Blocks In Java Language

We have three types of Blocks in Java.

a. Blocks defined inside a class but not inside a method/constructor.
    1) Static Block
    2) Non Static Block

b. Blocks defined in a method
    3) Normal Block

1. What is static block?

ans: static block is a block defined in the class with static modifier.

Example:

```
class Foo{
    static{
        // static block
    }
}
```

2. what is non static block?

ans: non static block is a block defined in the class without static modifier.

Example:

```
class Foo{
    {
        //non static block
    }
}
```

Note: like we have static methods and non static methods, similary we have static blocks and non static blocks.

3. What is normal block?

ans: normal block is a block defined inside a "class method" without static modifier.

Example:

```
class Foo{
        void show(){
                {
                        // normal block
                }
         }
}
```

4. Can we define a block with static modifier inside a method?

ans: No

5. How many static blocks can we define inside a class?

ans: any number of static blocks (multiple) or as many as you need.

Example:

```
class Foo{
        static{
                //static block 1
        }
         void show(){}
        static{
                 // static block 2
        }
}
```

6. How many non static blocks can we define inside a class?

ans: any number of non static blocks (multiple) or as many as you need.

Example:

```
class Foo{
         {
                // non static block1
        }
        void show(){
                // a method
        }
        {
                // non static block2
```

```
        }
    }
```

7. What is the difference between blocks and methods?

Ans: Blocks are executed automatically because of events like (class loading and object creations)

Where as methods are not executed automatically, methods are executed by method call statements.(method invocation statements)

```
Foo f = new Foo();  // static blocks and non static blocks will be executed automatically

f.show();           // method will be executed by call.
```

Points to remember:

1. we can define blocks in a method
2. Blocks defined inside a method are different from blocks defined inside a class.
3. Blocks defined inside a class are executed automatically whenever we are loading the class or whenever we are creating the object.
4. Blocks defined inside a method are executed along with method execution.
5. Blocks defined inside a method are not static or not non-static.
6. Blocks defined inside a method are normal blocks.

Blocks vs methods:

---

1. There is No name to the Blocks => 1. A methods has name
2. A block can not receive parameters => 2. A method can receive parameters
3. A block can not return any value => 3. A method can return a value.
4. A block is executed automatically => 4. A method is not executed automatically(we have to call it)
5. Static block is executed automatically whenever the class is loaded
6. Non static block is executed automatically whenever object is under creation.
7. A normal block is executed automatically with the method execution => 5. A method is called/invoked by the programmer with method call statement

8. When does JVM execute static blocks automatically?

ans: When ever JVM loads the class form HD into meta space(method area) then static blocks of the same class will be executed automatically.

9. When does JVM execute non static blocks automatically?

ans: When ever JVM creates the object of the class then JVM will execute automatically non static blocks of the same class.

i.e., if we are loading the class then static blocks of the class will be executed automatically
if we are instantiating the class (creating the object of the class) then non static blocks will be executed automatically.(remember i.e., before constructor execution)

10. If we have defined three static blocks in our class then how JVM will execute those static blocks?

ans: In which order they are defined in the class in that order.

11. If we have defined three non-static blocks in our class then how JVM will execute those non static blocks?

ans: In which order they are defined in the class in that order.

11. How many ways to load a class?

1st way

By supplying the class name to JVM by using java command at command prompt or terminal

"java ClassName"
whenever we issue the below command what happens inside JVM explained in detailed
java Sample
---

1. JVM loads Sample.class file from HD into RAM (JVM metaspace / JVM method area)
2. verifies Sample.class file byte code with current JVM version
   if verification failed then then stop the process otherwise go to step3
3. Create a String array to hold command line args
   if there are no command line arguments then create String array with zero elements.
   String cmdargs[] = new String[0];
4. Create Static data members of Sample class in metaspace(i.e. inside class memory)
   execute static data members initialization and static blocks execution in which order they are defined in sample class.
5. call the main method of Sample class with string array created in step
   Sample.main(cmdargs);

Sample.java

```
----------
        public class Sample{
                static{
                        System.out.println("Sample SB1");
                }
                public static void main(String[] args){
                        System.out.println("Sample main");
                }
                static{
                         System.out.println("Sample SB2");
                }
        }
```

```
javac Sample.java
java Sample
```

Modified Sample.java

```
----------------------
        public class Sample{
                static{
                         System.out.println("Sample SB1 a="+a);
                }
                 static int a;
                public static void main(String[] args){
                         System.out.println("Sample main a="+a);
                }
                static{
                        System.out.println("Sample SB2 a="+a);
                }
        }
```

```
javac Sample.java
Sample.java:3: error: illegal forward reference
    System.out.println("Sample SB1 a="+a);
                               ^
1 error
```

Modified Sample.java

```
--------------------
        public class Sample{
                static{
                         System.out.println("Sample SB1 a="+getA()); // here we must use get A
                                                                        otherwise c error
                 }
```

```
        static int a;
        public static void main(String[] args){
                System.out.println("Sample main a="+a);
        }
        static{
                System.out.println("Sample SB2 a="+getA()); // here we can use a, no
error
        }
        static int getA(){
                return a;
        }
}
```

```
C:\creator>java Sample
Sample SB1 a=0
Sample SB2 a=5
Sample main a=5
```

Modified Sample.java

```
public class Sample{
        static{
                System.out.println("Sample SB1 a="+getA()); // here we must use get A
                                                                otherwise c error
        }
        static int a=5;
        public static void main(String[] args){
                System.out.println("Sample main a="+a);
        }
        static{
                 System.out.println("Sample SB2 a="+getA()); // here we can use a no
                                                                error
        }
        static int getA(){
                return a;
        }
}
```

```
C:\creator>java Sample
Sample SB1 a=0
Sample SB2 a=5
Sample main a=5
```

Modified Sample.java

```
public class Sample{
        static{
                System.out.println("Sample SB1 a="+getA()); // here we must use get A
                                                                otherwise c error
        }
        static final int a=5;
        public static void main(String[] args){
                System.out.println("Sample main a="+a);
        }
        static{
                 System.out.println("Sample SB2 a="+getA()); // here we can use a no
                                                                error
        }
        static int getA(){
                return a;
        }
}
```

```
C:\creator>java Sample
Sample SB1 a=5
Sample SB2 a=5
Sample main a=5
```

2nd way:

Create object of the class for the first time

If our class has not loaded yet but we are creating object of the class for the first time then the JVM will load the class first and then create the object.

note: 2nd time, 3rd time object creation statements will never load the class again (becuase class is already loaded with first object creation)
a class will be loaded only once.

Example:

```
class Test{
        static{
                System.out.println("SB of test");
         }
}
```

```
public class Sample{
	public static void main(String[] args){
		 System.out.println("main begin");
		Test t1, t2, t3;
		System.out.println("-------");
		t1 = new Test();
		System.out.println("-------");
		t2 = new Test();
		System.out.println("-------");
		t3 = new Test();
		System.out.println("main end");
	}
}
```

```
C:\creator>javac Sample.java

C:\creator>java Sample
main begin
-------
SB of test
-------
-------
main end
```

3rd way is

Assume that our class is not loaded and if we are trying to access static members of the class for the first time then class will be loaded
remember: if we are trying to access static member of the class for 2nd time and3rd time..... then class will not be loaded again.(because class is already loaded)
a class will be loaded only once.

Example:

```
class Test{
	static int a = 5;
	static{
		System.out.println("SB of test");
	}
}

public class Sample{
	public static void main(String[] args){
		System.out.println("main begin");
		Test.a = 6;
```

```
                System.out.println("-------");
                Test.a = 7;
                System.out.println("-------");
                Test.a =8;
                System.out.println("main end");
        }
}
```

C:\creator>notepad Sample.java

C:\creator>javac Sample.java

C:\creator>java Sample

main begin
SB of test

----
main end

4th way is:
---
if we use the Class.forName("your-class-name"); for the first time then also. class file will be loaded.
and static block will be executed

2nd time use of Class.forName("your-class-name"); will not load the class again.(class is loaded only once)

if class is loaded by any one of the above ways for the first time then class will not be loaded again by any one of the above ways.
means(because class is loaded only once)

if class is loaded by 3rd way
you can not load the class again by using 1st,2nd,3rd and 4th ways.(because class is loaded only once)

Example:

```
class Test{
        static{
                System.out.println("SB of test");
        }
}
```

```
public class Sample{
	public static void main(String[] args)throws Exception{
		System.out.println("main begin");
		Class.forName("Test");
		System.out.println("------");
		Class.forName("Test");
		System.out.println("main end");
	}
}
```

```
C:\creator>java Sample

main begin
SB of test
------
main end
```

Note: we understand that SB is executed only once why because class is loaded only once.

12. Can we initialize static data member in the static block, where data member is declared above the block?

ans: yes

Example:

```
class Sample{
	static int a;
	static{
		a=5;
	}
}

public class BlocksDemo6{
	public static void main(String[] args){
		System.out.println(Sample.a);
	}
}
```

13. Can we initialize static data member in a static block where static data member declaration line is below the static block?

ans: yes

Example:

```
class Sample{
        static{
                a=5;
        }
        static int a;
}
public class BlocksDemo6{
        public static void main(String[] args){
                System.out.println(Sample.a);
         }
}
```

14. Can we access static data member in the above static block and in the below static block of data member declaration line?

ans: we can access only in the below static block
but not in the above static block.(it gives illegal forward reference error)

Example:

```
class Sample{
        static{
                System.out.println("In SB1 a="+a);// c error
        }
        static int a;
        static{
                System.out.println("In SB1 a="+a);//no c error
        }
}

public class BlocksDemo6c{

        public static void main(String[] args){
                System.out.println(Sample.a);
        }
}
```

15. Can we call static method of the same class in the static block?

Ans: Yes

Example:

```
class Sample{
	static{
		System.out.println("SB begin");
		fun();
		System.out.println("SB end");
	}
	static void fun(){
		System.out.println("inside fun");
	}
}

public class Demo{
	 public static void main(String[] args)throws Exception{
		Class.forName("Sample");
	}
}
```

Note: to overcome illegal forward reference error we have to use static getter method to access static data

Example:

```
class Sample{
	static{
		System.out.println("In SB1 a="+getA());
	}
	static int a;
	static{
		System.out.println("In SB1 a="+a);
	}
	static int getA(){
		return a;
	}
}

public class BlocksDemo6c{
	public static void main(String[] args){
		System.out.println(Sample.a);
	}
}
```

Note: Static final data Initialization is allowed in 1st static block or in 2nd block or in declaration line. But not in all three areas.
We can not initialize final static data member in a static method.

Ex1: allowed

```
class Sample{
        static{
        }
        static final int a = 5;          //initialization in declaration line
        static{
        }
}
```

Ex2: allowed

```
class Sample{
        static{
                a = 5;          // initialization in 1st static block
        }
        static final int a;
        static{
        }
}
```

Ex3: allowed

```
class Sample{
        static{
        }
        static final int a;
        static{
                a = 5;          // initialization in 2nd static block
        }
}
```

Ex4: not allowed

```
class Sample{
        static final int a;
        static void set(){
                a = 5;          //not allowed
        }
}
```

Note: Initialization of static final data member can be done only in its declaration line or in its static blocks. But not outside.

Note: without initializing static final data member access is not allowed.

16. When does JVM execute static block?

ans: We know that static block is executed after loading the class.

17. Why static block?

ans: To perform any task at class load time we use static block.

Examples:

1. Initialization of static data members.
2. If you want to open any file at class load time you can write that file opening logic inside static block of the class.
3. If you want to oepn a server connection/socket connection/db connection at class load time then you can write that logic inside static block of the class.
4. If you want to register some methods or some thing in class memory then you can write that logic inside static block of the class.

18. When does JVM execute non-static block?

Ans: some people will say during object creation non static blocks are executed.
some people will say after object creation non static blocks are executed. (i like this)
after object creation meaning is after memory allocation done for object in heap area.

Note: for every object creation of the class Non static blocks of the class are executed.

Example:

```
class Alpha{
	{
		System.out.println("NSB of Alpha");
	}
}

public class NSBTestCase1{
	public static void main(String[] args){
		System.out.println("main begin");
		Alpha a1,a2;
		System.out.println("------");
```

```
                a1 = new Alpha();
                System.out.println("------");
                a2 = new Alpha();
                System.out.println("main end");
        }
}
```

19. Can we initialize non static data members by using non static block?

Ans: yes

20. Does non static block contain this reference?

Ans: yes

21. Can we initialized non-static data member in the below non-static block?

Ans: yes

Example:

```
class Sample{
        int a;
        {
                a=5;
        }
}

public class BlocksDemo12{
        public static void main(String[] args){
        System.out.println(new Sample().a);//5
        }
}
```

22. Can we initialize non-static data member in the above non-static block of non-static data member declaration line?

Ans: yes

Example:

```
class Sample{
        {
                a=5;
         }
```

```
        int a;
}

public class BlocksDemo13{
        public static void main(String[] args){
                Sample s = new Sample();
                System.out.println(s.a);
        }
}
```

23. Can we access non-static data member in the above non-static block and below non-static block of data member declaration line?

Ans: we can access in the below non-static block but we can not in the above non-static block.

Example:

```
class Sample{
        {
                a = 5;          //allowed
                System.out.println("In NSB1 a="+a);     //direct access is not allowed
                System.out.println("In NSB1 a="+getA());       //indirect access is allowed
        }

        int a;
        {
                System.out.println("In NSB2 a="+a);//no c error
        }

         int getA(){
                return a;
        }
}

public class BlocksDemo14{
        public static void main(String[] args){
                Sample s = new Sample();
                System.out.println(s.a);
        }
}
```

24. If we have defined static block and non-static blocks in a class then what will be the execution order?

Ans: if load the class then static blocks will be executed.

Without loading the class if we create object of the class for the first time then Static block, non-static block are executed.

If we create object of the class 2nd time, 3rd time, â€¦ then static block will not be executed but only non-static block will be executed.

Every time After execution of non-static block, the specified constructor will be executed.

So we came to know that non static block is executed after every object creation.

And we came to know that after non static block execution constructors will be executed.

Example:

```
class Alpha{
	static{
		System.out.println("Alpha SB");
	}
	{
		System.out.println("Alpha NSB");
	}
	Alpha(){
		System.out.println("Alpha con");
	}
}

public class BlocksDemo15{
	 public static void main(String[] args)throws ClassNotFoundException{
		System.out.println("main begin");
		Class.forName("Alpha");
		System.out.println("----");
		Alpha r1 = new Alpha();
		System.out.println("----");
		Alpha r2 = new Alpha();
		System.out.println("main end");
	}
}
```

output

```
------
main begin
Alpha SB
----
Alpha NSB
Alpha con
```

Alpha NSB
Alpha con
main end

other example:

```
class Sample {
	{
		System.out.println("in NSB1 x=" + getX());
	}
	int x = 5;	// data member created with 0 and then initialized to 5
	{
		System.out.println("in NSB2 x=" + getX());
	}
	int getX() {
		return x;
	}
}

public class BlocksDemo9{
	public static void main(String[] args) {
		Sample s = new Sample();
	}
}
```

output:

in NSB1 x=0
in NSB2 x=5

BlocksDemo10.java

```
class Sample {
	{
		System.out.println("in NSB1 x=" + getX());
	}
	final int x = 5;	// data member created with 5
	{
		System.out.println("in NSB2 x=" + getX());
	}
	int getX() {
		return x;
	}
	}
```

```
public class BlocksDemo10{
	public static void main(String[] args) {
		Sample s = new Sample();
	}
}
```

output

```
------
in NSB1 x=5
```