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.

```
import java.util.concurrent.Semaphore;

class Demonstration {

    public static void main(String args[]) throws InterruptedException {
        IncorrectSemaphoreExample.example();
    }
}

class IncorrectSemaphoreExample {

    public static void example() throws InterruptedException {

        final Semaphore semaphore = new Semaphore(1);

        Thread badThread = new Thread(new Runnable() {

            public void run() {

                while (true) {

                    try {
                        semaphore.acquire();
                    } catch (InterruptedException ie) {
                        // handle thread interruption
                    }

                    // Thread was meant to run forever but runs into an
                    // exception that causes the thread to crash.
                    throw new RuntimeException("exception happens at runtime.");

                    // The following line to signal the semaphore is never reached
                    // semaphore.release();
                }
            }

        });

        badThread.start();

        // Wait for the bad thread to go belly-up
        Thread.sleep(1000);

        final Thread goodThread = new Thread(new Runnable() {

            public void run() {
                System.out.println("Good thread patiently waiting to be signalled.");
                try {
                    semaphore.acquire();
                } catch (InterruptedException ie) {
                    // handle thread interruption
                }
            }
        });

        goodThread.start();
        badThread.join();
        goodThread.join();
        System.out.println("Exiting Program");
    }
}
```

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.Semaphore;

class Demonstration {

    public static void main(String args[]) throws InterruptedException {
        CorrectSemaphoreExample.example();
    }
}

class CorrectSemaphoreExample {
    public static void example() throws InterruptedException {

        final Semaphore semaphore = new Semaphore(1);


        Thread badThread = new Thread(new Runnable() {

            public void run() {

                while (true) {

                    try {
                        semaphore.acquire();
                        try {
                            throw new RuntimeException("");
                        } catch (Exception e) {
                            // handle any program logic exception and exit the function
                            return;
                        } finally {
                            System.out.println("Bad thread releasing semahore.");
                            semaphore.release();
                        }

                    } catch (InterruptedException ie) {
                        // handle thread interruption
                    }
                }
            }
        });

        badThread.start();

        // Wait for the bad thread to go belly-up
        Thread.sleep(1000);

        final Thread goodThread = new Thread(new Runnable() {

            public void run() {
                System.out.println("Good thread patiently waiting to be signalled.");
                try {
                    semaphore.acquire();
                } catch (InterruptedException ie) {
                    // handle thread interruption
                }
            }
        });

        goodThread.start();
        badThread.join();
        goodThread.join();
        System.out.println("Exiting Program");
    }
}
```

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