Note that other than calling its parent classes through the *super* method, this new exception type doesn't do anything in particular. However, it is convenient to have our own named exception type so that the compiler will warn us of the type of exception we must catch when we attempt to create an instance of PrintSpooler.

Throwing an Exception

Let's write the skeleton of our Spooler class—we'll omit all of the printing methods and just concentrate on correctly implementing the Singleton pattern:

```
public class Spooler {
    //this is a prototype for a printer-spooler class
    //such that only one instance can ever exist
    static boolean instance_flag = false;    //true if one instance

    public Spooler() throws SingletonException
    {
       if(instance_flag)
           throw new SingletonException("Only one printer
           allowed");
       else
           instance_flag = true;        //set flag for one instance
        System.out.println("printer opened");
    }
}
```

Creating an Instance of the Class

Now that we've created our simple Singleton pattern in the PrintSpooler class, let's see how we use it. Remember that we must enclose every method that may throw an exception in a try-catch block.

```
public class singleSpooler
{
    static public void main(String argv[])
    {
        Spooler pr1, pr2;
        //open one printer--should always work
        System.out.println("Opening one spooler");
        try {
        pr1 = new Spooler();
        }
        catch (SingletonException e)
        (System.out.println(e.getMessage());}
        //try to open another printer--should fail
        System.out.println("Opening two spoolers");
        try {
```