

Resolve :

All the symbolic references (#7) will be converted into direct references OR actual reference.

javap -verbose FileName.class

Note :- By using above command we can read the internal details of .class file.

Initialization :

Here class initialization will takes place. All the static data member will get their actual/original value and we can also use static block for static data member initialization.

Here, In this class initialization phase static variable and static block is having same priority so it will executed according to the order.(Top to bottom)

Static block OR Static Initializer in java :

* A static block is a special block which is automatically executed at the **time of loading the .class file** int JVM memory.

Example of static block :

```
static
{
}
```

* A static block is executed before the main method.

```
package com.ravi.static_block_demo;

public class Test
{
    public static void main(String[] args)
    {
        System.out.println("Main Method");
    }

    static
    {
        System.out.println("Static Block");
    }
}
```

* In java, A class is not loaded automatically, It will be loaded as per user request.

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

    {
        System.out.println("Non static Block..");
    }

    static
    {
        System.out.println("Static block...");
    }
}

public class StaticBlockDemo
{

    public static void main(String [] args)
    {
        System.out.println("Main Method Executed ");
    }
}
```

Note : Here Foo.class file is not loaded so static block is not executed.

* The main purpose of static block to initialize the static data member of the class so it is also known static initializer.

```
package com.ravi.static_block_demo;

class Sample
{
    static int x;

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

public class StaticBlockDemo1
{
    public static void main(String[] args)
    {
        System.out.println("Main Method");
        new Sample();
    }

    static
    {
        System.out.println("Main method SB");
    }
}
```

* If we have more than one static block then It will be executed according to the order.

```
package com.ravi.static_block_demo;

class Demo
{
    static int x;

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

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

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

public class StaticBlockDemo2 {

    public static void main(String[] args)
    {
        System.out.println("x value is :" + Demo.x);
    }
}
```

* A static variable is having default value at prepare phase.

```
package com.ravi.static_block_demo;

class Alpha
{
    static int x;

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

public class StaticBlockDemo3
{
    public static void main(String[] args)
    {
        new Alpha();
    }
}
```

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

```
package com.ravi.static_block_demo;

class Beta
{
    final static int x; //static blank final field

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

public class StaticBlockDemo4
{
    public static void main(String[] args)
    {
        new Beta();
    }
}
```

* A static blank final field also have default value.

```
package com.ravi.static_block_demo;

class A
{
    final static int A; //static blank final field

    static
    {
        print();
        A = 555;
        System.out.println("User value is :" + A);
    }

    public static void print()
    {
        System.out.println("Default value is :" + A);
    }
}

public class StaticBlockDemo5 {

    public static void main(String[] args)
    {
        new A();
    }
}
```

If we don't declare static variable before static block body execution then we can perform write operation(Initialization is possible due to prepare phase) but read operation is not possible directly otherwise we will get an error Illegal forward reference, It is possible with class name because now compiler knows that variable is coming from class area OR Method area.

Example :

```
static
{
    x = 100; //Valid
    System.out.println(x); //Invalid
    System.out.println(ClassName.x); //Valid
}
static int x;
```

/* Program

```
class Sample
{
    static
    {
        x = 100;
    }

    static int x = 200; //x = 0
}
```

public class IllegalForwardRefDemo
{
 public static void main(String[] args)
 {
 System.out.println(Sample.x);
 }
}

class Demo
{
 static
 {
 x = 200;
 System.out.println("x value is :" + x); //Illegal Forward Ref
 }

 static int x;
}

public class IllegalForwardRefDemo1
{
 public static void main(String[] args)
 {
 new Demo();
 }
}

class Sample
{
 static
 {
 x = m1();
 System.out.println(Sample.x); //class name is compulsory
 }

 public static int m1()
 {
 return 100;
 }

 static int x;
}

public class IllegalForwardRefDemo2
{
 public static void main(String[] args)
 {
 new Sample();
 }
}

* We cannot write return keyword inside static block.
* A class is loaded is only one time so static block will be executed only 1 time.

```
class Foo
{
    static int x;

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

public class StaticBlockDemo2
{
    public static void main(String[] args)
    {
        new Foo();
    }
}
```

Note : static variable are having default value

```
class A //AD BC EF
{
    static
    {
        System.out.println("A");
    }

    {
        System.out.println("B");
    }

    A()
    {
        System.out.println("C");
    }
}

class B extends A
{
    static
    {
        System.out.println("D");
    }

    {
        System.out.println("E");
    }

    B()
    {
        System.out.println("F");
    }
}

public class StaticBlockDemo4
{
    public static void main(String[] args)
    {
        new B(); //class Loading + Object Creation
    }
}
```