**Static Blocks in Java**

In java you see "static variables", "static methods", "static classes" and "static blocks". Static variables, static methods and static classes are known to everyone but what is this "static block". Lets see what, where and how these static blocks are used.

But before going into "static block", lets refresh what other static stuff are. Now "static variables" are class variables i.e., there will be only one copy for each class and not one copy for each object of the class and these variables will be accessed without instantiating the class. Then what are static methods. Again they are class methods i.e., they can be accessed without creating an instance of the class and like static variables, static methods will be accessed without instantiating the class. Note that static methods cannot access instance variables. They can access only static variables. Next what are static classes. You cannot declare a top-level class as a static class. Java will throw a compilation error. Only inner classes that are member classes can be declared as static. If we declare member classes as static, we can use it as a top-level class outside the context of top-level class. One catch here is "The static keyword does not do to a class declaration what it does to a variable or a method declaration." - what it means is say for example you have a static variable, then to access that static variable you will use the notation   
*<<Class Name>>.<<Variable Name>>*   
but when you want to use the static inner class, you need to instantiate like   
*<<Top-level class name>>.<<Inner static class name>> newClass = new <<Top-level class name>>.<<Inner static class name>>();*

*Static blocks* are also called *Static initialization blocks* . A static initialization block is a normal block of code enclosed in braces, { }, and preceded by the static keyword. Here is an example:

static {

// whatever code is needed for initialization goes here

}

A class can have any number of static initialization blocks, and they can appear anywhere in the class body. The runtime system guarantees that static initialization blocks are called in the order that they appear in the source code. And dont forget, this code will be executed when JVM loads the class. JVM combines all these blocks into one single static block and then executes. Here are a couple of points I like to mention:

* If you have executable statements in the static block, JVM will automatically execute these statements when the class is loaded into JVM.
* If you’re referring some static variables/methods from the static blocks, these statements will be executed after the class is loaded into JVM same as above i.e., now the static variables/methods referred and the static block both will be executed.

Lets see an example:

public class StaticExample{

static {

System.out.println("This is first static block");

}

public StaticExample(){

System.out.println("This is constructor");

}

public static String staticString = "Static Variable";

static {

System.out.println("This is second static block and "

+ staticString);

}

public static void main(String[] args){

StaticExample statEx = new StaticExample();

StaticExample.staticMethod2();

}

static {

staticMethod();

System.out.println("This is third static block");

}

public static void staticMethod() {

System.out.println("This is static method");

}

public static void staticMethod2() {

System.out.println("This is static method2");

}

}

What will happen when you execute the above code? You will see below output.

This is first static block

This is second static block and Static Variable

This is static method

This is third static block

This is constructor

This is static method2

Now lets the output. First all static blocks are positioned in the code and they are executed when the class is loaded into JVM. Since the static method staticMethod() is called inside third static block, its executed before calling the main method. But the staticMethod2() static method is executed after the class is instantiated because it is being called after the instantiation of the class.

Again if you miss to precede the block with "static" keyword, the block is called "constructor block" and will be executed when the class is instantiated. The constructor block will be copied into each constructor of the class. Say for example you have four parameterized constructors, then four copies of contructor blocks will be placed inside the constructor, one for each. Lets execute the below example and see the output.

public class ConstructorBlockExample{

{

System.out.println("This is first constructor block");

}

public ConstructorBlockExample(){

System.out.println("This is no parameter constructor");

}

public ConstructorBlockExample(String param1){

System.out.println("This is single parameter constructor");

}

public ConstructorBlockExample(String param1, String param2){

System.out.println("This is two parameters constructor");

}

{

System.out.println("This is second constructor block");

}

public static void main(String[] args){

ConstructorBlockExample constrBlockEx =

new ConstructorBlockExample();

ConstructorBlockExample constrBlockEx1 =

new ConstructorBlockExample("param1");

ConstructorBlockExample constrBlockEx2 =

new ConstructorBlockExample("param1", "param2");

}

}

The output is.

This is first constructor block

This is second constructor block

This is no parameter constructor

This is first constructor block

This is second constructor block

This is single parameter constructor

This is first constructor block

This is second constructor block

This is two parameters constructor

The above example is self-explanatory.

Now lets go back to static blocks.

There is an alternative to static blocks —you can write a private static method.

class PrivateStaticMethodExample {

public static varType myVar = initializeClassVariable();

private static varType initializeClassVariable() {

//initialization code goes here

}

}

The advantage of private static methods is that they can be reused later if you need to reinitialize the class variable. So, you kind of get more flexibility with a private static method in comparison to the corresponding static initialization block. This should not mislead that a 'public' static method can't do the same. But, we are talking about a way of initializing a class variable and there is hardly any reason to make such a method 'public'.

*So what are the advantages of static blocks?*

* If you’re loading drivers and other items into the namespace. For ex, Class class has a static block where it registers the natives.
* If you need to do computation in order to initialize your static variables,you can declare a static block which gets executed exactly once,when the class is first loaded.
* Security related issues or logging related tasks

*Ofcourse there are limitations for static blocks*

* There is a limitation of JVM that a static initializer block should not exceed 64K.
* You cannot throw Checked Exceptions.
* You cannot use this keyword since there is no instance.
* You shouldn’t try to access super since there is no such a thing for static blocks.
* You should not return anything from this block.
* Static blocks make testing a nightmare.

*Finally how to handle Exceptions in static blocks?*   
In methods, an exception can be handled by either passing through the Exception or handling it. But in a static block code, you cannot handle exceptions this way.

Generally a clean way to handle it is using a try-catch block but here since we dont have this option lets look at the available three options.

First: After logging the exception throw a RuntimeException which will end the current thread (unless caught by code instantiating / calling a static method on the class for the first time).

Second is calling System.exit(1) but this is not desirable in a managed environment like a servlet. This option is only for java applications and only if the static initializer block performs some critical (without which the program cannot be run successfully) function like loading the database driver.

Third and final option is to set a flag indicating failure. Later the constructors can check the flag and throw exceptions or retry in rare cases.

Finally, if the operation is not important to the functioning of the program then maybe a simple log entry is all that is required