static

- static member is defined for the class itself & exist independently of any object of that class

- We can declare both methods and variables to be static.

- To create such a member, precede its declaration with the keyword static.

- To call static member use the following syntax;

```
clsName.varName;
clsName.methodName(args);
```

Static variable :

Only one copy of variable is created.

For static variable,

int is initialized to 0,

float to 0.0,

boolean initialized to false,

String to null

```
class   FirstProgram
{
        static int i;
        static float f;
       static char c;
       static String s;
       static boolean b;
      public static void main(String args[])
    {       System.out.println("int i="+i);
            System.out.println("float f="+f);
            System.out.println("char c="+c);
            System.out.println("string s="+s);
            System.out.println("boolean b="+b);
    }
}
```

OUTPUT :

int i= 0
float f= 0.0
char c=
string s= null
boolean b= false

static method :

The Math class provides several static methods some are

```
static int max(int I, int j);
static int min(int I, int j);
static double pow(double x, double y);
static double sqrt(double d);
static int abs(int i);
```

```
Class  prg
{
        public static void main(String args[])
        {
                System.out.println(" largest : "+Math.max(5,10));
                System.out.println(" smallest : "+Math.min(5,10));
                System.out.println(" power : "+Math.pow(2,3));
        }
}
```

```
largest : 10
 smallest : 5
 power : 8.0
```

```
class StaticDemo
 {     static int  a = 42;
            static int  b = 99;
            static void callme()
            {          System.out.println("a = " + a);
            }
}
class StaticByName
{          public static void main(String args[])
            {
                        StaticDemo.callme();
                        System.out.println("b = " + StaticDemo.b);
            }
}
```

```
class Counter
{           int c1=0;
            static int c2;
           Counter()
           {            c1++;
                        c2++;      }
           void disp()
           {             System.out.print("First Counter : "+c1);
                         System.out.println("   Second Counter : "+c2);   }
}
class prg3
{           public static void main(String args[])
           {            Counter  o1 = new Counter();
                        o1.disp();
                        Counter  o2 = new Counter();
                        o2.disp()
                        Counter  o3 = new Counter();
                        o3.disp();
            } }
```

First Counter : 1 Second Counter : 1

First Counter : 1 Second Counter : 2

First Counter : 1 Second Counter : 3

```
class Counter
{           int c1=0;
            static int c2;
            Counter()
           {            c1++;
                        c2++;    }
           void disp()
           {             System.out.print("First Counter : "+c1);
                        System.out.println("   Second Counter : "+c2);    }
           }
class prg3
{          public static void main(String args[])
           {             Counter  o1 = new Counter();
                         Counter  o2 = new Counter();
                         Counter  o3 = new Counter();
                         o1.disp();
                          o2.disp();
                          03.disp();
            }}
```

First Counter : 1 Second Counter : 3

First Counter : 1 Second Counter : 3

First Counter : 1 Second Counter : 3

```
class Counter
{           int c1=0;
            static int c2;
           void increment()
           {            c1++;
                        c2++;   }
           void disp()
           {             System.out.println("First Counter : "+c1+"   Second Counter : "+c2);    }
}
class prg4
{          public static void main(String args[])
           {             Counter  o1 = new Counter();
                         Counter  o2 = new Counter();
                         Counter  o3 = new Counter();
                          o1.increment();
                          o1.increment();
                          o2.increment();
                          o3.increment();
                         o1.disp();   o2.disp();   o3.disp();        }   }
```

First Counter : 2 Second Counter : 4

First Counter : 1 Second Counter : 4

First Counter : 1 Second Counter : 4

Methods declared as static have following restrictions:

- They can only directly call other static methods.
- They can only directly access static data.
- They cannot refer to this or super.

When we require to do **computation** in order to **initialize static variables,** we can declare a **static block** that gets executed **exactly once,** when the **class is first loaded.**

```
class UseStatic
{       static int a = 3;
        static int b;
        static void meth(int  x)
        {       System.out.println("x = " + x);
                System.out.println("a = " + a);
                System.out.println("b = " + b);
        }
        static {
                System.out.println("Static block initialized.");
                b = a * 4;
            }
        public static void main(String args[])
        {       meth(42);
        }
 }
```

```
class UseStatic
{       static int a = 3;
        static int b;
        static void meth(int  x)
        {       System.out.println("x = " + x);
                System.out.println("a = " + a);
                System.out.println("b = " + b);
        }
        static {
                System.out.println("Static block initialized.");
                b = a * 4;
            }
        public static void main(String args[])
        {       meth(42);
        }
}
```

```
Static block initialized.
x = 42
a = 3
b = 12
```

```
Class X
{           static int array[];
            static {
                  array = new int[6];
                  for(int i=0;i<6;i++)
                  array[i] = i;
            }
}
class prg
{             public static void main(String args[])
              {
                        for(int i=0;i<6;i++)
                        System.out.println(X.array[i]);
            }
}
```

```
Class X
{           static int array[];
            static {
                  array = new int[6];
                  for(int i=0;i<6;i++)
                  array[i] = i;
            }
}
class prg
{             public static void main(String args[])
              {
                        for(int i=0;i<6;i++)
                        System.out.println(X.array[i]);
            }
}
```

OUTPUT : 0 1 2 3 4 5

Nested Class

- Java allows to define a class within another class. Such a class is called a *nested class.*

```
class OuterClass
{
                    ...
     class NestedClass
     {
          ...
     }

}
```

Here NestedClass is called nested class.

OuterClass is called enclosing class.

Nested Class

Nested class: class within another class.

The scope of a nested class is bounded by scope of enclosing class.

If class B is defined within class A, then B does not exist independently of A.

Nested class has access to the members, including the private members, of the class in which it is nested.

Enclosing class does not have access to the members of the nested class.

Nested class types:

- **static :** It must access the members of its enclosing class through an object. ie, it cannot refer to members (non static) of its enclosing class directly. It can access static data members of outer class including private.

- **non - static: Inner class**

  It has access to all of the variables and methods of its outer class and may refer to them in the same way that other non-static members of the outer class do

Nested classes are divided into two categories: static and non-static. Nested classes that are declared static are simply called *static nested classes*. Non-static nested classes are called *inner classes*.

```
class OuterClass
{
        static class StaticNestedClass
        {
                ...
        }
        class InnerClass
        {
                ...
        }
}
```

```
class Outer
{           int  outer_x = 100;
            void test()
           {       Inner  inner = new Inner();
                   inner.display();
           }
            class Inner
          {       void display()
                  {       System.out.println("display: outer_x = " + outer_x);    }
           }
  }
class InnerClassDemo
{       public static void main(String args[])
       {      Outer  outer = new Outer();
              outer.test();
        } }
```

```
class Outer
{     int outer_x = 100;
       void test()
      {        Inner  inner = new Inner();
               inner.display();      }
       void showY()
      {          System.out.println(y);     // error, y not known here!       }
      class Inner
      {        int y = 10;  //y is local to Inner
               void display()
               {              System.out.println("display: outer_x = " + outer_x);       }
      }
}
class InnerClassDemo
{       public static void main(String args[])
       {       Outer  outer = new Outer();
               outer.test();
       }  }
```

Command-Line Arguments

Passing information into a program during the execution is carried out by passing command line arguments to main().

Command-line is the information that directly follows the program name on the command line when it is executed.

They are stored as strings in a string array passed to the args parameter of main().

```
//Display all command line arguments
class CommandLine
{
        public static void main(String args[])
        {
                for(int i=0; i< args.length; i++)
                System.out.println("args[" + i + "]: " + args[i]);
        }
}
```

Output:

```
args[0] :this
args[1]:is
args[2]:a
args[3]:test
args[4]:100
args[5]:-1
```

Program execution at command prompt:

```
Java CommandLine      this is a test 100 -1
```

final

A variable can be declared as **final**. Doing so prevents its contents from being modified.

This means that we must initialize a **final** variable when it is declared.

```
final int FILE_NEW = 1;
final int FILE_OPEN = 2;
final int FILE_SAVE = 3;
final int FILE_SAVEAS = 4;
final int FILE_QUIT = 5;
```

Access Protection

	Private	No modifier	Protected	Public
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Table 9-1. *Class Member Access*