(/)

Runtime.getRuntime().halt() vs System.exit() in Java

Last updated: November 17, 2022

> Written by: Kamlesh Kumar (https://www.baeldung.com/author/kamlesh-kumar)

Java (https://www.baeldung.com/category/java) +

JVM (https://www.baeldung.com/tag/jvm)

Get started with Spring 5 and Spring Boot 2, through the *Learn Spring* course:

> CHECK OUT THE COURSE (/ls-course-start)

1. Overview

In this tutorial, we'll look into *System.exit()* (/java-system-exit), *Runtime.getRuntime().halt()*, and how these two methods compare with each other.

2. *System.exit()*

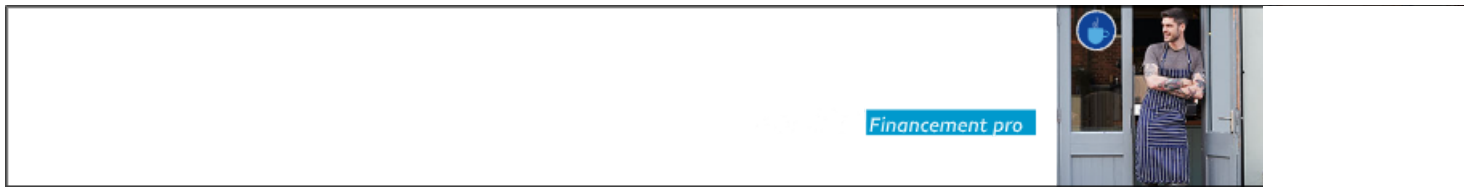
The *System.exit()* method **stops the running Java Virtual Machine**. But, before stopping the JVM, it **calls the shutdown sequence**, also known as an orderly shutdown. Please refer to this article (/adding-shutdown-hooks-for-jvm-applications) to learn more about adding shutdown hooks.

The shutdown sequence of JVM first invokes all registered shutdown hooks and waits for them to complete. Then, it runs all uninvoked finalizers if *finalization-on-exit* is enabled. Finally, it halts the JVM.

This method, in fact, calls the *Runtime.getRuntime().exit()* method internally. It takes an integer status code as an argument and has a *void* return type:

```
public static void exit(int status)
```

If the status code is nonzero, it indicates that the program stopped abnormally.



(https://freestar.com/?
ı_campaign=branding&utm_medium=banner&utm_source=baeldung.com&utm_content=baeldung_leaderboarc
_1)

3. *Runtime.getRuntime().halt()*

The *Runtime* class allows an application to interact with the environment in which the application is running.

It has a *halt* method that can be used to **forcibly terminate the running JVM**.

Unlike the *exit* method, this method does not trigger the JVM shutdown sequence. Therefore, **neither the shutdown hooks or the finalizers are executed** when we call the *halt* method.

This method is non-static and has a similar signature to *System.exit()*:

```
public void halt(int status)
```

Similar to *exit*, the non-zero status code in this method also indicates abnormal termination of the program.

(https://freestar.com/?_campaign=branding&utm_medium=banner&utm_source=baeldung.com&utm_content=baeldung_leaderboard2)

4. Example

Now, let's see an example of *exit* and *halt* methods, with the help of a shutdown hook.

To keep it simple, we'll create a Java class and register a shutdown hook in a *static* block. Also, we'll create two methods; the first calls the *exit* method and the second calls the *halt* method:

```
public class JvmExitAndHaltDemo {

    private static Logger LOGGER = LoggerFactory.getLogger(JvmExitAndHaltDemo.class);

    static {
        Runtime.getRuntime()
          .addShutdownHook(new Thread(() -> {
            LOGGER.info("Shutdown hook initiated.");
          }));
    }

    public void processAndExit() {
        process();
        LOGGER.info("Calling System.exit().");
        System.exit(0);
    }

    public void processAndHalt() {
        process();
        LOGGER.info("Calling Runtime.getRuntime().halt().");
        Runtime.getRuntime().halt(0);
    }

    private void process() {
        LOGGER.info("Process started.");
    }

}
```

So, to test the exit method first, let's create a test case:

```
@Test
public void givenProcessComplete_whenExitCalled_thenTriggerShutdownHook() {
    jvmExitAndHaltDemo.processAndExit();
}
```

Let's now run the test case and see that the shutdown hook is called:

```
12:48:43.156 [main] INFO com.baeldung.exitvshalt.JvmExitAndHaltDemo - Process started.
12:48:43.159 [main] INFO com.baeldung.exitvshalt.JvmExitAndHaltDemo - Calling System.exit().
12:48:43.160 [Thread-0] INFO com.baeldung.exitvshalt.JvmExitAndHaltDemo - Shutdown hook initiated.
```

Similarly, we'll create a test case for the *halt* method:

(https://freestar.com/?_campaign=branding&utm_medium=banner&utm_source=baeldung.com&utm_content=baeldung_leaderboard3)

```
@Test
public void givenProcessComplete_whenHaltCalled_thenDoNotTriggerShutdownHook() {
    jvmExitAndHaltDemo.processAndHalt();
}
```

Now, we can run this test case also and see that the shutdown hook is not called:

```
12:49:16.839 [main] INFO com.baeldung.exitvshalt.JvmExitAndHaltDemo - Process started.
12:49:16.842 [main] INFO com.baeldung.exitvshalt.JvmExitAndHaltDemo - Calling
Runtime.getRuntime().halt().
```

5. When to Use *exit* and *halt*

As we've seen earlier, the *System.exit()* method triggers the shutdown sequence of JVM, whereas the *Runtime.getRuntime().halt()* terminates the JVM abruptly.

We can also do this by using operating system commands. For example, we can use SIGINT or Ctrl+C to trigger the orderly shutdown like *System.exit()* and SIGKILL to kill the JVM process abruptly.

Therefore, we rarely need to use these methods. Having said that, we may need to use the *exit* method when we need the JVM to run the registered shutdown hooks or return a specific status code to the caller, like with a shell script.

However, it is important to note that the shutdown hook may cause a deadlock, if not designed properly. Consequently, the ***exit* method can get blocked** as it waits until the registered shutdown hooks finish. So, a possible way to take care of this is to use the *halt* method to force JVM to halt, in case *exit* blocks.

Finally, an application can also restrict these methods from accidental use. Both these methods call the *checkExit* method of the *SecurityManager* (/java-security-manager) class. So, to **disallow the *exit* and *halt* operations**, an application can create a security policy using the *SecurityManager* class and throw the *SecurityException* from the *checkExit* method.

6. Conclusion

In this tutorial, we've looked into the *System.exit()* and *Runtime.getRuntime().halt()* methods with the help of an example. Moreover, we've also talked about the usage and best practices of these methods.

As usual, the complete source code of this article is available over on Github (https://github.com/eugenp/tutorials/tree/master/core-java-modules/core-java-jvm).

Get started with Spring 5 and Spring Boot 2, through the *Learn Spring* course:

>> CHECK OUT THE COURSE (/ls-course-end)

Learning to build your API
with Spring?

Download the E-book (/rest-api-spring-guide)

Comments are closed on this article!

COURSES

ALL COURSES (/ALL-COURSES)
ALL BULK COURSES (/ALL-BULK-COURSES)
ALL BULK TEAM COURSES (/ALL-BULK-TEAM-COURSES)
THE COURSES PLATFORM (HTTPS://COURSES.BAELDUNG.COM)

SERIES

JAVA “BACK TO BASICS” TUTORIAL (/JAVA-TUTORIAL)
JACKSON JSON TUTORIAL (/JACKSON)
APACHE HTTPCLIENT TUTORIAL (/HTTPCLIENT-GUIDE)
REST WITH SPRING TUTORIAL (/REST-WITH-SPRING-SERIES)
SPRING PERSISTENCE TUTORIAL (/PERSISTENCE-WITH-SPRING-SERIES)
SECURITY WITH SPRING (/SECURITY-SPRING)
SPRING REACTIVE TUTORIALS (/SPRING-REACTIVE-GUIDE)

ABOUT

ABOUT BAELDUNG (/ABOUT)
THE FULL ARCHIVE (/FULL_ARCHIVE)
EDITORS (/EDITORS)
JOBS (/TAG/ACTIVE-JOB/)
OUR PARTNERS (/PARTNERS)

PARTNER WITH BAELDUNG (/ADVERTISE)

TERMS OF SERVICE (/TERMS-OF-SERVICE)

PRIVACY POLICY (/PRIVACY-POLICY)

COMPANY INFO (/BAELDUNG-COMPANY-INFO)

CONTACT (/CONTACT)