Modifiers applicable for the constructors :

* We can declare a constructor with any access modifier like private, default, protected and public.

* Generally, We declare the constructor with private access modifier for factory method [object must be created from the same ELC class] as well as **singnlton design pattern.**

* We cannot declare a constructor with **final, static, abstract and synchronized** modifier.

What is non static block OR Instance Initializer in Java :

* A non static block is a special block which we define **inside a class** and it will be **automatically executed** whenever we **create an object** for the class.

```
public class Student
{
    private int id;
    private String name;

    {
        ------> non static block
    }
```

```
package com.ravi.non_static_block;

class Test
{
    private int j;

    {
        System.out.println("Non static block done");
    }
}
public class NonStaticBlockDemo1
{
    public static void main(String[] args)
    {
        Test t1 = new Test();
        Test t2 = new Test();
        Test t3 = new Test();
    }
}
```

* The first line of any constructor is reserved for super() or this(). If developer and not write then automatically compiler will add super() to the first line of constructor.

* If any constructor first line is containing **super()** then automatically the 2nd line of the constructor will be reserved for non static block that means automatically java compiler will place the non static block in the 2nd line of the constructor.

Case 1 :

Demo.java

```
public class Demo
{
    private int x;

    public Demo()
    {
        S.o.p("Im Arg Const");
    }

    {
        S.o.p("NSBT");
    }
}
```

javac

Demo.class

```
public class Demo
{
    private int x;

    public Demo()
    {
        super();//added by javac
        {
            S.o.p("NSBT"); //placed by javac
        }
    }
}
```

Case 2 :

Demo.java

```
public class Demo
{
    public Demo()
    {
        this(5);
        S.o.p("NoAC");
    }
    public Demo(int x)
    {
        S.o.p("PC");
    }
    {
        S.o.p("NSBT");
    }
}
```

javac

```
public class Demo
{
    public Demo()
    {
        this(5);
        S.o.p("NoAC");
    }
    public Demo(int x)
    {
        super();//added by javac
        {
            S.o.p("NSBT"); //Placed by javac
        }
    }
}
```

Note : Non static block will be automatically placed in the constructor which contains super() as a first statement.

```
package com.ravi.non_static_block;

class Demo
{
    public Demo()
    {
        System.out.println("No Argument constructor");
    }

    public Demo(int x)
    {
        System.out.println("Parameterized constructor");
    }

    {
        System.out.println("Non static block");
    }
}
public class NonStaticBlockDemo2
{
    public static void main(String[] args)
    {
        Demo d1 = new Demo();
        Demo d2 = new Demo(100);
    }
}
```

Note :three non-static block will be executed two times because two super()

```
package com.ravi.non_static_block;

class Demo
{
    public Demo()
    {
        this(10);
        System.out.println("No Argument constructor");
    }

    public Demo(int j)
    {
        System.out.println("Parameterized constructor");
    }

    {
        System.out.println("Non static block");
    }
}
public class NonStaticBlockDemo2
{
    public static void main(String[] args)
    {
        Demo d1 = new Demo();
    }
}
```

* The non static block will be executed before constructor body execution.

```
package com.ravi.non_static_block;

class Sample
{
    public Sample()
    {
        System.out.println("No Argument Constructor");
    }

    {
        System.out.println("Non static block");
    }
}
public class NonStaticBlockDemo3
{
    public static void main(String[] args)
    {
        Sample s1 = new Sample();
        System.out.println("-----------------");
        Sample s2 = new Sample();
    }
}
```

* The main purpose of non static block to initialize the non-static fields that is the reason it is also known as "Instance Initializer" as well as it is also used to provide some common **message/logic for all the objects.**

```
package com.ravi.non_static_block;

class Foo
{
    int x;

    {
        System.out.println("Object creation is in progress!!");
        x = 100;
    }
}
public class NonStaticBlockDemo4
{
    public static void main(String[] args)
    {
        Foo f1 = new Foo();
        System.out.println(f1.x);

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

        Foo f2 = new Foo();
        System.out.println(f2.x);
    }

}
```

* If a class contains multiple non static blocks then non static blocks will be executed according to the order [top to bottom].

```
package com.ravi.non_static_block;

class Instance
{
    private int i;

    public Instance()
    {
        i = 10;
        System.out.println("i value is :"+i);
    }

    {
        i = 200;
        System.out.println("i value is :"+i);
    }

    {
        i = 300;
        System.out.println("i value is :"+i);
    }

    {
        i = 400;
        System.out.println("i value is :"+i);
}
}

public class NonStaticBlockDemo4 {
    public static void main(String[] args)
    {
        Instance i = new Instance();

    }
}
```

* We cannot write any kind of return statement inside non static block because an initializer must be executed normally.

```
package com.ravi.non_static_block;

class Instance
{
    private int n;

    public Instance()
    {
        n = 100;
        System.out.println("n value is :"+n);
        return;
    }

    {
        System.out.println("NSB");
        return;
    }

}

    public class NonStaticBlockDemo5 {

        public static void main(String[] args)
        {
            Instance i = new Instance();

        }
}
```

Non static field initialization order :

All the instance variables are initialized in the following order during the life cycle

1) It will initialized with default value at the time of Object creation. Time when the object is created variable will be initialized with default value, init method is working internally.

2) Now control will verify whether, we have initialized at the time of variable declaration or not.

3) Now control will verify whether, we have initialized
   inside non static block or not. [If present]

   Note : Variable declaration can initialization and non static block, both are having same priority so, executed according to the order.

4) Now control will verify whether, we have initialized in the body of the constructor or not.

5) Now control will verify whether, we have initialized the RHS inside method body or not but it is not recommended because Object is already created, we need to call the method explicitly, It is not the part of the object creation.

So, The final conclusion is
Variable initialization <> At the time of declaration <> In the body of non static block [Priority Basis] <> In the body of constructor <> Inside method body [Not Recommended]

```
package com.ravi.oof;

class Test
{
    public Test()
    {
        x = 200;
    }

    int x =100 ;

    {
        x = 200;
    }

}
public class NonStaticFieldInitializationOrder
{
    public static void main(String[] args)
    {
        Test t1 = new Test();

        System.out.println(t1.x);

    }

}
```