Output of an Uncaught Exception

The program in the Activity window tries to print the value of 5/0. Division by 0 is not defined, so the attempt to divide by 0 during execution is an error. Java handles this error by "throwing an Exception"--- it causes the program to terminate abnormally.

```
public static void main(String[] args) {
    System.out.println(5/0);
}
```

We'll explain what an Exception is later. For now, we concentrate on the consequences of throwing this Exception. Throwing it causes some messages to appear in the Java console:

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at Ex.main(Ex.java:4)
```

The first line tells you that an Exception occurred and that it was an arithmetic exception; it also tells you where this Exception is defined ---in package java.lang, class ArithmeticException. **Note:** *The output has changed over the years as Java seeks to improve. The content is there, but the format may be different!*

The next line tells you that the Exception occurred in method main of class Ex, which happens to be in source file Ex.java, and it was on line 4.

You already know that execution of the program begins by the system calling method main. There may be more lines telling you that the call to main occurred in a method run, somewhere deep in the bowels of the system, in source file JMAWTContextImpl.java. Finally, the last line tells you that this method run was called by another method run in class Thread, in source file Thread.java. *If these lines don't appear, that's fine!*

The output might look slightly different, depending on the system you are using, but the contents should be the same.

The important point for you is that you now know what method was being executed when the Exception occurred, *main*, and in what file the method is defined, *Ex.java*. This information will help you get started on finding out why the Exception occurred. The rest of the information is not very helpful, and we will not show it again.

When an Exception occurs, you have to study your program to find out why and then correct the error. As mentioned above, the messages in the Java console tell you the kind of Exception that occurred and the method that was being executed at the time, and this can be helpful. But the Java console will contain a bit more. To illustrate, let us change the program so that the division by 0 occurs within a different method. We'll get rid of the comment to save space, add two more static methods, and change the body of method main to call method first.

```
public static void main(String[] args) {
    first();
}
public static void first() {
   second();
}
public static void second() {
   System.out.println(5/0);
}
```

Execution now begins with a call of main, which calls first, which calls second, which attempts to divide by 0.

The list of calls that have been started but have not completed is called the "call stack". When the attempt to divide by zero occurs, the same exception is thrown, and the following appears in the Java console.

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
    at A0.second(A0.java:11)
    at A0.first(A0.java:8)
    at A0.main(A0.java:4)
```

As before, the first line explains the nature of the Exception. The next line tells you that the Exception occurred while executing a call on method second of class Ex, which is defined in file Ex.java. The next line says that second was called from first, and the next line says that first was called from main. And you also see the line number on which each call appeared.

Thus, when an Exception occurs, the Java console describes the call stack: the complete stack of methods that have been called but that have not yet completed.

You can use this stack of calls to help figure out how your program got to the point of throwing an exception.