JVM Shutdown Hook in Java

Shutdown Hooks are a special construct that allow developers to plug in a piece of code to be executed when the JVM is shutting down. This comes in handy in cases where we need to do special clean up operations in case the VM is shutting down.

Handling this using the general constructs such as making sure that we call a special procedure before the application exists (calling System.exit(0)) will not work for situations where the VM is shutting down due to an external reason (ex. kill request from O/S), or due to a resource problem (out of memory). As we will see soon, shutdown hooks solve this problem easily, by allowing us to provide an arbitrary code block, which will be called by the JVM when it is shutting down.

From the surface, using a shutdown hook is downright straight forward. All we have to do is simply write a class which extends the java.lang.Thread class, and provide the logic that we want to perform when the VM is shutting down, inside the public void run() method. Then we register an instance of this class as a shutdown hook to the VM by calling Runtime.getRuntime().addShutdownHook(Thread) method. If you need to remove a previously registered shutdown hook, the Runtime class provides the removeShutdownHook(Thread) method as well.

For Example :

```
public class ShutDownHook
{
  public static void main(String[] args)
  {

    Runtime.getRuntime().addShutdownHook(new Thread()
    {
      public void run()
      {
        System.out.println("Shutdown Hook is running !");
      }
    });
    System.out.println("Application Terminating ...");
  }
}
```

Run on IDE

When we run the above code, you will see that the shutdown hook is getting called by the JVM when it finishes execution of the main method.

Output:

```
Application Terminating ...
Shutdown Hook is running !
```

Simple right? Yes it is.

While it is pretty simple to write a shutdown hook, one needs to know the internals behind the shutdown hooks to make use of those properly. Therefore, in this article, we will be exploring some of the 'gotchas' behind the shutdown hook design.

1. Shutdown Hooks may not be executed in some cases!

First thing to keep in mind is that it is not guaranteed that shutdown hooks will always run. If the JVM crashes due to some internal error, then it might crash down without having a chance to execute a single instruction. Also, if the O/S gives a SIGKILL (http://en.wikipedia.org/wiki/SIGKILL) signal (kill -9 in Unix/Linux) or TerminateProcess (Windows), then the application is required to terminate immediately without doing even waiting for any cleanup activities. In addition to the above, it is also possible to terminate the JVM without allowing the shutdown hooks to run by calling Runime.halt() method.

Shutdown hooks are called when the application terminates normally (when all threads finish, or when System.exit(0) is called). Also, when the JVM is shutting down due to external causes such as user requesting a termination (Ctrl+C), a SIGTERM being issued by O/S (normal kill command, without -9), or when the operating system is shutting down.

2. Once started, Shutdown Hooks can be forcibly stopped before completion.

This is actually a special case of the case explained before. Although the hook starts execution, it is possible to be terminated before it completes, in cases such as operating system shutdowns. In this type of cases, the O/S waits for a process to terminate for a specified amount of time once the SIGTERM is given. If the process does not terminate within this time limit, then the O/S terminates the process forcibly by issuing a SIGTERM (or the counterparts in Windows). So it is possible that this happens when the shutdown hook is half-way through its execution.

Therefore, it is advised to make sure that the Shutdown Hooks are written cautiously, ensuring that they finish quickly, and do not cause situations such as deadlocks. Also, the JavaDoc [1] specifically mentions that one should not perform long calculations or wait for User I/O operations in a shutdown hook.

3. We can have more than one Shutdown Hooks, but their execution order is not guaranteed.

As you might have correctly guessed by the method name of addShutdownHook method (instead of setShutdownHook), you can register more than one shutdown hook. But the execution order of these multiple hooks is not guaranteed by the JVM. The JVM can execute shutdown hooks in any arbitrary order. Moreover, the JVM might execute all these hooks concurrently.

4. We cannot register / unregister Shutdown Hooks with in Shutdown Hooks

Once the shutdown sequence is initiated by the JVM, it is not allowed to add more or remove any existing shutdown hooks. If this is attempted, the JVM throws IllegalStateException.

5. Once shutdown sequence starts, it can be stopped by Runtime.halt() only.

Once the shutdown sequence starts, only Runtime.halt() (which forcefully terminates the JVM) can stop the execution of the shutdown sequence (except for external influences such as SIGKILL). This means that calling System.exit() with in a Shutdown Hook will not work. Actually, if you call System.exit() with in a Shutdown Hook, the VM may get stuck, and we may have to terminate the process forcefully.

6. Using shutdown hooks require security permissions.

If we are using Java Security Managers, then the code which performs adding / removing of shutdown hooks need to have the shutdownHooks permission at runtime. If we invoke this method without the permission in a secure environment, then it will result in SecurityException.