```
    catch (FileNotFoundException fnfe)
    {
      System.out.println(fnfe + " caught in method m1");
      fnfe.printStackTrace(System.err);
    }
  }
```

Remember when using this technique to redirect the standard error stream output to a file. This enables the file to be parsed to determine if any exceptions were generated. On Windows NT and UNIX, this is done by directing standard error and standard output to the same file as follows:

```
java Test > out.dat 2>&1
```

The best approach, however, is not to put off the dirty work of exception and error handling, but rather to deal with it as they arise.

PRAXIS 18: **Never hide an exception**

Exceptions are hidden when one is thrown from a `catch` or `finally` block during the processing of a previously thrown exception. Only the last exception generated is propagated to the calling method. If you are interested only in the fact that a method failed due to one or more exceptions, you might not care if a previously thrown exception is hidden. However, you want to avoid hiding exceptions when you want to know the original cause of a method's failure.

Often during `catch` or `finally` block processing, you call methods that can throw exceptions. How do you handle this so that previously thrown exceptions are not hidden? You need a mechanism to transfer *all* generated exceptions of a method to the calling method. Here is an example of the problem:

```
class Hidden
{
  public static void main (String args[])
  {
    Hidden h = new Hidden();
    try {
      h.foo();
    }
    catch (Exception e) {
      System.out.println("In main, caught exception: " +
                         e.getMessage());
    }
  }
```