium IDE & Selenium RC
- 2\* Intro Selenium WebDriver
- 3\* Frameworks

20-25 classes

CORE JAVACHAPTER 1 : BASICSNOTES :

- 1\*) All java statements are saved as javafile with extension ".java".
- 2\*) Java Compiler generates Classfile based on javafile.
- 3\*) Class file contains byte code, which is understood only by jvm [ Java Virtual Machine ].
- 4\*) jvm is a interpreter, it converts byte code line by line into Machine Level Language (M.L.L) during execution.
- 5\*) Java compiler and JRE (Java Runtime Environment) together is called jdk.
- 6\*) Java is platform independant i.e only needs classfile, only condition it needs jvm.

## Interview Questions

1\*) What is executable format of Java?

A: Bytecode i.e. classfile (only need JVM, i.e. JRE).

\* Editor: - To edit java file.

various editors in IT :

1) Notepad

2) Notepad++

3) Editplus

4) Text pad

5) **Eclipse** - Selenium script

6) NetBeans

2\*) How To Install:

Step 1 : To Install      1) Editor - editplus

2) Java Compiler ] - Jdk [ Java development Kit ]  
3) JRE

Step 2: Version : jdk - 1.5

jdk - 1.6

**jdk - 1.7** - latest

Step 3: Types of Releases:

Selenium - 1) J2SE - Java 2 Standard Edition [ standalone, Client server applications ]

2) J2EE - Java 2 Enterprise Edition [ Web Applications ]

3) J2ME - Java 2 Micro Edition [ Mobile Applications ]

- Step 4: Install :
- 1) Google → jdk 1.7 download → Java SE Downloads
  - 2) JavaSE → Latest Release → Java Platform
  - 3) Accept License → Windows X64
  - 4) Download jdk-...-64.exe.

While installing remember path i.e C:\Program Files\Java\jdk 1.7.0-01

- 5) Settings → path → jdk → bin → Copy bin folder path
- 6) My Computer right click → Properties → Advanced  
→ Environment Variables → System variables  
→ path → EDIT  
  
7) Variable Value → scroll to end → paste copied bin folder  
then path  
OK ←
- 6) Command prompt : java -version

If error: path error 'java' is not recognised as an internal or external  
command (put; and then paste correct path).

- 1\*) **Program Statements** contains
- 1.1) keywords
  - 1.2) Identifier
  - 1.3) Literals

2\*) **Keyword**: \* pre-defined reserved word.

\* All keywords in java are in lowercase.

Ex: int, if, switch, void, class, final

3\*) **Identifiers** : \* are the names given/provided in program by programmer.

\* Rules to be followed to give names:

\* Keyword should not be used.

\* Use alphanumeric characters.

\* Always starts with Alphabet / starting character should be alphabet only.

Ex: i) int age;      ? all variable names, method names,  
ii) double salary1;    ? program names are identifiers

4\*) **Literals** : \* Values provided in program

Generally 3 types:

\* numeric literals

\* character literals

\* String literals.

5\*) Developing a java program involves 3 steps

## 5.1) Coding

## 5.2) Compilation

### 5.3) Execution

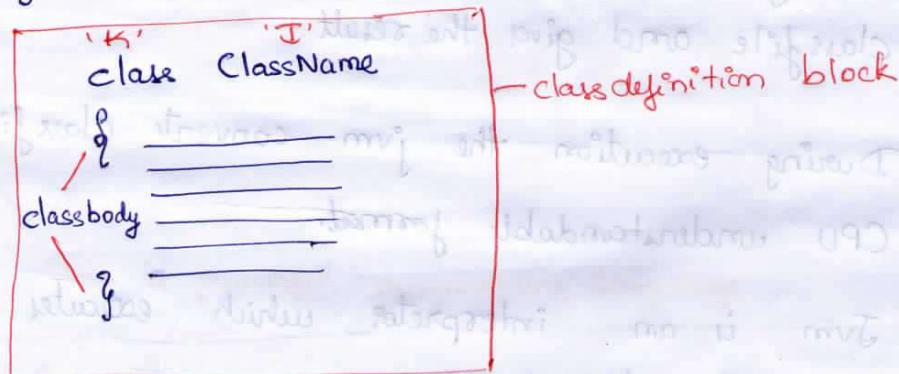
6\*) Coding : Writing a java program is known as Coding.

Here we write programmes using java statement.

The program written should be saved with extension '.java'. This file is known as javafile.

A javafile should have class definition block.

The syntax to write class definition block is



It's by convention the classname should begin Uppercase. The java statements has to be written inside the class body.

7\*) While saving the java program, the filename should be same as class name.

## 8\*) Compilation :

8.1\*) Is process of converting java file into class file.

8.2\*) The class file contains bytecode.

8.3\*) Java Compiler translates java statements into bytecodes at the time of compilation.

8.4\*) Java Compiler generates the classfile and saves the classfile in the location where javafie is available.

8.5\*) The Syntax to compile any javafie is

`javac filename.java`

## 9\*) Execution :

9.1\*) Running the classfile is known as execution.

9.2\*) The jvm executes the bytecodes present in the classfile and gives the result.

9.3\*) During execution the jvm converts classfile into CPU understandable format.

9.4\*) Jvm is an interpreter which executes the statements line by line

9.5\*) Syntax to execute classfile is

`java classname`

## VOTE

- 10\*) JVM starts execution only if java class contain main method otherwise JVM throw error.
- 11\*) A java class which doesn't have a main method cannot be executed.
- 12\*) To compile a javafile change the working directory to location where javafile is saved.

Example:

1) P1:

```
class SampleProgram
{
    public static void main(String[] args)
    {
        System.out.println("Hi, Welcome to Java"); // displays the message on the screen, print new line char
        System.out.println("Java is powerful programming Language");
    }
}
```

O/p

Hi , Welcome to Java  
Java is powerful programming Langauge

2) P2:

```
class SampleProgram
{
    public (String[] args)
    {
        System.out.print ("Hi, Welcome to java \n"); // \n is to print in newline
        System.out.print ("Java is powerful programming language ");
    }
}
```

3) class SampleProgram2

```
{
    psvm (String [ ] args)
    {
        System.out.println("Hi, Welcome again " + "and again to Java");
        // '+' To join two String data with
        // separate " "
    }
}
```

O/p:-

Hi, Welcome again and again to Java

P4:

4) class Demo1

```
{
    psvm( )
    {
        Sop (10+10); // here + adds 10+10 first and then
        // displays the result
    }
}
```

O/p:- 20

P5:

5) class Demo2

```
{
    psvm (String [ ] args)
    {
        Sop ("Program starts....");
        int id; // declaration
        id = 2012345; // initialization
        Sop (id); // points id value
        Sop ("Program ends....");
    }
}
```

O/p:

Program starts...

2012345

Program ends...

NOTE: Before using a variable, we need to initialize, before initializing we need to declare the variable.

P68) class AssignProgram

```
public static void main(String[] args){  
    System.out.println("Program starts...");  
    int num1;  
    int num2;  
    int sum;  
    num1 = 10;  
    num2 = 20;  
    sum = num1 + num2;  
    System.out.println(sum);  
    System.out.println("Program ends...");  
}
```

O/p: Program starts...  
30  
Program ends...

P70) class Demo4

```
public static void main(String[] args){  
    System.out.println("Program starts...");  
    final int empID; // final  
    empID = 12973;  
    System.out.println("empID is " + empID);  
    System.out.println("Program ends...");  
}
```

O/p: Program starts...  
empID is 12973

Program ends...

P8) class Demo3

```
public static void main(String[] args){  
    System.out.println("Program starts...");  
    double salary = 15000.98;  
    System.out.println("Salary is " + salary);  
    salary = 25128.54;  
    System.out.println("Salary is " + salary);  
    System.out.println("Program ends...");  
}
```

O/p: Program starts  
Salary is 15000.98  
Salary is 25128.54  
Salary is 35978.54  
Program ends

14\*) If we need to initialize a variable only once, then such variable has to be declared as final by using final keyword

Variables

- 15\*) In java variables are used to store the data of programming.
- 16\*) The variable has to be declared first before initializing.
- 16.1\*) The syntax to declare a variable

`datatype VariableName;`

17\*) Datatype: indicates the type of value that we assign to the variable. Java supports 8 type of datatypes.

1. byte

2. short

3. int

4. long

5. float

6. double

7. char

8. boolean

**NOTE** 18\*) In java there is no string variable. In order to store string value java provides a class `String`, `StringBuilder` & `StringBuffer`.

19\*) Every variable has to be initialized before using in any operation. The syntax to initialize variable is

`VariableName = value;`

20\*) If any variable is used without initialization, then compiler throws an error.

21\*) A variable can have diff values initialized, throughout the program execution.

### P9) class AreaOfCircle

```
{ psvm (String[] args)
```

```
{ Sop ("Program starts...");
```

```
final float PI;
```

```
PI = 3.14f;
```

```
double r;
```

```
r = 2.5;
```

```
float Area;
```

```
Area = PI * r * r;
```

```
Sop ("Area of Circle " + Area);
```

```
Sop ("Program ends...");
```

```
}
```

O/p: Program starts...

Area of Circle -

Program ends...

// float variablename = value f;  
long variablename = value l;

```
final double PI = 3.14f;
```

```
double radius = 10.0;
```

```
double Area;
```

```
Area = PI * radius * radius;
```

```
Sop ("Area of circle with radius " + radius + " is"  
+ Area);
```

```
Sop ("Program ends...");
```

```
}
```

P10

### 10) class AssignSimpleInterest

```
{ psvm (String[] args)
```

```
{ Sop ("Program starts...");
```

```
double Interest;
```

```
long Principle = 10000L;
```

```
double Rate = 0.15;
```

```
int Time = 2;
```

```
Interest = Principle * Rate * Time;
```

```
Sop ("The Simple Interest for " + Principle + " amount at the rate of "  
"Rate for " + Time + " years duration is " + Interest);
```

```
Sop ("Program ends...");
```

```
}
```

O/p:

Program starts...

The Simple Interest for 10000 amount at the rate of 15% for 2 yrs ⑦

duration is 3000

Program ends...

1) class AssignDegToFahrenheit

```
{    psvm (String [ ] args)
{
    Sop ("Program starts...");  

    double Fahrenheit;  

    double DegreeCelsius = 28;  

    Fahrenheit = (DegreeCelsius * 1.8) + 32;  

    Sop ( "Degree Celsius is equal to " + Fahrenheit + " Fahrenheit");  

    Sop ("Program ends...");  

}
```

O/P:

Program starts...

28 Degree Celsius is equal to 82.4 Fahrenheit

Program ends...

\*\*\*\*\*

15-01-2013 TUESDAY

Condition Statements

P1Q

## 1. Class Demo 3

## ) If - Else Statements

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

```
{ System.out.println("Program Starts...");
```

```
final int empID = 1021;
```

```
if (empID == 1021);
```

```
{ System.out.println("This empID belongs to Project Manager");
```

```
}
```

```
else {
```

```
System.out.println("This empID doesn't belong to Project Manager");
```

```
}
```

```
System.out.println("Program ends...");
```

```
}
```

P13

## 2. Class Demo 4

```
public (String[] args)
```

```
{ System.out.println("Program Starts...");
```

```
final int empID = 1021;
```

```
if (empID == 1021)
```

```
{ System.out.println("This empID belongs to Project Manager");
```

```
else if (empID == 1022)
```

```
{ System.out.println("This empID belongs to Team Lead");
```

```
else
```

```
{ System.out.println("This empID doesn't belong to PM or TL");
```

```
System.out.println("Program ends...");
```

```
}
```

(8)

## Assign

1. Declare 3 variable integer type and display highest of the three.  
 (And) && (Or) ||

P14

3. class Demo4

```
{ psvm (String[] args)
```

```
{ S.o.p ("Program starts...");
```

```
int a = 223;
```

```
int b = 34;
```

```
int c = 183;
```

```
if ( a > b && a > c ) // T&&T
```

```
{ S.o.p (a + " is highest");
```

```
}
```

```
else if ( b > a && b > c ) // T&&T
```

```
{ S.o.p (b + " is highest");
```

```
}
```

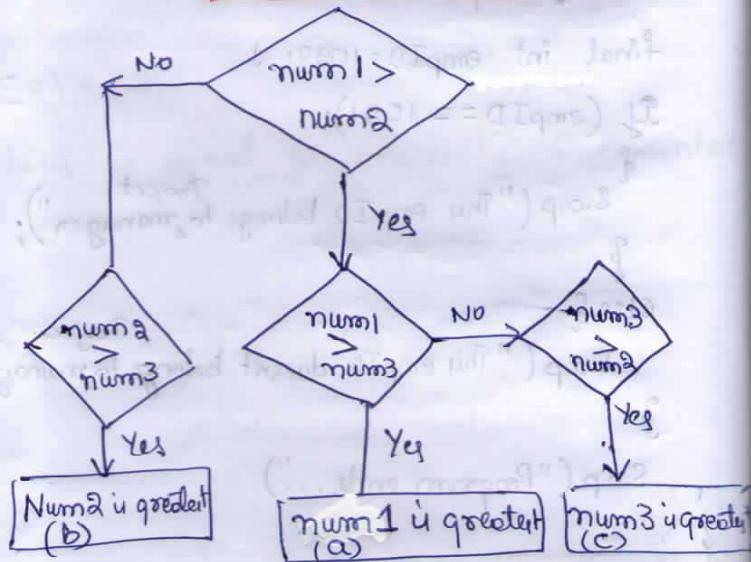
```
else
```

```
{ S.o.p (c + " is highest");
```

```
}
```

```
} S.o.p ("Program ends...");
```

T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F



(or)

```
if ( a > b )
```

```
{ (if ( a > c )
```

```
{ (a is highest);
```

```
? else if ( c > b )
```

```
{ (c is highest);
```

```
else
```

```
{ (b is highest);
```

class Demo5

psvm (String[] args)

```
{
    System.out.println("Program starts...");
```

```
int empID = 10213; // args
```

```
// loops
```

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

```
{
```

```
    System.out.println("value of empID is " + empID);
```

```
}
```

```
System.out.println("Program ends...");
```

```
}
```

Loops

for loop :- [execute intended number of times]

for (initialization; condition; inc/dec)

    {  
        // code  
    }

}

loop body

  ↳ initial

  ↳ condition

  ↳ increment

↓

loop body

  ↳ initial

  ↳ condition

  ↳ increment

↓

  ↳ increment

↓

  ↳ increment

  ↳ increment

↓

  ↳ increment

↓

  ↳ increment

↓

[16-01-2013] WEDNESDAY

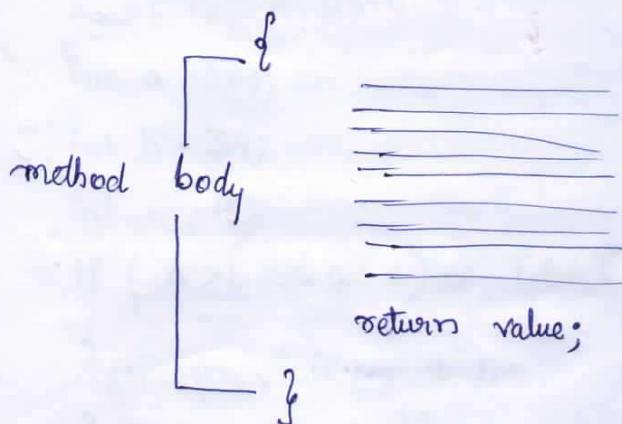
\* Modification of [underline number 6]

## METHODS

SYNTAX:

modifiers returnType methodName (args)

data passed



// Mandatory statement

Example:

P16  
class Demo1

```
{  
    public static void main (String[] args)  
    {  
        System.out.println ("Program Starts...");  
        sample ();  
        System.out.println ("Program ends...");  
    }  
}
```

// invoking method (or) Calling method

```
static int sample()  
{  
    System.out.println ("running Sample method");  
    return 10;  
}
```

O/P -  
Program Starts...  
running Sample method  
Program ends...

```
6. int ret = sample();  
7. System.out.println ("value of ret is "+ret);
```

Program starts...  
running Sample method  
value of ret is 10  
Program ends...

O/P =

### Example 2 :

P17

```

class Demo2
{
    psvm (String[] args)
    {
        Sop ("Programm starts...");  

        Sample();  

        Sop ("Programm ende...");  

    }

    static void sample()
    {
        Sop, out, println ("running sample method");
        // compiler writes return statement here
    }
}

```

O/P: Program starts...  
running sample method  
Program ends...

I.Q

Is return statement mandatory in a method in java?

Ans:- Yes, but in case of a void method [compiler writes return statement] No.

### Example 3 :

P18

```

class Demo3
{
    psvm (String[] args)
    {
        Sop ("Programm starts...");  

        int res = Sample (12,34);  

        Sop ("result is "+res);  

        Sop ("*****");  

        int res = Sample (23,64);  

        Sop ("result is "+res);  

        Sop ("Programm ende...");  

    }
}

```

[ P.T.O ]

Static int sample (int a, int b)

```
{  
    Sop ("running sample method");  
    Sop ("value of a is " + a);  
    Sop ("Value of b is " + b);  
    int sum = a+b;  
    return sum;  
}
```

}

OP:

Program starts...

running sample method

Value of a is 12;

Value of b is 34

Value of res is 46

\*\*\*\*\*

running sample method

Value of a is 23

Value of b is 64

Value of res is 87

Program ends...

### NOTES

24\*) While developing programs the repetitive tasks/operations core concept  
develop as method.

25\*) method is a block when invoked executes all the statements of method body.

26\*) The syntax to develop a method in java is

modifiers return type methodName (arg0, arg1, arg2)

{

=====

return value;

}

- 25\*) In every method return statement is mandatory
- 26\*) A return statement indicates the type of value the method has to return.  
return value should match return type.
- 27\*) Args/~~parameters~~ are used to variable used to store data passed for method.
- 28\*) A void return type indicates that method returns nothing.
- 29\*) If a method return type is declared as void it is not mandatory to write return statement, in such case compiler writes return statement at the time of compilation.
- 

### Assign

1. S.I , Degree to F , Area of Circle using method.

P19

```

class AssignCombine
{
    public static void main(String[] args)
    {
        System.out.println("Program starts...");

        double res = SimpleInterest(10000, 0.15, 2);
        System.out.println("Simple Interest is "+res);
        System.out.println("*****");
        double res = degreeToFahrenheit(28);
        System.out.println("Fahrenheit is "+res);
        System.out.println("*****");

        double area = areaOfCircle(20);
        System.out.println("Area of Circle is "+area);
        System.out.println("*****");
        System.out.println("Program ends...");
```

```
static double simpleInterest (double a, double rate, int time)
```

{

```
System.out.println("Running simpleInterest method");
```

```
double interest = a * rate * time;
```

```
Sop ("The amount " + a + " at the rate of " + rate + " for " + time + " years.");
```

```
return interest;
```

}

```
static double degreeToFahrenheit (double degreeCelsius)
```

{

```
Sop ("Running Degree Celsius to Fahrenheit");
```

```
double res = (degreeCelsius * 1.8) + 32;
```

```
Sop (" " + degreeCelsius + " Degree Celsius is equal to ");
```

```
return res;
```

{

```
static double areaOfCircle (double radius)
```

{

```
Sop ("Running area of Circle method");
```

```
final double PI = 3.14;
```

```
double area = PI * radius * radius;
```

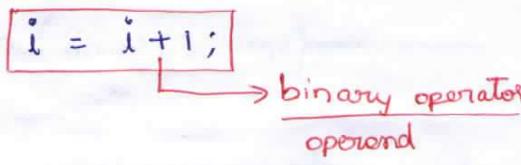
```
Sop ("For radius " + radius);
```

```
return area;
```

{

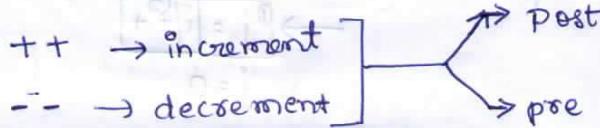
}

```
int i = 10;
```



### \* Unary operators:

- \* on only one operand



### Example 1:

P20

```
class Demo5
{
    public (String[] args)
    {
        System.out.println("Program starts...");

        int i = 1;
        System.out.println("i value bef inc " + i);
        i++;
        // pre increment

        System.out.println("i value aft inc " + i);
        System.out.println("Program ends");
    }
}
```

%P  
Program starts...  
i value bef inc is 1  
i value aft inc is 2  
Program ends...

\*  $i++;$   
System.out.println("i value after inc " + i);

// post increment

(12)

## Example 2:

P21

class Demo6

{

psvm (String[] args)

{

Sop ("Program starts...");

int i=0;

int j = 0;

j = i++;

Sop ("value of i " + i);

Sop ("value of j " + j);

Sop ("Program ends...");

{

}

O/P:

Program starts...

value of i is 1

Value of j is 0

Program ends...

Example 3:

P22

class Demo7

{

psvm (String[] args)

{

Sop ("Program starts...")

int i=0;

int j=0;

j = i++ + 1;

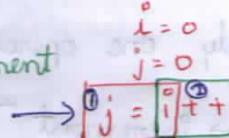
Sop ("value of i is " + i);

Sop ("value of j is " + j);

Sop ("Program ends...");

{

// post increment



$$\begin{aligned} & Q. \quad i++ = 0+1 = 1 \\ & \Rightarrow i=1 \end{aligned}$$

$$\begin{aligned} & j = \underline{i} \underline{++} + 1 \quad i = \underline{i} \underline{++} \\ & j = 0 + 1 \quad = 0 + 1 \\ & \boxed{j = 1} \quad \boxed{i = 1} \end{aligned}$$

O/P:  
Program starts...  
value of i is 1  
value of j is 1  
Program ends...

### Example 4:-

P23

```
class Demo8
```

```
{ psvm (String[] args)
```

```
{ Sop ("Program starts...");
```

```
int i = 0;
```

```
int j = 0;
```

```
j = i++ + i;
```

```
S.o.p ("Value of i " + i);
```

```
Sop ("Value of j " + j);
```

```
Sop ("Program ends...");
```

```
}
```

```
}
```

O/P:- Program starts...

value of i is 1

value of j is 1

Program ends...

### Example 5:- P24

\* 3

```
j = i++ + i++ + i
```

O/P:- Program starts...

value of i = 2

value of j = 3

Program ends...

$j = 0 + 1 + 2$

$j = 3$

### Example 6 :-

P25 class Demo9

```
{
    psvm (String[] args)
    {
        Sop ("Program starts...");

        int i = 0;
        int j = 0;
        j = ++i;
        // i++ = 1 => j = 1
        Sop ("Value of i is " + i);
        Sop ("Value of j is " + j);
        Sop ("Program ends...");
    }
}
```

O/p:- Program starts...

value of i is 1

value of j is 1

Program ends...

### Example 7 :- P26

```
8* j = i + ++i + i++ + ++i;
// = 0 + 1 + 1 + 3
```

$$j = 0 + 1 + 1 + 3 \quad i = 3$$

(++)

O/P:- Program starts...

value of i is 3

value of j is 5

Program ends...

$$i + ++i + ++i = i$$

$$i + 1 + 1 + 3 = 5$$

$$i + 1 + 1 + 3 = 5$$

$$i + 1 + 1 + 3 = 5$$

Example 8 :-

P27

```

class Demo10
{
    public void (String[] args)
    {
        System.out.println("Program starts . . .");
        int i = 12;
        test1(i++);
        System.out.println("i = " + i);
        System.out.println("Program ends . . .");
    }
}

static void test1(int a)
{
    System.out.println("a = " + a);
}

```

*i*  
1. test1(12)      2.  $i = 12 + 1$   
*i* = 13

%:- Program starts . . .  
a = 12  
i = 13  
Program ends . . .

Example 9 :- P28

7\* test1(++i);

%:- Program starts . . .

a = 13

i = 13

Program ends . . .

Example 10 :- P29

```

class Demo11
{
    public void (String[] args)
    {
        System.out.println("Program starts . . .");
        int i = 1;
        int j = test1(i);
        System.out.println("i = " + i);
        System.out.println("j = " + j);
    }
}

int test1(int i)
{
    return i + i;
}

```

```

    Sop("Program ends...");

}

static int test1(int a)
{
    System.out.println("a = " + a);
    return a + 1;
}

15*                                     // returns current value of a i.e 1
                                         and then increments a but it cannot
                                         be accessed as control goes to main
                                         method from test1 method

```

O/P :- Program starts...

a = 1  
i = 1  
j = 1

Program ends...

---

15\* return ++a; // return value after increment a i.e 2

O/P :- Program starts...

a = 1      i = 1  
i = 1  
j = 2

Program ends...

---

Example 11: P30

```

class Demo12
{
    psvm(String[] args)
    {
        Sop("Program starts...");

        int i = 0;
        int j = i++ + ++i + test1(i++) + i;
        // 0 + 2 + 3 + 3
    }
}
```

Sop("i = " + i);

Sop("j = " + j);

Sop("Program ends...");

}

```
static int test1(int a)
{
    System.out.println("a = " + a);
    return ++a;
}
```

### NOTES

- 30\*) Unary operator works on single operand. Java supports two types unary operators
- 30.1\*) Increment       $++$
- 30.2\*) Decrement       $--$
- 31\*) These operators are again classified as
- 31.1) pre
- 31.2) post
- 32\*) Post increment - means use current value for operation & increment variable value by 1.
- 33\*) Pre increment - means increment current value & use final value after increment.
- 34\*) Same for post decrement & pre decrement only cliff value gets reduced by 1.



## GLOBAL VARIABLE

P31

### Example 1

class Demo13

{

    static int i; // global variable & also for long, short, byte

    static double d;

    static char c;

    static boolean b;

    public (String [] args)

{

        System.out.println("Program starts...");

        System.out.println("i = " + i);

        System.out.println("d = " + d);

        System.out.println("c = " + c);

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

        System.out.println("Program ends...");

}

}

O/P:- Program starts...

i=0

d=0.0

c=

b=false

Program ends...

### Example 2 :

P32

class Demo13

{

    static int = 12; // static variable

    psvm (String[] args)

{

        Sop("Program starts ...");

        Sop("i = " + i); // referring global

        i = 24; // global static other variable go to memory first (1st)

        Sop("i = " + i); // global

        int i = 78; // local variable declaration

        Sop("i = " + i); // local

        i = 45; // local

        Sop("i = " + i); // local

        Sop("i = " + Demo13.i); // global

        Sop("Program ends...")

}

}

O/P : Program starts...

i = 12

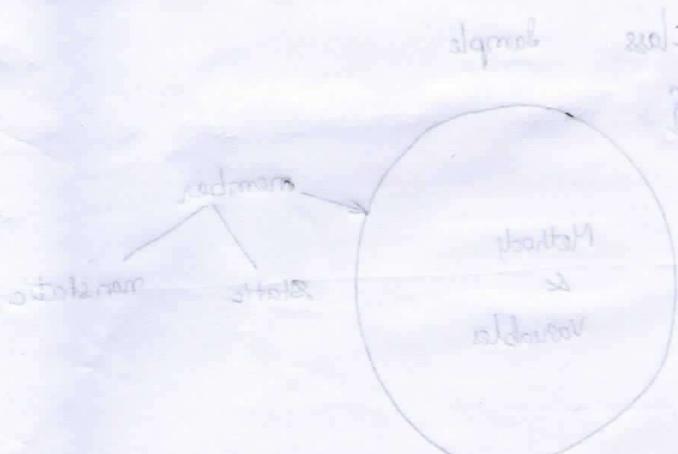
i = 24

i = 78

i = 45

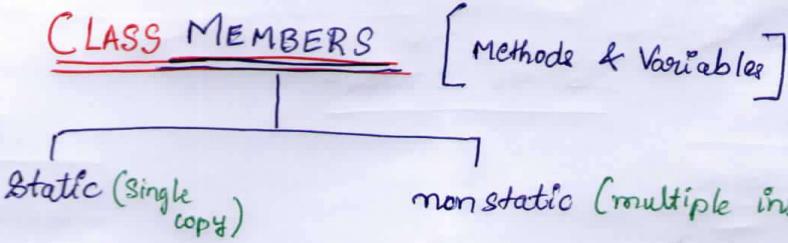
i = 24

Program ends...



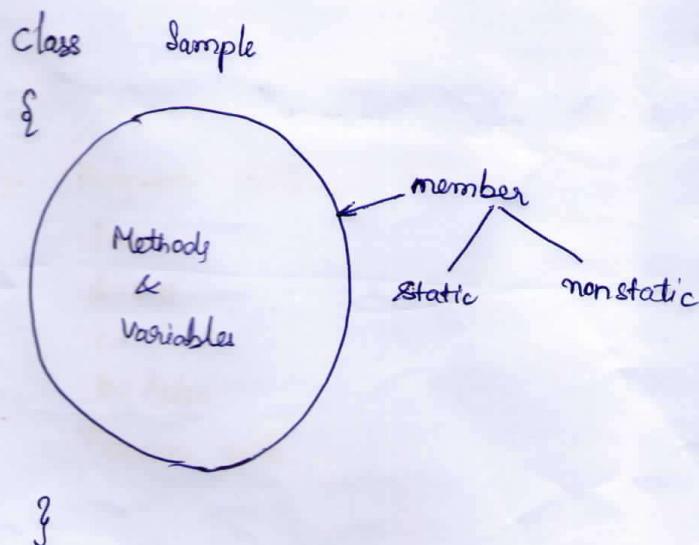
21-01-2013 / MONDAY

8 days



### Members of Class :

- 1\*) The members of class are categorized as static members and non static members.
- 2\*) Static members are declared with static keyword whereas non static members are declared without any keyword.
- 3\*) Static members of class can be accessed by using Class name.
- 4\*) Non static members are accessed by using Reference variable.



### Example 1:

P33

```

class memberA {
    static int i;
    static double d;

    public void display() {
        System.out.println("Program starts...");

        i = 10;
        d = 23.45;

        System.out.println("i = " + i);
        System.out.println("d = " + d);

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

    public static void main(String[] args) {
        memberA obj = new memberA();
        obj.display();
    }
}

```

O/P: Program starts...

i=0

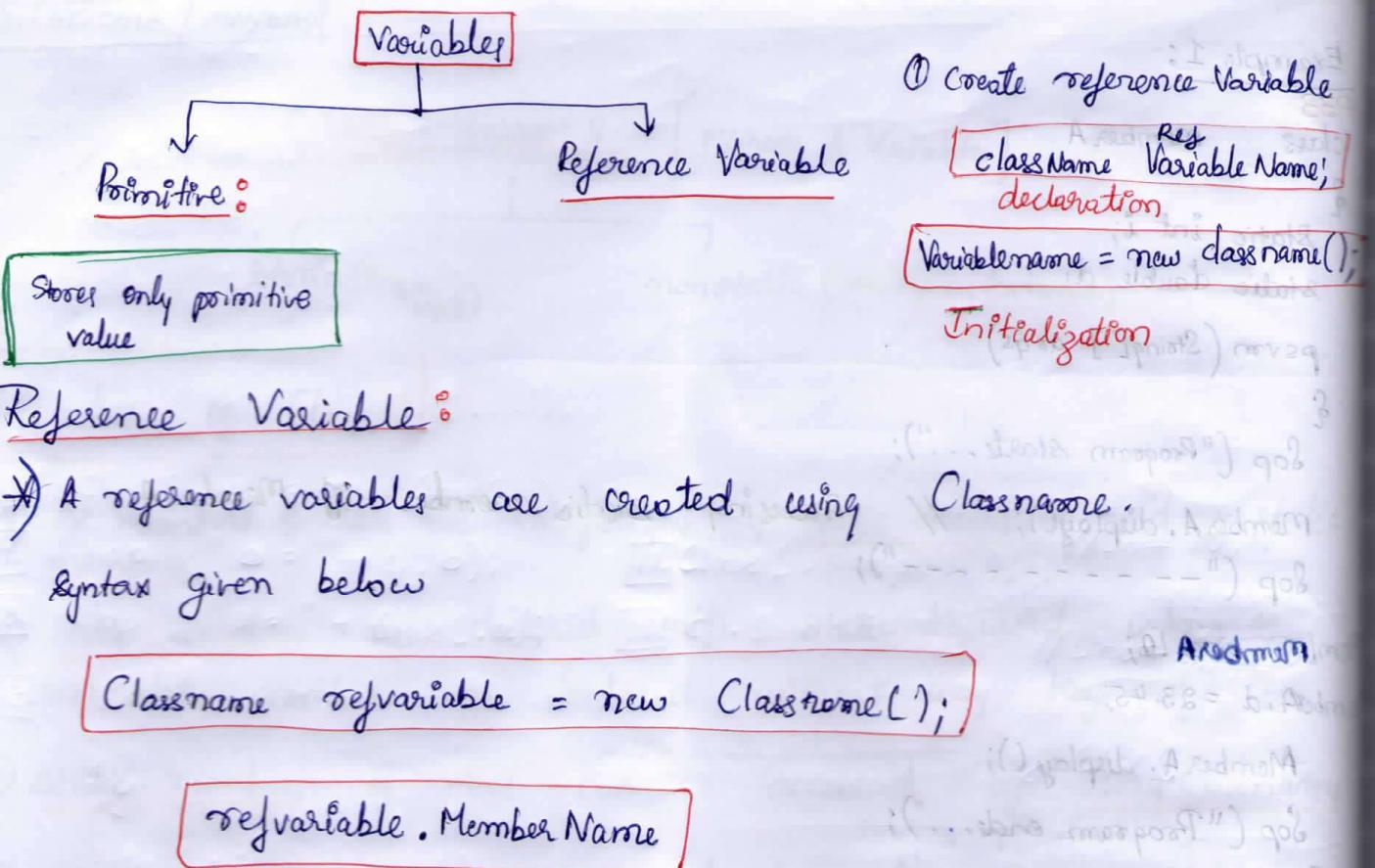
i=0.0

-----

i=10

i=23.45

Program ends...



## Reference Variable:

\* A reference variables are created using

Syntax given below

`Classname refvariable = new Classname();`

`refvariable.Member Name`

134 class ClassB

```
{  
    int k;  
    double s;  
    psvm (String[] args)  
}
```

Sop ("Program starts...");

ClassB b1 = new ClassB();

Sop ("k=" + b1.k);

Sop ("s=" + b1.s);

b1.test1();

Sop ("Program ends...");

}

void test1()

{

Sop ("running test1()...");

}

}

Op: Program starts...

k=0

s=0.0

running test1()...

Program ends...

P35

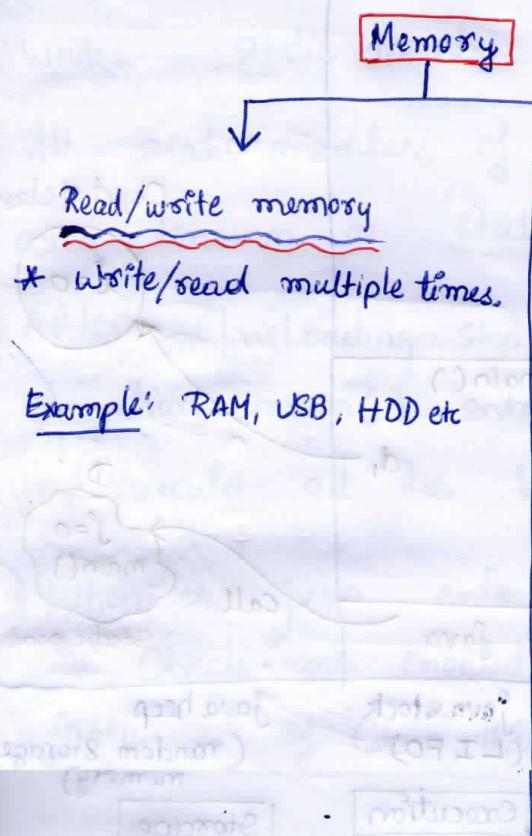
```
class ClassC  
{  
    static int a;  
    double b;  
    public (String[] args)  
    {  
        Sop ("Program start...");  
  
        Line# 9. class C c1 = new ClassC();  
        Sop ("b = " + c1.b); //Line# 10,11  
        Line# 10. Sop ("a = " + ClassC.a); //Line# 11,12  
        Line# 11. ClassC.sample1();  
        Line# 12. c1.sample2();  
        Sop ("Program end...");  
    }  
  
    static void sample1()  
    {  
        Sop ("running sample1()...");  
    }  
  
    void sample2()  
    {  
        Sop ("running sample2()...");  
    }  
}
```

%:-

22-01-2013

Tuesday

java = jum



### Read only memory

- \* write once.
- \* read many times.

Example: cd ROM

DVD ROM

### Read/write memory

RAM	HDD
* volatile.	1. non volatile.
* made of IC's	* Made of Magnetic type

## Example 1 : P36 MEMORY MANAGEMENT In JAVA

class D

{

int i;

static int j;

psvm (String[] args)

{

Sop ("Program starts...");

Sop ("j=" + D.j);

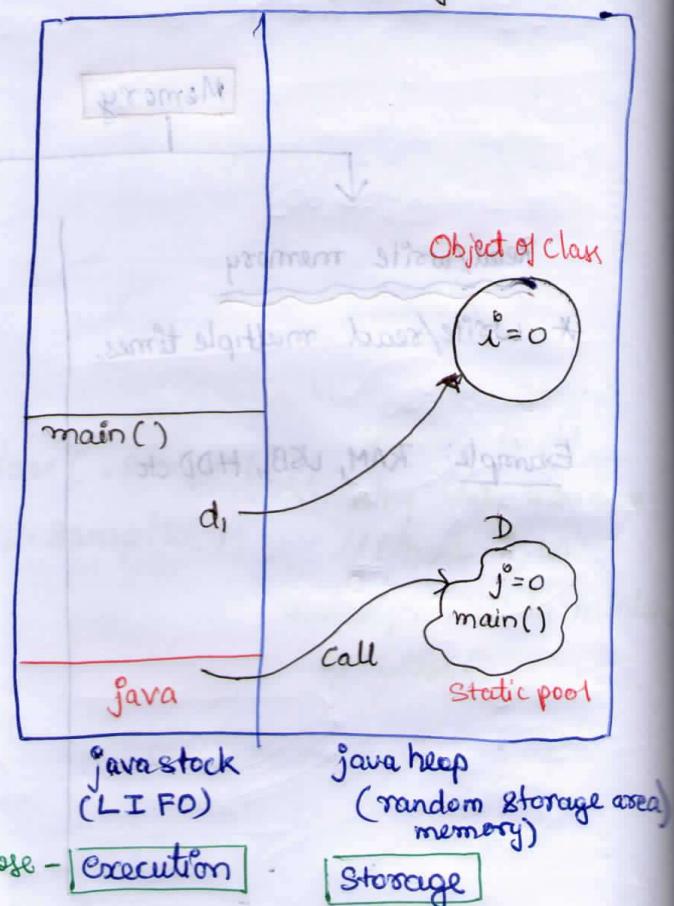
D d1 = new D();

Sop ("i=" + d1.i);

Sop ("Program ends...");

}

}



## Memory Management in Java

1) Whenever a java class is executed the memory gets allocated. This memory is divided into 2 areas

1. java stack

2. java heap

2\*) Java stack is a Last In First Out implementation and this stack is used for execution purpose.

3\*) Java heap are random storage area, is used for storing members of Class.

4\*) After memory allocation java enters into stack and calls ClassLoader.

- 5) Classloader is a java Class which loads the members of the Class which is needed to be executed.
- 6) While static members of Class are loaded into heap.
- 7) All static members of Class resides in the common area known as static pool.
- 8) After Classloading step, java calls main() method for execution. main() enters into stack on top of java and executes all the statements of main() line by line.
- 9) Whenever jvm encounters Object creation Statement, the Object are created in the heap. An Object is an instance of Class, this object is referred by reference variable.
- 10) Whenever an Object is created, the non static members of the class are loaded into Object memory.
- 11) The members available in static pool, should be accessed through Classname. This is also known as Class reference.
- 12) The members loaded in the Object [non static] can be accessed by using Reference Variable, this is also known as Object reference.
- 13) Once the main() completes all the statement, the main() gets out of the stack and control returns back to java [jvm].
- 14) The java calls Garbage Collector, which cleans the

heap memory. After this java gets out of stack and releases the memory back to Main Memory [RAM].

### Example 2 P37

class E

{

    int i=10;  
    static int j=12;

    void test1()

    {  
        System.out.println("running test1()...");

    static void test2()

    {  
        System.out.println("running test2()...");

}

    public static void main(String[] args)

{

    System.out.println("Program starts...");

    // accessing static members

    System.out.println("j=" + E.j);

    E.test2();

    System.out.println("\*\*\*\*\*");

    // non static members

    E e1 = new E();

    System.out.println("i=" + e1.i);

    e1.test1();

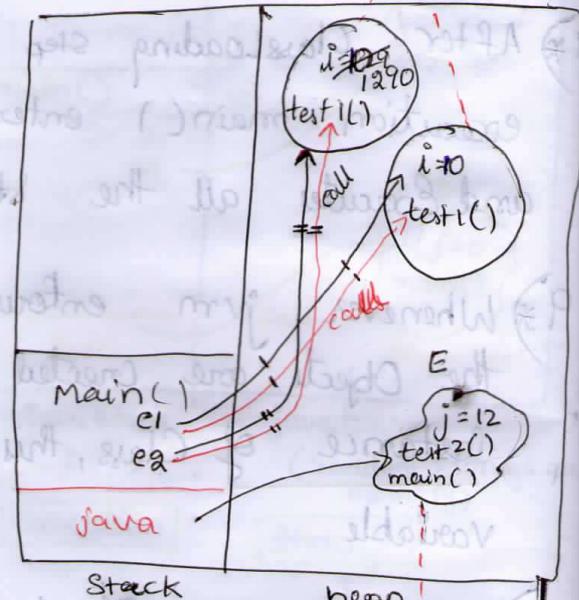
    System.out.println("\*\*\*\*\*");

    E e2 = new E();

    System.out.println("i=" + e2.i);

(multiple instances of class)

[Object/instance of Class]



(only one copy)

ea.  $i = 1290;$

Sop (" $i = " + ea. i);$

ea. test 1();

Sop ("Program ends . . .");

}

OP:

Program starts . . .

$j = 12$

running test 2() . . .

\* \* \* \* \*

$i = 10;$

running test 1() . . .

\* \* \* \* \*

$i = 1290$

running test 2() . . .

program ends.

### Example 3: P38

QUESTION 30A-10-06

class F

{

int i=120; // Global variable,  
non static member

void test1()

{

System.out.println("Running test1()...");

int i=187; // local variable

System.out.println("i = " + i);

}

public String[] args)

{

System.out.println("Program starts...");

F f1 = new F();

System.out.println("i = " + f1.i);

f1.test1();

System.out.println("Program ends...");

}

}

O/P:

Program starts...

i=120

Running test1()...

i=187

Program ends...

→ Here 'i' is local variable,  
which gets load into stack  
memory when

## Abundant Object

Example 4: P39

class G

```
int j = 187  
static void test1()
```

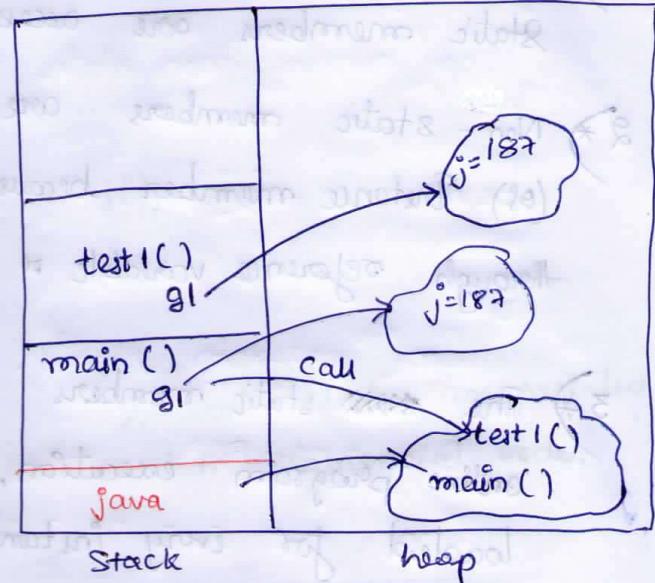
```
System.out.println("running test1() method...");  
G g1 = new G(); // local  
System.out.println("j=" + g1.j);
```

?

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

```
System.out.println("Program starts...");  
G.g1.test1();  
System.out.println("Program ends...");
```

```
[ G g1 = new G(); ]  
[ System.out.println("j=" + g1.j); ]
```



\* Any object which doesn't have a reference variable is called ~~abundant~~ abundant object.

## Summary:

- 1\*) Static members are also known as class members because static members are accessed through class.
- 2\*) <sup>1st</sup> Non-static members are also known as Object members  
(or) Instance members because these members are accessed through reference variable or objects.
- 3\*) The static members will be loaded only once for entire program execution, whereas non-static members are loaded for every instance of the Class.
- 4\*) If a member of an instance is changed that change will not reflect in the other instance.
- 5\*) The static members can be shared across the Objects.
- 6\*) During execution a method enters into stack to execute all the statements of method. The local variables will be residing inside stack memory.
- 7\*) The life of local variable, is as long as the method stays in stack.
- 8\*) All objects will be created in heap, the reference to the object can be in the heap or in the stack.
- 9\*) If any object exist without reference variable then such objects are known abundant variable