Syllabus

Selenium Webdriver :

ECLIPSE : ——— Pg 51

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 [Datastructures] : ——— Pg

7 * Exception Handling : ——— Pg

8 * File Handling : ——— Pg

9 * Threads : ——— Pg

API Classes

1* JUnit / Test NG

2* Ant Tool

3* Log4j

4* Html/CSS

5 - 6 classes

SELENIUM

1* Introduction to Selenium IDE & Selenium RC

2* Intro Selenium WebDriver

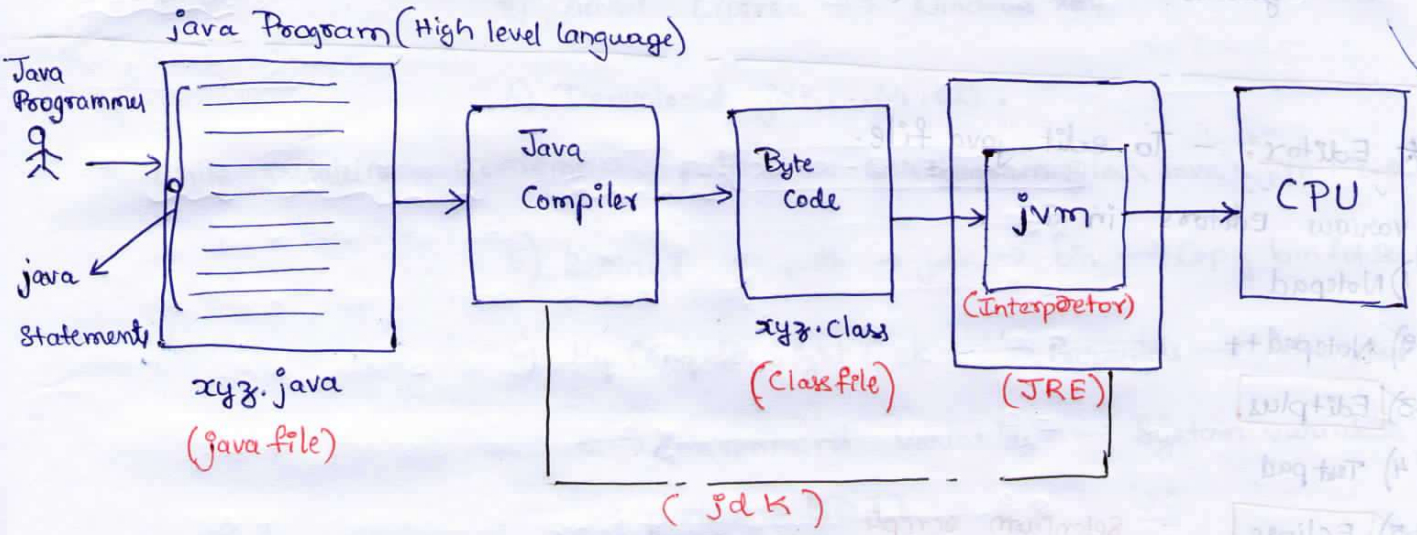
3* Frameworks

20 - 25 classes

08-01-2013 | Tuesday

CORE JAVA

CHAPTER 1 : BASICS



NOTES :

1) * All java statements are saved as javafile with extension '.java'.

2) * Java Compiler generates Classfile based on javafile.

3) * Class file contains bytecode, which is understood only by jvm [Java Virtual Machine].

4) * jvm is a interpretor, 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) Test pad
5) Eclipse - Selenium script
6) NetBeans

2*) How To Install:

Step1 : To Install 1) Editor - editplus
2) Java Compiler
3) Jre
] — Jdk [Java development Kit]

Step2: Version : jdk - 1.5
jdk - 1.6
jdk - 1.7 - latest

Step3: 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\jdk1.7.0_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 → put ; & then paste copied bin folder 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*) **Identifier** : * 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 Alphabets / starting character should be alphabet only.

Ex: 1) int age;
2) double salary1;

} all variable names, method names, 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

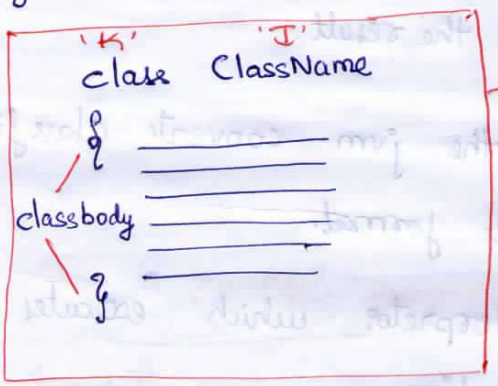
6*) Coding :

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



Its 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 Classname.

8*) Compilation :

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

8.2*) The class file containe 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 javafile is available.

8.5*) The Syntax to compile any java file 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
```

NOTE

10*) jvm starts execution only if java class containe main method otherwise jvm throws error.

11*) A java class which doesn't have a main method connot 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 Language

---

2) P2:

```
class SampleProgram1
{
  psvm(String[] args)
  {
    System.out.print("Hi, Welcome to java \n");// \n is to print in newline
    System.out.print("Java is powerful programming language");
  }
}
```

P3:

```
class SampleProgram2
{
  psvm (String[] args)
  {
    System.out.println("Hi, Welcome again " + "and again to Java");
                                    // '+' To join two String data with
                                    seperate " "
  }
}
```

O/P:-

Hi, Welcome again and again to Java

P4:

```
class Demo1
{
  psvm( )
  {
    Sop (10 + 10);      // here + adds 10+10 first and then
                           displays the result
  }
}
```

O/P:- 20

P5:

```
class Demo2
{
  psvm(String[] args)
  {
    Sop ("Program starts...");
    int id;  // declaration
    id = 2012345; //initialization
    Sop (id);  // prints id value
    Sop ("Program ends...");
  }
}
```

O/P:
Program starts...
2012345
Program ends...

NOTE: 13) Before using a variable, we need to initialize, before initializing we need to declare the variable.

P6:
6) class AssignProgram1
{
public static void main(String[] args)
{
Sop("Program starts...");
int num1;
int num2;
int sum;
num1 = 10;
num2 = 20;
sum = num1 + num2;
Sop(sum);
Sop("Program ends...");
}
}

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

P7:
7) class Demo4
{
psvm( )
{
Sop("Program starts...");
final int empID; // final
empID = 12973;
Sop("empID is "+empID);
Sop("Program ends...");
}
}

O/P: Program starts...
empID is 12973
Program ends...

P8
8) class Demo3
{
psvm( )
{
Sop("Program starts...");
double salary = 15000.98;
Sop("Salary is "+salary);
salary = 25128.54;
Sop("Salary is "+salary);
salary = 35978.54;
Sop("Salary is "+salary);
Sop("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, the such variable has to be declared as final by using final keyword

11-01-2013 | Friday

Variables

15*) In java variables are used to store the data of programmes

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. Inorder to store String value java provides a class String, StringBuffer & StringBuilder.

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

9) 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 = 3.14;
   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 = 10000 l;
    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 2yrs
duration is 3000
Program ends...

11) PTI

```
class AssignDegToFahrenheit
{
    psvm(String[] args)
    {
     Sop("Program starts...");
     double Fahrenheit;
     double DegreeCelsius = 28;
     Fahrenheit = (DegreeCelsius * 1.8) + 32;
     Sop( DegreeCelsius + "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

P12

1) If - Else Statements

```
1. class Demo3
{
   public static void main(String[] args)
   {
     S.o.p("Program Starts...");
     final int empID=1029;
     If (empID == 1021);
      {
        S.o.p("This empID belongs to Project manager");
      }
     else{
        S.o.p("This empID doesn't belongs to Project manager");
     }
     S.o.p("Program ends...")
   }
```

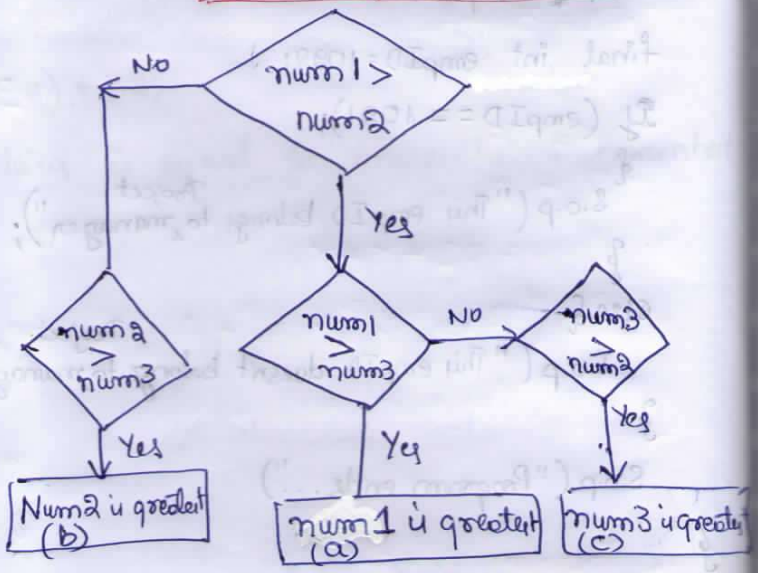
P13

```
2. Class Demo 4
   psvm (String[] args)
{  S.o.p ("Program Starts...");
   final int empID = 1026;
   if (empID == 1021)
   {
     S.o.p ("This empID belongs to Project Manager");
   } else if (empID==1022)
          {
            S.o.p ("This empID belongs to Team Lead");
   } else
   {
     S.o.p ("This empID doesn't belong to PM or TL");
   }
     S.o.p ("Program ends...");
}
```

Assign

1. Declare 3 variable integer type and display higheit of the three.

		(And) &&	(or) ||
T	T	T	T
T	F	F	T
F	T	F	T
F	F	F	F



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.op ("Program ends...");
}
```

(or)

```
if( a >b)
{   if( a>c)
    {   (a is highest);
    }else if (c>b)
    {   (c is highest);
    }
else
{   (b is highest);
}
```

P15

```
class Demo5

psvm (String[] args)
{
 S.o.p ("Program starts...");
 int empID = 10213;
 // loops
 for (int i=0; i<5 ; i++)
 {
  S.o.p ("value of empID is "+empID);
 }
 S.o.p ("Program ends...");
}
```

Loops

for loop:- [execute intended number of times]

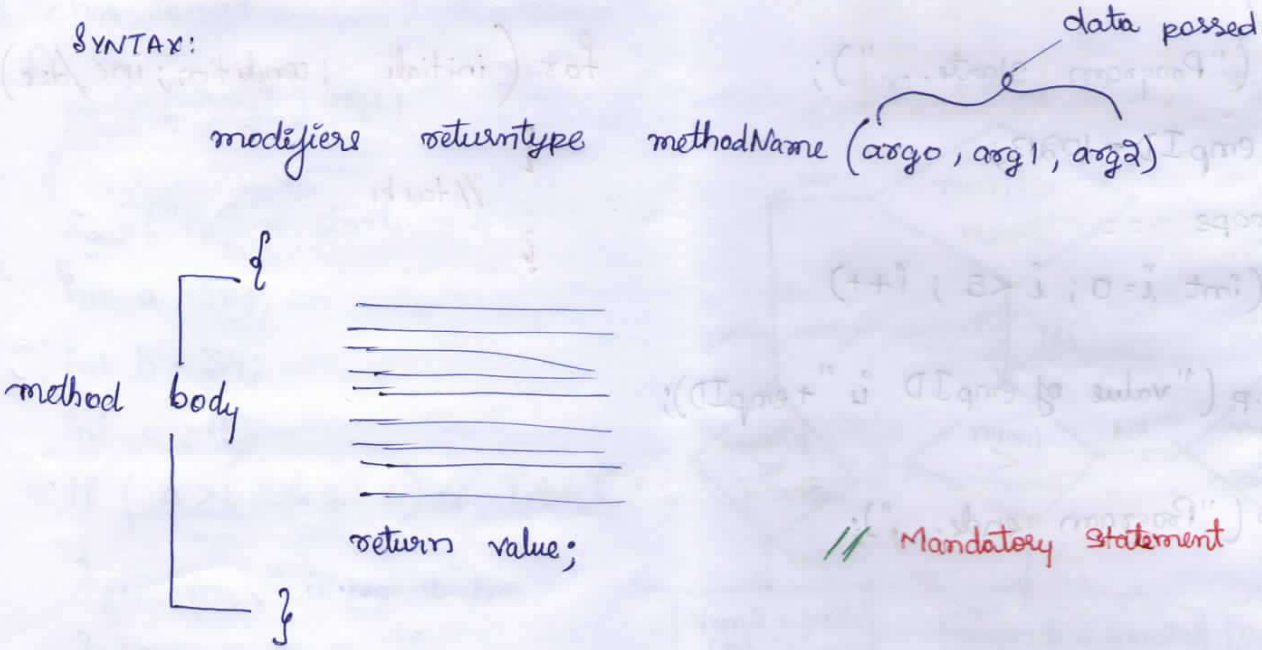
```
for ( initial ; condition ; inc/dec)
{
   // tasks
}
```

16-01-2013 WEDNESDAY

* Modification of Code line number 6

METHODS :

SYNTAX:



Example:

```
P16
class Demo1
{
   psvm (String[] args)
   {
    S.o.p("Program Starts...");
6.* Sample();          // invoking method (or) Calling method
    S.o.p ("Program ends...");
   }
}
 Static int sample();
 {
   S.o.p("running Sample method");
   return 10;
 }

}
```

o/p - Program Starts...
running Sample method
Program ends...

```
*
6. int ret = sample();
7  S.o.p (" value of ret is "+ret);
```

o/p = Program starts...
running sample method
value of ret is 10
Program ends...

Example 2:

P17

```
class Demo2
{
  psvm (String[] args)
  {
   Sop ("Program starts...");
   sample();
   Sop ("Program ends...");
  }
  static void sample()
  {
    Sop. out. println ("running sample method");
    // compiler writes return statement here
  }
}
```

I.Q

Is return statement mandatory in a method in java?

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

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

Example 3:

P18

```
class Demo3
{
  psvm (String[] args)
  {
   Sop ("Program starts...");
   int res = sample (12, 34);
   S.o.p ("result is "+res);
   S.o.p ("*******");
   int res = sample (23, 64);
   Sop ("result is "+res);
   Sop ("Program ends...");
  }
```

PTO

```
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;
}

}
```

O/P:

```
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

29*) While developing programs the repeatative tasks/operation core develop as method.

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

24*) The syntax to develop a method in java is

```
modifiers returntype 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 returned.
return value should match return type.

27*) Args ~~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
{ psvm(String[] args)
  {
    sop("Program starts...");

    double res = SimpleInterest(10000, 0.15, 2);
    Sop("Simple Interest is "+res);
    Sop("*********");
    double res = degreeToFahrenheit(28);
    Sop("Fahrenheit is "+res);
    Sop("**********");
    double area = areaOfCircle(20);
    Sop("Area of Circle is "+area);
    Sop("**********");
    Sop("Program ends...");
  }
```

```
Static double simpleInterest (double a, double rate, int time)
{
 System.o.p("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;
  S.o.p (" degreeCelsius + " Degree Celsius is equal to ");
  return res;
  }
  Static double areaOfCirde (double radius)
  {
    S.o.p ("Running area of Cirde method");
    final double PI= 3.14;
    double area = PI * radius * radius;
    S.o.p (" For radius "+radius);
    return area;
   }

}
```

17-01-2013 | THURSDAY

```
int i = 10;
```

```
i = i + 1;
```
→ binary operator
operand

* Unary operater:

* on only one operand

++ → increment
-- → decrement

→ post [use current value first & then increment]
→ pre

Example 1:

P20

```
class Demo5
{
  psvm (String[] args)
  {
    System.o.p("Program starts...");
    int i = 1;
    Sop(" i value bef inc "+i);
1*  ++i;                          // pre increment
    Sop("i value after inc "+i);
    S.o.p("Program ends");
  }
}
```

O/P:-
Program starts...
i value bef inc is 1
i value aft inc is 2
Program ends...

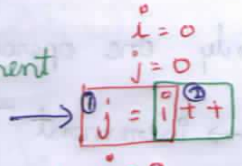
```
2*  i++;                          // post increment
    Sop("i value after inc "+i);
```

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...");
    }
}
```

// post increment



1. j = 0
2. i++ = 0+1 = 1
   ⇒ i = 1

O/P:

```
Program start...
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..."),
    }
}
```

//

j = i++ + 1 ; i = i++

j = 0 + 1 = 0++

j = 1 ; i = 1

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...");
  }
}
```

i = 0++ = 1

j = 0++ + 1
j = 0 + 1
j = 1

i = 1

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

Example 5:- P24

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

j = 0 + 1 + 2
j = 3

O/P:- Program starts...
value of i = 2
value of j = 3
Program ends...

Example 6 :-

P25

```
Class Demo9
{
  psvm (String[] args)
  {
  Sop ("Program starts...");
  int i = 0;
  int j = 0;
  j = ++i;                    // 1. ++i = 1  2. => 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 (1+2)

j = 0 + 1 + 1 + 3 i = 3

O/p :- Program starts...
value of i is 3
value of j is 5
Program ends...

Example 8 :

P27

```
class Demo10
{
  psvm (String[] args)
  {
    Sop ("Program starts...");
    int i=12;
    test1 (i++);
    Sop ("i = "+i);
    Sop("Program ends...");
  }
  static void test1 (int a)
  {
    Sop ("a="+a);
  }
}
```

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

O/P: Program starts...
a = 12
i = 13
Program ends...

Example 9 : P28

/* test1 (++i);

O/P:- Program starts...
a=13
i=13
Program ends...

Example 10 : P29

```
class Demo11
{
  public svm (String[] args)
  {
    Sop("Program starts...");
    int i=1;
    int j = test1(i)
    Sop ("i= "+i);
    Sop ("j= "+j);
```

```
    Sop("Program ends...");
  }
  static int test1(int a)
  {
    System.out.println("a = "+a);
15*  return a++;          // 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
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)
{
  Sop ("a = "+a) //2
  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*) Poe increment - means increment current value & use final value after increment.

34*) Same for post decrement & pre decrement only diff 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;                    // also for float f
  static char c;
  static boolean b;
  psvm (String[] args)
  {
   Sop ("Program starts...");
   Sop ("i ="+i);
   Sop ("d="+d);
   Sop ("c="+c);
   Sop ("b="+b);
   Sop ("Program ends...");
  }
}
```

O/p :- Program starts...

i=0

d=0.0

c=

b=false

Program ends...

Example 2:

P32

```
class Demo13
{
    static int i=12;
    psvm (String[] args)
    {
     Sop("Program starts ...");
     Sop(" i ="+i);     // referring global
     i=24;              // global
     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

CLASS MEMBERS [Methods & Variables]

- Static (single copy)
- non static (multiple instance)

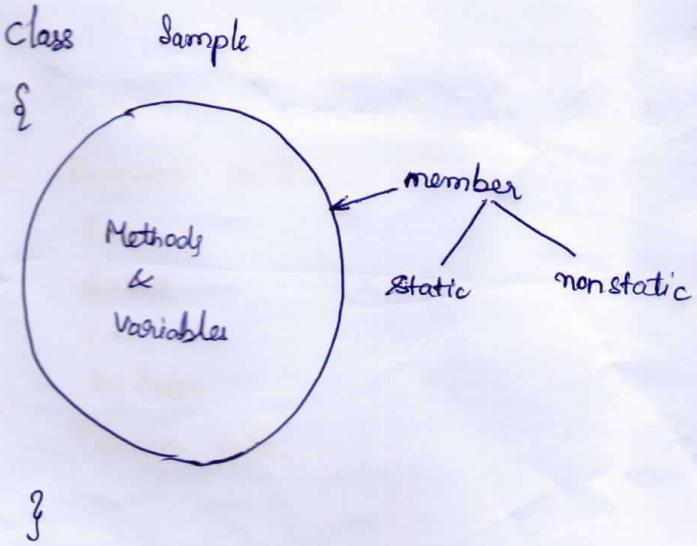
Members of Class:

1*) The members of class are catagerized as static members and non static members.

2*) Static members are declared with static keyword where as non static members are declared without any keyword.

3*) Static members of class can be accessed by using Classname.

4*) Non static members are accessed by using Referencevariable.



Example 1:

P33

```
class MemberA
{
  static int i;
  static double d;
  psvm (String[] args)
  {
    Sop ("Program starts...");
    MemberA.display(); // Accessing static member of MemberA
    Sop ("----------");
MemberA.i = 10;
MemberA.d = 23.45;
    MemberA.display();
    Sop ("Program ends...");
  }
  static void display()
  {
    Sop ("i=" + MemberA.i);
    Sop ("d=" + MemberA.d);
  }
}
```

O/P: Program starts...

i = 0

i = 0.0

---

i = 10

i = 23.45

Program ends...

Variables

Primitive:

Stores only primitive value

Reference Variable

① Create reference Variable

```
ClassName RefVariableName;
```

declaration

```
Variablename = new classname();
```

Initialization

Reference Variable:

A reference variables are created using Classname.

Syntax given below

```
Classname refvariable = new Classname();
```

```
refvariable.MemberName
```

P34

```
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()...");
  }
}
```

O/p: Program starts...
k=0
s=0.0
running test1()...
Program ends...

P35

```
class ClassC
{
    static int a;
    double b;
    psvm (String[] args)
    {
     Sop ("Program start...");

Line#
9.  Sop ("b = "+ c1.b);          11. ClassC.sample1();   //Line# 10,11
10. Sop ("a = " + ClassC.a);     12. c1.sample2();       accessing static members
13. Sop ("Program end...");                              //Line# 9,12
    }                                                    accessing non static members
                                                         of ClassC
    static void sample1()
    {
      Sop ("running sample1()...");
    }
    void sample2()
    {
      Sop ("running sample2()...");
    }
}
```

O/p :-

22-01-2013 | Tuesday

java = jvm

Read/write memory

* write/read multiple times.

Example: RAM, USB, HDD etc

Read only memory

* write once.
* read many times.

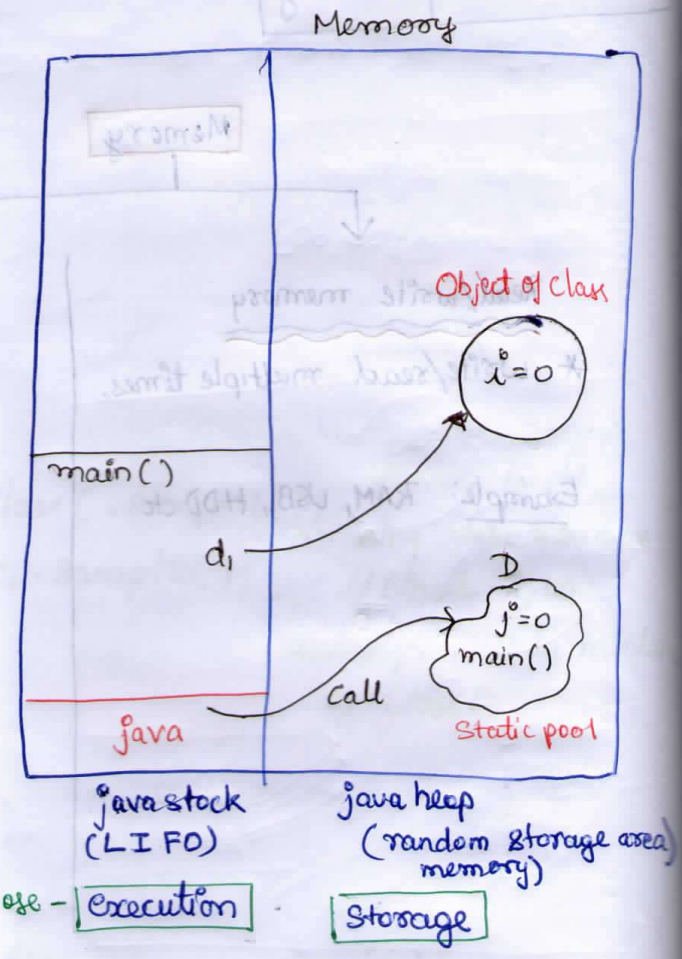
Example: cd ROM
DVD ROM
:
:

Read/write memory

RAM	HDD
1. 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 get allocated. This memory is divided into 2 area
1. java stack
2. java heap

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

3*) Javaheap 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 needs 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 enterunters Object creation Statement, the Objects 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 Objectreference.

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

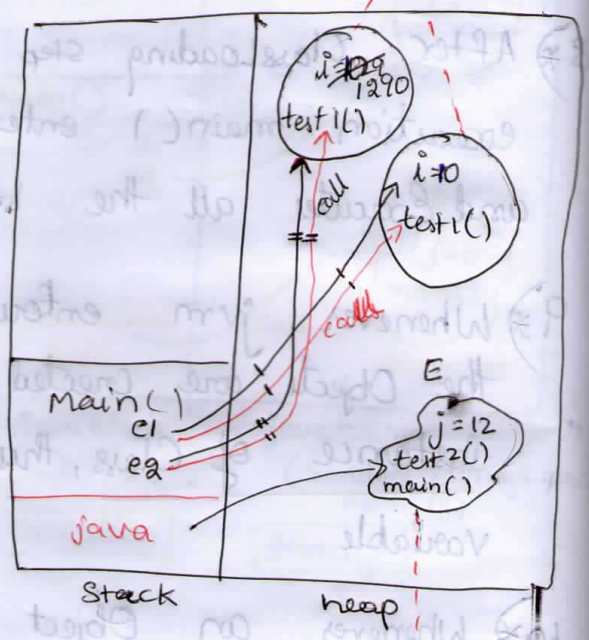
heap memory. After this java get out of stack and releases the memory back to Main Memory [RAM].

Example 2 P37

(multiple instances of class)

[Object/instance of Class]

```
class E
{
    int i=10;
    static int j = 12;
    void test1()
    {
        Sop ("running test1()...");
    }
    static void test2()
    {
        Sop ("running test2()...");
    }
    psvm(String[] args)
    {
        Sop ("Program starts...");
        // accessing static members
        Sop ("j = "+E.j);
        E.test2();
        Sop ("* ** ** *** *");
        // non static members
        E e1 = new E();
        Sop ("i = "+e1.i);
        e1.test1();
        Sop("* ** ** *** *");
        E e2 = new E();
        Sop ("i=
```



(only one copy)

```
        e2.i = 1290;
        Sop ("i = "+e2.i);
        e2.test1();
        Sop ("Program ends...");
    }
}
```

O/P:

```
Program starts...
j=12
running test2()...
* * * * * * * * *
i=10;
running test1()...
* * * * * * * * *
i=1290
running test2()...
program ends.
```

Example 3: P38

```
class F
{
  int i=120;  // Global variable, non static member
  void test1()
  {
    Sop ("running test1() ...");
    int i = 187; // local variable
    Sop (" i = " + i);
  }
psvm (String[] args)
{
  Sop ("Program starts...");
  F f1 = new F();
  Sop ("i=" + f1.i);
  f1.test1();
  Sop ("Program ends...");
  }
}
```

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

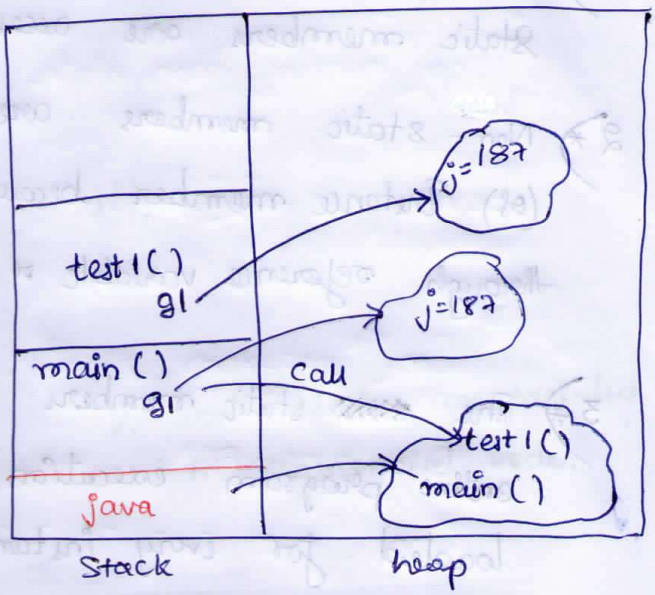
O/P:

```
Program starts...
i=120
running test1()...
i=187
Program ends...
```

Abundant Object

Example 4: P39

```
class G1
{
    int j = 187;
    static void test1()
    {
        Sop("running test1() method...");
        G1 g1 = new G1(); // local
        Sop("j = " + g1.j);
    }
    psvm (String[] args)
    {
        Sop("Program starts...");
        G1.test1();
        Sop("Program ends...");
    }
}
```



* Any object which doesn't have a reference variable is called ~~abandant~~ abuntant object.

```
G g1 = new G1();
Sop("j = " + g1.j);
```

Summary:

1 *) Static members are also known as class member because Static members are accessed through class.

2 *) Non - static members are also known as Object members (or) Instance member because these member 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 changes 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

Blocks:

Example P40

Jspiders\Selenium\corejava\blocks ← create new folder.

```
Class A
{
   static
     {
       Sop("running static block-1");
     }
   psvm(String[] args)
   {
     Sop("Program starts...");
     Sop("Program ends...");
   }
   static
   {
     Sop("running static block-2");
   }
}
```

// static blocks are executed in sequencial order

O/P:

```
running static block-1
running static block-2
Program starts
Program ends
```

* When

P41

```
class B
{
    {
      Sop("running non static block-1");
    }

    psvm ( String[] args)
    {
      Sop("program starts...");
      B b1 = new B();          // non static blocks are executed during object creation
      Sop ("- - - - - - - - -");
      B b2 = new B();
      Sop ("program ends...");
    }

    {
      Sop ("running non static block-2");
    }
}
```

O/P:

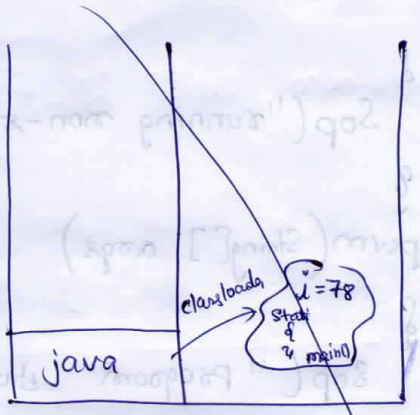
```
Program starts...
running non static block-1
running non static block-2
- - - - - - - - - -
running non static block-1
running non static block-2
Program ends...
```

```
class C
{
  {
    Sop("running non-static block");
  }
  psvm(String[] args)
  {
    Sop("Program starts...");
    C c1 = new C();
    Sop("program ends...");
  }
  static
  {
    Sop("running static block");
  }
  static
  {
    Sop("running static block");
  }
}
```

O/P:

```
running static block
running static block
Program starts...
running non-static block
Program ends...
```

```
class D
{
    static int i = 78;
    static
    {
        i = 29;
    }
    psvm (String[] args)
    {
        Sop ("program starts...");
        Sop ("i = " + D.i);
        Sop ("program ends...");
    }
}
```



P43

```
class D
{
    static int i = 78;
    int j = 557;
    static
    {
        i = 29;
    }
    {
        j = 4354
    }
    psvm ( )
    {
        Sop ("Program starts...");
        Sop ("i = " + D.i);
        Sop ("- - - - - - - - - - ");
```

```
        D d1 = new D();
        Sop ("j= " + d1.j);
        Sop ("Program ends...");
    }
}
```

O/P:

```
Program starts...
i=29
- - - - - - -
j= 4354
Program ends...
```

Summary:-

1*) blocks are classified into 2-types;

1.1) static block

1.2) Non-static block

2*) static block: 2.1*) gets executed before running the main() in a sequential order.

2.2*) We can develop multiple static block, all static blocks will be executed in a sequenctial order.

2.3*) Static block are used to initialize the static variable of Class.

3*) Non-static block: 3.1*) gets executed at the time of Object Creation.

3.2*) We can develop multiple non-static blocks in Class

3.3*) All non static blocks gets executed in a seq order.

3.4*) For every instance of Class non-static block gets executed.

3.5*) Non static blocks are used to initialize non static variables of Class.

---

* * * * * * * *

P44

```
class E
{
    int i;
    double d;
    void print()
    {
      Sop("i=" + i);
      Sop("d=" + d);
    }
    psvm(String[] args)
    {
      Sop("program starts...");
      E e1 = new E();
      e1.print();
      Sop("program ends...");
    }
    {
      i = 218;
      d = 876.23;
    }
}
```

O/p :

```
Program starts...
i = 218
d = 876.23
Program ends...
```

P45

```
class Employee
{
    final static String compName = "Jspiders";
    int empID;
    double empSalary;
    void dispEmpDetails()
    {
        Sop(empID + "\t\t" + empSalary);
    }
    psvm(String[] args)
    {
        Sop("******");
        Sop("Employees of "+compName);
        Sop.();
        Sop("Employee ID \t Employee Salary");
        Sop("------\t-------");
        Employee empl = new Employee();
        empl.empID = 6263;
        empl.empSalary = 23789.788;
        empl.dispEmpDetails();

        Employee emp2 = new Employee();
        emp2.empID = 8923;
        emp2.empSalary = 34789.788;
        emp2.dispEmpDetails();

        Employee emp3 = new Employee();
        emp3.empID = 9198
        emp3.empSalary = 54657.788
        emp3.dispEmpDetails();
        Sop("******");
    }
```

}

O/p :

```
* * * * * * * * * * *
Employees of Jspiders
Employee ID        Employee Salary
-----------        ---------------
8263               23789.788
8923               34789.788
2198               54657.788
* * * * * * * * * * *
```

Assignment

1. List the difference between 1) static & non static members. 7 points
   2) static & non-static blocks.

Summary

1. Inside static context, a static member can be accessed directly, where as non static member should be accessed through reference variable.
2. Inside non-static context, both static & non-static member can be accessed/referred directly.
3. Static block is also known as static initialization block. where non static block is also known as instance Initialization block.

Jspiders Selinium \ corejava \ Basic Blocks \

java files [src] ——————————

class files [bin] (or) [class] ——————————

Create two new folders inside blocks folders

```
javac -d ..\bin ClassName.java
```

→ Syntax to save class file in particular location. [folder = bin]

```
class Demo1
{
  psvm( )
  {
    Sop("running Demo1...");
  }
}

class Demo2
{
  psvm( )
  {
    Sop("running Demo2...");
  }
}
```

Save as → "Demo1.java" (or) "Demo2.java"

"ClassName.java"

Here ClassName should be any of the class present in java file.
ie Demo1 or Demo2

```
class Demo4
{
    psvm( )
    {
        Demo3 d1 = new Demo3();
        Sop("i="+d1.i);
        Sop("d="+Demo3.d);
    }
}
```

P46

```
class Demo3
{   // business class:
    int i = 120;
    static double d=76.21;
}

class Demo4
{
    psvm( )
    {   //main class
        Sop("Program starts...");
        Sop("d="+Demo3.d); //referring
        // member of class Demo3
        Demo3 d1 = new Demo3();
        Sop("i="+d1.i);
        Sop("Program ends...");
    }
}
```

O/p: Program starts...
d = 76.21
i = 120
Program ends...

Summary :-

1*) A java file can have any number of class definition blocks, whenever such java files are compiled, the compiler generates .class file for each class definition block.

2*) Inside a class, we can refer member of another class based on member type.
Static are referred by Class ref. whereas non-static are referred by reference variable.

Assignment :

Difference between :-

	Static member	Non-static member
1.	Static members are also known as Class member	Non-static members are also known as Object members.
2.	Static members are loaded only once for entire program execution	Non-static members are loaded for every instance of Class.
3.	If static members are changed, it is changed for entire program	If a non-static members of an instance is changed, that change will not reflect in the other instance of Class.
4.	Static members can be shared across objects	Non-static members cannot be shared across the Objects
5.	Static members are loaded into heap by ClassLoader.	Non-static members are loaded into heap when jvm encounter Object creation statement.
6.	All static members reside in common area known as static pool.	Non static members are loaded into Object Memory after Object creation.
7.	Static member are accessed through ClassName.	Non-static members are accessed by using reference variable.

2.

Static block	Non-static block
1. Gets executed before running the main() - in a sequential order	1. Gets executed at the time of Object Creation.
2. static blocks are used to initialize the static members of Class	2. Non-static blocks are used to initialize non static variables of Class.
3. static block gets executed only once for entire Program execution.	3. For every instance of Class non-static block gets executed.
4. Inside static block static member can be accessed directly where as non static member should be accessed through reference variable.	4. Inside non static block, non static members and static members can be accessed directly.
5. Static block is called as static initialization block	5. Non-static block is called as instance initialization block.

28-01-2013 | MONDAY

CHAPTER

Constructors

P47

1)

```
class A
{
  int i=10;
  A()
  {
    Sop("running class A constructor");
  }
}

class Run1
{
   psvm( )
   {
    Sop("program starts...");
    // refer i member of class A
    A a1 =new A();
    Sop("i="+a1.i);
    Sop("program ends...");
   }
```

O/p:

```
Program starts...
running class A constructor
i=10
Program ends...
```

Syntax of Constructor:-

```
ConstructorName()
{
  Constructor body
}
```

1*) A constructor is a block just like method which is invoked at the time of Object Creation.

2*) The constructor should be written with above Syntax.

3*) The name of constructor should be same as Class Name.

4*) The constructor body should not have return statement

5*) The constructor should not be declared with any return type.

6*) Whenever a constructor is invoked, the body of constructor get executed.

7*) A constructor is developed to initialize Object variables at the time of Object Creation.

8*) Every java class should have a constructor, if we don't develop a constructor compiler writes a constructor known as default Constructor

2) P48

```
class B
{
    double d=3.4;
    B()
    {
        Sop("running Class B constructor");
        d=12.45;
    }
}

class Run2
{
    psvm()
    {
        Sop("Program starts...");
        B b1 = new B();
        Sop("d="+b1.d);
        Sop("----------");
        B b2 = new B();
        Sop("d="+b2.d);
        Sop("Program ends...");
    }
}
```

O/p :-

```
Program starts...
running class B constructor
d=12.45
----------
running class B constructor
d=12.45
Program ends...
```

9.* Default Constructor doesn't have any arguments
The body of the default will have mandatory statements.

10.* While developing a constructor we can pass a value as a argument.

11.* A constructor which is defined with arguments are known parameterized constructor (or) argument constructor.

12.* Developing more then one constructor in a Class is known as constructor overloading (or) Overloaded Constructor.

In order to develop this, following critirea should be fulfilled.

12.1.* The argument list/parameter should differ interms of datatype.

12.2.* The arguments should differ interms of number of args.

12.3.* The argument should differ with respect to position

13.* In order to develop Overloaded Constructor any of the above should be fulfilled.

3)

```
Class C
{
  double d=3.4;
   C (double a)
   { System.out.p("running C() constructor");
     d=a;
   }
}

class Run3
{
  psvm ( )
  {
   Sop ("Program starts...");
   C c1 = new C(2.3);
   Sop ("d="+c1.d);
   Sop ("----------");
   C c2 = new C(34.21);
   Sop ("d="+c2.d);
   Sop ("Program ends...");
  }
}
```

O/P:

```
Program starts...
running C() constructor
d=2.3
----------
running C() constructor
d=34.21
Program ends...
```

a Class is to be created with different implementation or data we go for constructor Overloading.

15*) If we develop any type of constructor inside the Class, then compiler will not write default Constructor

" Compiler writes default Constructor only if Class doesn't have any type of Constructor "

```
class D
{
  int i;
  double d;
  //overloaded constructor
  D(int a)
  {
    Sop("running D(int) constructor");
    i=a;
  }
  D(double b)
  {
    Sop("running D(double) constructor");
    d=b;
  }
  D(int a, double b)
  {
    Sop("running D(int, double) constructor");
    i=a;
    d=b;
  }
  void println()
  {
    Sop("i="+i);
    Sop("d="+d);
  }
}
Class Run4
{
  psvm()
  {
    Sop("Program starts...");
    D d1 = new D(12);
    d1.println();
    Sop("----------");
    D d2 = new D(6.32);
    d2.println();
    Sop("----------");
    D d3 = new D(78, 34.53);
    d3.println();
    Sop("Program ends...");
  }
```

continued program...

O/P :-

```
Program starts...
running D(int) constructor
i=12
d=0.0
----------
running D(double) constructor
i=0
d=6.32
----------
running D(int, double) constructor
i=78
d=34.53
Program ends.
```

29-01-2013 | Tuesday

5) P51

```
class E
{
   { Sop("running non-static block");
   }
   E()
   { Sop("running E() constructor");
   }
}

class Run6
{
  psvm()
  {
   Sop("Program starts...");
   E e1 = new E();
   Sop("------------");
   E e2 = new E();
   Sop("Program ends...");
  }
}
```

O/P:

```
Program starts...
running non-static block
running E() constructor
------------
running non-static block
running E() constructor
Program ends...
```

1*) If a Class refers member of another Class, then first static block of referred Class will be executed.

If the current/running Class contains static block, then the static block of the running Class will be executed first, then static block of referred Class will be executed.

2*) If in a running Class a reference is made through Object, then the static block of the running Class gets executed first, then static block of referred Class will be executed first and then non-static block of referred Class will be executed and then constructor body will be executed.

3*) A constructor invokes a non-static block whenever an Object is created.

6) P52

```
class E
{
  static
  {
    Sop("running static block of class E");
  }
  {
    Sop("running non-static block ");
  }
  E()
  {
    Sop ("running E() constructor");
  }
}
class Run6
{
  static
  {
    Sop("running static block of Class Run6");
  }
  psvm ( )
  {
    Sop("Program starts...");
    E e1 = new E();
    Sop("-----------");
    E e2 = new E();
    Sop("Program ends...");
  }
}
```

O/p:

```
Running static block of Class Run6
Program starts...
Running static block of Class E
Running non-static block
Running E() constructor
-----
running non-static block
running E() constructor
Program ends...
```

this keyword

P53

1)
```
class F
{
  int i;
  double d;
  F(int i, double d)
  {
   Sop("running F() constructor...");
   this.i = i;
// global i = local i
   this.d = d;
  }
  void println()
  {
   System.Op("i=" + i);
   Sop("d=" + d);
  }
}
Class Run7
{
   psvm( )
   {
   Sop("Program starts...");
   F f1 = new F(123, 78.56);
   f1.print()
   Sop("Program ends...");
   }
}
```

O/P:
```
Program starts...
running F() constructor...
i=123
d=78.56
Program ends...
```

1*) 'this' keyword refers to current Class instance, using 'this' keyword we can refers the non-static member of the current Class with the Class. 'this' keyword has to be used in constructor (or) inside non-static context.

2*) A constructor of a Class can Call another constructor of Same class, this can be done by using "this()" statement. Using 'this()' you can call any constructor of current Class.

Recursive constructor Call is not allowed in java.

3*) 'this()' must be the first statement in constructor body.

P54

2)

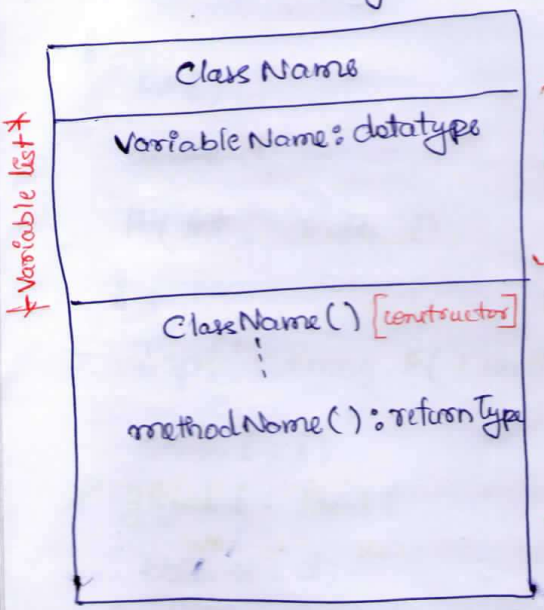
```
class G
{
  G()
  {
    Sop("running default() constructor");
  }
  G(int i)
  {
    this();
    Sop("running G(int) constructor");
  }
  G(double d)
  {
    this(23); // Statement
    Sop("running G(double) constructor");
  }
}

class Run8
{
  psvm()
  {
    Sop("Program starts...");
    G g1 = new G(54.34); //double type Constructor
    Sop("Program ends...");
  }
}
```

O/p: Program starts...
running default() constructor
running G(int) constructor
running G(double) constructor
Program ends...

30-01-2013 | WEDNESDAY

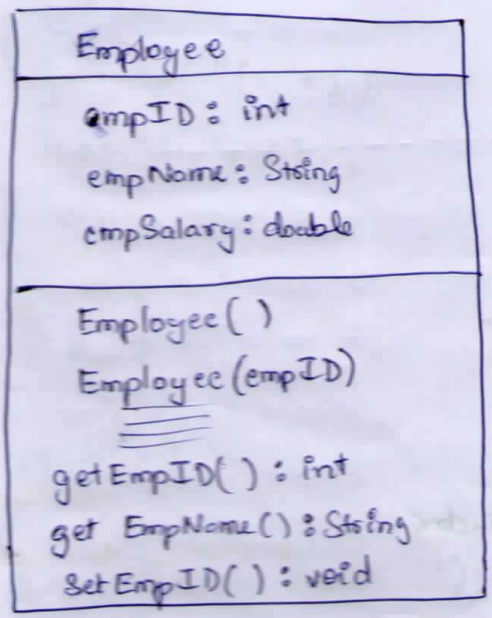
Class Diagram



1*) Class Diagram is pictorial representation of java Class, which describes the behaviour and properties of the Class.

2*) Class Diag are used while designing the relationship of Class.

Ex:



Assign: Develop a program for above Class Diagram

```
Class Employee
{
  int empID;
  String empName;
  double empSalary;

  Employee()
  { Sop("Employee List of Jspiders");
  }
  Employee(int empID)
  { Sop("EmpID="+empID);
  } this.empID = empID;
  Employee(int empID, double empSalary)
  { this.empID = empID;
    this.empSalary = empSalary;
  }
  Employee(String empName)
  { this.empName = empName;
  }
  void print()
  { Sop("Employee "+empName+"Emp
      ID "+empID+"with Salary"+
      empSalary);
  }
```

String