Static block Programs :

```
//static block
class Foo
{
      Foo()
      {
            IO.println("No Argument constructor..");
      }

      {
            IO.println("Non static Block..");
      }

      static
      {
            IO.println("Static block...");
      }
}

public class StaticBlockDemo
{
      void main()
      {
            IO.println("Main Method Executed ");
      }

}
```

Here Foo.class file is not loaded into JVM Memory so static block of Foo class will not be executed.

```
class Test
{
      static int x;

      static
      {
            x = 100;
            IO.println("x value is :"+x);
      }

      static
      {
            x = 200;
            IO.println("x value is :"+x);
      }

      static
      {
            x = 300;
            IO.println("x value is :"+x);
      }
}
public class StaticBlockDemo1
{
      void main()
      {
            IO.println("Main Method");
            IO.println(Test.x); //Here Test.class will be loaded
      }
}
```

Note : If a class contains more than 1 static block then it will be executed from top to bottom.

```
class Foo
{
      static int x;

    static
      {
            IO.println("x value is :"+x);
      }
}

public class StaticBlockDemo2
{
      void main()
      {
             new Foo(); //Here Foo.class file will be loaded
    }
}
```

Note : static variables are also having default value provided by prepare phase.

```
class Demo
{
      final static int a ; //static Blank  final field

      static
      {
            m1();
            a = 100;
            IO.println("User Value :"+a);
      }

      public static void m1()
      {
            IO.println("Default Value :"+a);
      }

}
public class  StaticBlockDemo3
{
      void main()
      {
           new Demo(); //Demo.class will be loaded here
      }
}
```

A static blank final field must be initialized by static block only and It also have default value.

```
class Alpha
{
      static
      {
            IO.println("Alpha class static block");
      }
}
class Beta extends Alpha  //Inheritance [Alpha-super, Beta-Sub]
{
      static
      {
            IO.println("Beta class static block");
      }
}

public class StaticBlockDemo4
{
      void main()
      {
            new Beta(); //Try to load Beta class
      }
}
```

In class loading phase, First of all super class will be loaded then only sub class will be loaded.

```
//illegal forward reference
class Demo
{
   static
     {
           i = 100;  //Initialization is possible due to prepare phase
           IO.println(i); //Illegal forward reference
     }

   static int i;
}

public class StaticBlockDemo5
{
  void main()
     {
        new Demo();
     }
}
```

Note : In post declaration we can perform write operation but directly read operation is not possible.

```
class Demo
{
    static
      {
            i = 100;
            //IO.println(i); //Illegal forward reference
            IO.println(Demo.i);
    }

    static int i;
}

public class StaticBlockDemo6
{
    void main()
      {
            IO.println(Demo.i);
      }
}
```

```
class StaticBlockDemo7
{
      static
      {
     IO.println("Static Block");
        return;
      }

      void main()
      {
            IO.println("Main Method");
      }
}
```

```
class Test
{
      final static int A;

      public static void initialize()
      {
            A = 100;
      }
}

public class StaticBlockDemo8
{
  void main()
     {
       Test.initialize();
     }
}
```

A static blank final field must be initialized inside staic block only.

```
class Test
{
      public static final Test t1 = new Test();  //t1 = null

      static
      {
        IO.println("static block");
      }


      {
        IO.println("Non static block");
      }

      Test()
      {
           IO.println("No Argument Constructor");
      }

}

public class StaticBlockDemo9
{
      void main()
      {
           new Test();  //2 steps (class loading + Object creation)
      }
}
```

Note : static block and static field both are having same priority in the class initialization phase so executed according to the order

```
class Sample
{
      static
      {
            IO.println("Static Block");
            x = m1();
            IO.println("x from static block :"+Sample.x);
      }

      public static int m1()
      {
        IO.println("Static Method");
        return 100;
      }

      static int x;
}
public class StaticBlockDemo10
{
      void main()
      {
            IO.println("x from main method :"+Sample.x);
      }
}
```

```
class Demo
{
      public static void print()
      {
            x = 120;
            IO.println("x value is :"+x); //class name is not reqd
      }

      static int x;
}

public class StaticBlockDemo11
{
      void main()
      {
            Demo.print();
      }
}
```

What are the various members of a class :

Class Members

static member [Class Loading] Layer 1	Non static member [Object creation] Layer2
a) Static field	a) Non static Field
b) Static Block	b) Non static Block
c) Static Method	c) Constructor
d) Static Nested Class	d) Non static Method
	e) Non static inner class

Note : From layer1 (static area), If we want to access layer 2 (non static member) data then It is not possible directly we need to create an object as shown in the program

```
class Welcome
{                 //error
      int x = 200; //Layer 2

      public static void main(String [] args) //Layer 1
      {
            IO.println(x);  //error
            m1();
      }

    public void m1() //Layer 2
      {
      }

}
```

Note : That is the reason this keyword is also not allowed from static context

Note : From Layer2 , We can access Layer1 as well as Layer2 data directly as shown below

```
class Welcome
{
      static int y = 500;
      int x = 200; //Layer 2

      void main() //Layer 2
      {
            IO.println(this);
            IO.println(x);
            IO.println(y);
            m1();
      }

    public void m1() //Layer 2
      {
            IO.println("m1 non static method");
      }

}
```

Complete Java Variable Story :

Index No	Variables types	Declaration Position	Scope	Memory Area	Applicable Modifiers	Accessed by	Lifetime
01	Class Variable OR Static Field	Inside Class	Belongs to class	Method Area	private, protected, public, final, transient and volatile.	Class name OR Object reference	Entire program execution (class is unloaded)
02	Instance Variable OR Non-Static Field	Inside Class	Belongs to an object (non-static)	Heap Area	private, default, protected, public, final, transient and volatile.	Object reference	As long as object lives
03	Local Variable	Inside Method	Inside a method/block /constructor	Stack Area	Only final modifier is applicable.	Only inside method/block	Until method ends
04	Parameter Variable	Inside Method	Passed to methods/constructors	Stack Area	Only final modifier is applicable.	Passed when Calling method	Until method ends

- ➔ Invalid Modifier for **static field**: abstract, synchronized, native, strictfp and final + volatile.
- ➔ Invalid Modifier for **non-static field**: abstract, synchronized, native and strictfp.
- ➔ Invalid Modifier for **Local Variable**: Only final is allowed.
- ➔ Invalid Modifier for **Parameter Variable**: Only final is allowed.