Semaphore in Java

Semaphore

Java's semaphore can be `releas()-ed` or `acquire()-d` for signalling amongst threads. However the important call out when using semaphores is to make sure that the permits acquired should equal permits returned. Take a look at the following example, where a runtime exception causes a deadlock.

```
1  import java.util.concurrent.Se
2
3  class Demonstration {
4
5      public static void main(St
6          IncorrectSemaphoreExamp
7      }
8  }
9
10 class IncorrectSemaphoreExampl
11
12     public static void example
13
14         final Semaphore semaph
15
16         Thread badThread = new
17
18             public void run() 
19
20                 while (true) {
21
22                     try {
23                         semaph
24                     } catch (I
25                         // han
26                     }
27
28                     // Thread 
29                     // exceptio
30                     throw new 
31
```

Incorrect Use of Semaphore

The above code when run would time out and show that one of the threads threw an exception. The code is never able to release the semaphore causing the other thread to block forever. Whenever using locks or semaphores, remember to unlock or release the semaphore in a **finally** block. The corrected version appears below.

```
import java.util.concurrent.Se

class Demonstration {

    public static void main(St
        CorrectSemaphoreExample
    }
}

class CorrectSemaphoreExample

    public static void example

        final Semaphore semaph

        Thread badThread = new

            public void run()

                while (true) {

                    try {
                        semaph
                        try {
                            th
                        } catc
                            //
                            re
                        } fina
                            Sy
                            se
```

Running the above code will print the `Exiting Program` statement.