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)

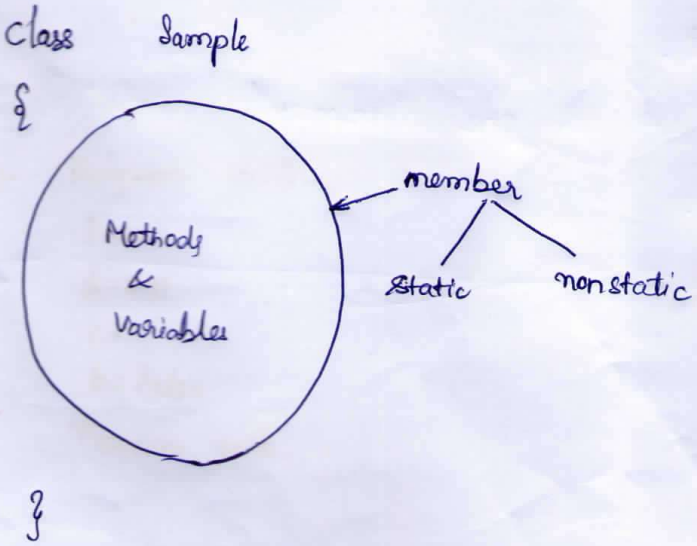
Members of Class:

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.

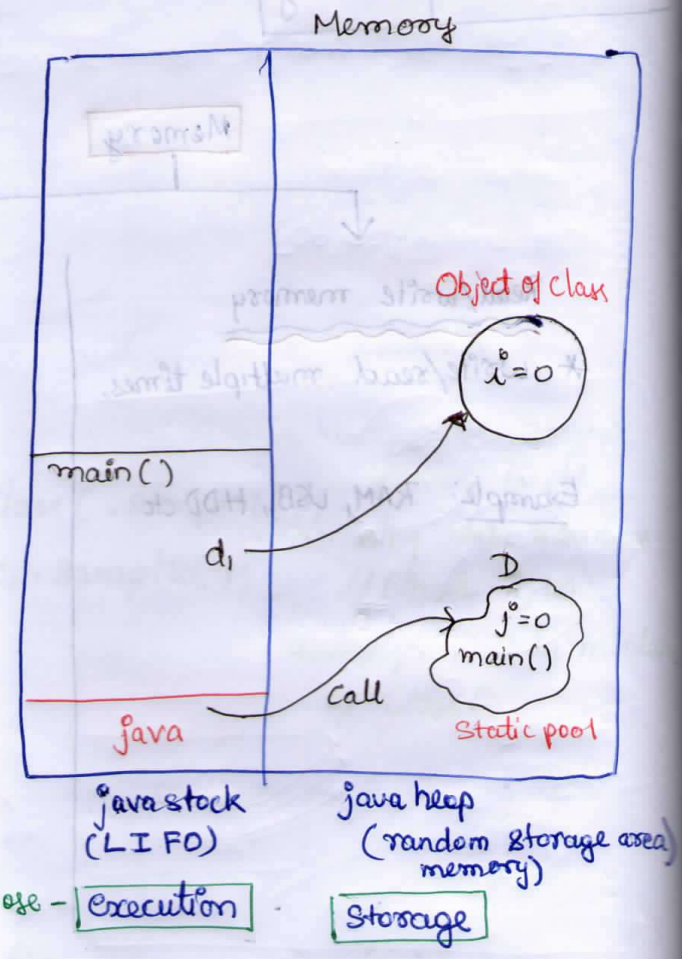
Example: cd ROM
DVD ROM
:
:

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

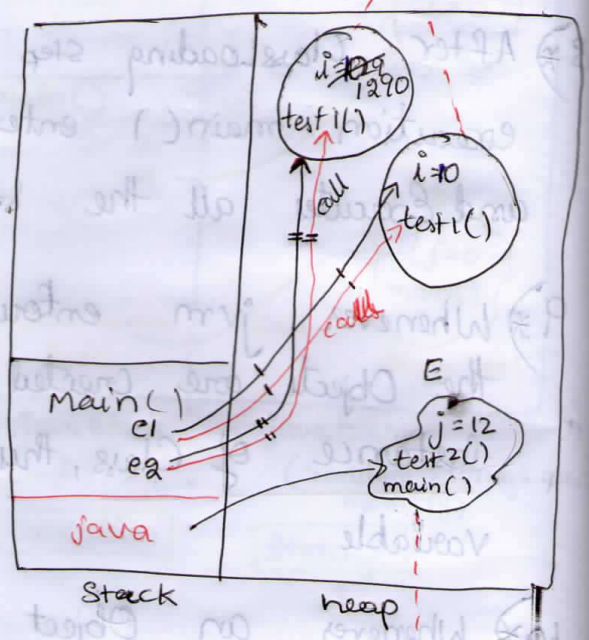
heap memory. After this java get out of stack and releases the memory back to Main Memory [RAM].

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

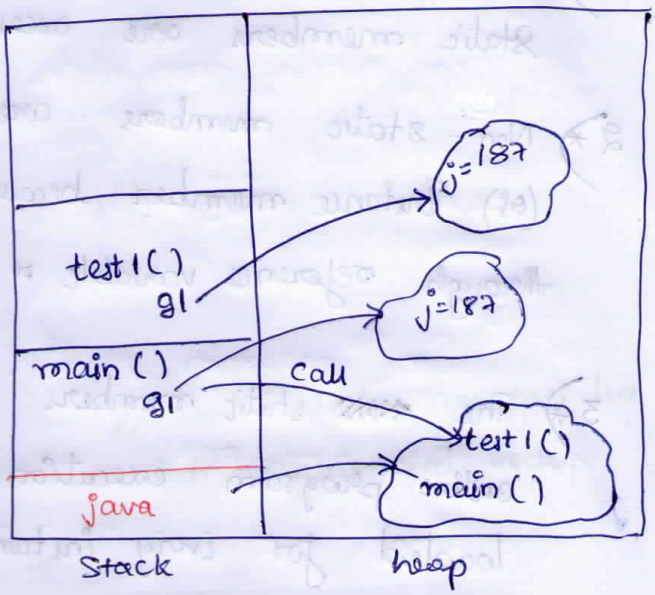
O/P:

```
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