Initialization order of non static variable :

The following steps describes the initialization order of non static variable :

1) Whenever we create an object in java then non static variable will be initialized with default value with the help of new keyword and Java compiler.

2) Due to super(), control will move to Object class, While coming back our user-defined class, It will verify whether the non sttaic variable is initialized at the time of DECLARATION OR NOT ?

3) Now, Control will verify whether the non static variable is initialized inside the non static block or not

Note : NON STATIC BLOCK AND INSTANCE VARIABLE DECLARATION CUM INITIALIZATION, BOTH ARE HAVING SAME PRIORITY, THE EXECUTION WILL DEPEND UPON THE ORDER.

Case 1 :

```
package com.ravi.nsv_order;

class Test
{
    private int x = 100;

    {
        x = 200;
    }

    public int getX()
    {
        return this.x;
    }
}
public class InitializationOrder
{
    public static void main(String[] args)
    {
        Test t1 = new Test();
        System.out.println(t1.getX()); //200
    }

}
```

Note : The non static variable value 100 is replaced by 200 using non static block

Case 2 :

```
package com.ravi.nsv_order;

class Test
{
    {
        x = 200;
    }

    public int getX()
    {
        return this.x;
    }

    private int x = 100;
}
public class InitializationOrder
{
    public static void main(String[] args)
    {
        Test t1 = new Test();
        System.out.println(t1.getX());

    }

}
```

Note : Non static variable 200 value is replaced by 100 using variable declaration cum initialization

4) After non static block OR variable declaration cum initialization, control will verify whether non static variable is initialized inside the constructor body or not ?

5) We can also initialized inside a method body but It is not recommended because Object creation is already completed with constructor body execution.

Summary : Default value (new keyword + Javac) => Declaration cum initialization OR non static block [Order] => Constructor body => Method body (not Recommended)

```
package com.ravi.nsv_order;

class Test
{
    private int x = 100;

    {
        x = 200;
    }

    public Test()
    {
        x = 300;
    }


    public int getX()
    {

        return this.x;
    }
}
public class InitializationOrder
{
    public static void main(String[] args)
    {
        Test t1 = new Test();
        System.out.println(t1.getX());

    }
}
```

What is blank final field ?
----

* A final variable must be initialized by developer only once.

* If a **non static final variable** is **not initialized** at the time of **declaration** then it is called blank final field.
  Example :

```
public class Student
{
    final String name; //blank final field
}
```

* A final field cannot be initialized by default constructor.

```
class Student
{
    final String name; //Blank final field          [error at compilation]
}

public class BlankFinalFieldDemo1
{
    public static void main(String[] args)
    {
        Student s = new Student();
        System.out.println(s.name);
    }
}
```

* A blank final field we cannot initialized inside the method body.

```
class Student
{
    final String name ; //Blank final field

    public void set()
    {
        name = "Raj"; //error
    }
}

public class BlankFinalFieldDemo1
{
    public static void main(String[] args)
    {
        Student s = new Student();
        s.set();
        System.out.println(s.name);
    }
}
```

* A blank final field must be initialized by the developer till the **Object creation**. [Till constructor body execution]
* A blank final field we can initialized in the following two places explicitly :
  a) Inside non static block
  OR
  b) Inside Constructor Body

```
class Student
{
    final String name ; //Blank final field

    {
        name = "A";
    }

    public Student()
    {
     //name = "B";
    }
}

public class BlankFinalFieldDemo1
{
    public static void main(String[] args)
    {
        Student s = new Student();
        System.out.println(s.name);
    }
}
```

* A blank final field also have default value as shown in the program

```
package com.ravi.blank_final_field;

class Test
{
    final int x;

    {
        print();
        x = 999;
    }

    public void print()
    {
      System.out.println("Default Value :"+x);
    }
}
public class BlankFinalField
{
    public static void main(String[] args)
    {
        Test t1 = new Test();
        System.out.println("User value :"+t1.x);
    }
}
```

* A blank final field must be initialized in all the overloaded constructors available in the class.

```
package com.ravi.blank_final_field;

class Employee
{
     final double salary;

     public Employee()
     {
        salary = 40000;
     }

     public Employee(double salary)
     {
        this.salary = salary;
     }

    @Override
    public String toString()
    {
        return "Employee [salary=" + salary + "]";
    }
}
public class BlankFinalFieldExample
{
    public static void main(String[] args)
    {
        Employee e1 = new Employee();
        System.out.println(e1);


        Employee e2 = new Employee(50000);
        System.out.println(e2);
    }
}
```

```
package com.ravi.blank_final_field;

class Customer
{
    final String name;

    public Customer()
    {
    }

    public Customer(String name)
    {
    }

    {
        name = "Raj";
    }

    @Override
    public String toString() {
        return "Customer [name=" + name + "]";
    }
}
public class BlankFinalFieldDemo
{
    public static void main(String[] args)
    {
        Customer c1 = new Customer();
        System.out.println(c1);

        Customer c2 = new Customer(null);
        System.out.println(c2);
    }

}
```