Exceptions in threads

If one of the threads of your program throws an exception that is not caught by any method within the invocation stack, the thread will be terminated. If such an exception occurs in a single-threaded program, the entire program will stop, because the JVM terminates the running program as soon as there are no more **non-daemon** threads left.

```
public class SingleThreadProgram {
   public static void main(String[] args) {
      System.out.println(2 / 0);
   }
}
```

This program outputs:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
  at
org.example.multithreading.exceptions.SingleThreadProgram.main(Single
ThreadProgram.java:6)

Process finished with exit code 1
```

The code 1 means the process was finished with an error.

If an error occurs inside a new thread we've created, the whole process will not be stopped:

```
public class ExceptionInThreadExample {
   public static void main(String[] args) throws InterruptedException {
      Thread thread = new CustomThread();
      thread.start();
      thread.join(); // wait for thread with exception to terminate
      System.out.println("I am printed!"); // this line will be printed
   }
}

class CustomThread extends Thread {

   @Override
   public void run() {
      System.out.println(2 / 0);
   }
}
```

Despite the uncaught exception, the program will be successfully completed.

Exception in thread "Thread-0" java.lang.ArithmeticException: / by zero
at
org.example.multithreading.exceptions.CustomThread.run(ExceptionInThr
eadExample.java:15)
I am printed!

Process finished with exit code 0

The code 0 means the process successfully finished.

Exceptions in different threads are handled independently. While the process has an alive non-daemon thread, it won't be stopped in case an uncaught exception occurs. Still the good practice is to handle exceptions in threads.