**package** mypack;

**import** java.lang.reflect.Method;

**class** SomeClass

{

**static**

{

System.***out***.println("in SomeClass static block");

}

}

**public** **class** Main

{

**public** **static** **void** main(String[] args)

{

Class c=SomeClass.**class**;

System.***out***.println("after creating Class reference");

Method arr[]=c.getDeclaredMethods();

}

}

Here SomeClass.class returns a reference to the Class object associated with SomeClass. But there’s a bit of nuance to how Java class loading works, and I'll clarify that for you.

**Class Loading vs. Class Metadata (Class Reference)**

When you use SomeClass.class, what you’re doing is accessing the **class literal** for SomeClass. This literal refers to the Class object associated with SomeClass—which is an instance of the Class class. This Class object holds metadata about the class, such as its name, methods, fields, and more. However, **just obtaining the Class object does not trigger class loading** unless you do something that explicitly requires the class to be initialized.

Here’s the key point: **The Class object itself is created only when the class it represents is loaded**. But simply referencing SomeClass.class doesn’t mean the class SomeClass is loaded or initialized. It's just referencing the Class object. Java’s class loading mechanism kicks in only when certain actions are taken to **actually load and initialize the class**.

**Why is the static block not called?**

The static block in SomeClass is executed when the class is initialized, but **just obtaining the Class object (using SomeClass.class) does not cause the class to be initialized**. The static block is executed during class initialization, but obtaining the Class reference doesn’t necessarily do that.

Class initialization (including static block execution) happens in these scenarios:

* **Creating an instance of the class:** new SomeClass().
* **Accessing a static field or method of the class.**
* **Using reflection to call a constructor or static method.**

**The Role of Class Object:**

In your code, SomeClass.class gives you a reference to the Class object for SomeClass, but it doesn’t trigger class loading and initialization. The Class object will only get instantiated when SomeClass is loaded into memory, and this typically happens when the class is used in some way, like by creating an instance or calling a method/field.

**When Does the Class Object Get Created?**

When SomeClass.class is accessed, the Class object is created for SomeClass (if it hasn’t been created yet). However, **this does not cause the SomeClass class to be loaded and initialized**—it's only when the class is actually used that it gets loaded.

**How Can We Force Class Initialization?**

Here’s how you can force class initialization and cause the static block to execute:

1. **Create an instance of the class:**
2. public static void main(String[] args) {
3. Class c = SomeClass.class;
4. System.out.println("after creating Class reference");
6. // Force class initialization by creating an instance
7. SomeClass obj = new SomeClass();
9. Method arr[] = c.getDeclaredMethods();
10. }
11. **Access a static field or method:**
12. public static void main(String[] args) {
13. Class c = SomeClass.class;
14. System.out.println("after creating Class reference");
16. // Force class initialization by accessing a static field or method
17. SomeClass.someStaticMethod(); // Or any static method/field
18. Method arr[] = c.getDeclaredMethods();
19. }
20. **Using reflection to invoke a constructor:**
21. public static void main(String[] args) {
22. Class c = SomeClass.class;
23. System.out.println("after creating Class reference");
25. // Force class initialization using reflection
26. try {
27. SomeClass obj = (SomeClass) c.getDeclaredConstructor().newInstance();
28. } catch (Exception e) {
29. e.printStackTrace();
30. }
31. Method arr[] = c.getDeclaredMethods();
32. }

In all of these cases, class initialization is **triggered**, and the static block is executed.

**Recap and Why This Happens:**

* SomeClass.class simply gives you a reference to the Class object for SomeClass. The Class object is created when SomeClass is loaded, but this does **not** cause the class to be initialized or run its static block.
* **Class initialization** happens only when the class is actively used, such as by creating an instance, accessing a static member, or invoking a method via reflection.
* The static block in SomeClass is executed when SomeClass is loaded and initialized into memory, but **just getting a reference to the Class object (SomeClass.class) does not trigger the loading**.

This is the crux of why the static block isn't executed in your original code.