Chapter 7: Review Exercise Solutions

R7.1

If you open a file for reading that doesn't exist, Java will throw a `FileNotFoundException`.

If you open a file for writing that doesn't exist, Java will create a new empty file.

R7.2

Java will throw a `FileNotFoundException` whose error message says "(Access is denied)".

R7.3

You need to use an escape character (an extra backslash) before the backslash, like this:

```
String filename = "c:\\temp\\output.dat";
```

R7.4

```
args[0]: -Dname=piglet
args[1]: -I\eeyore
args[2]: -v
args[3]: heff.txt
args[4]: a.txt
args[5]: lump.txt
```

R7.5

Throwing an exception creates a new exception and causes the current method to halt. Catching an exception occurs at a higher level than the method that threw it; usually the exception is processed, but can also be ignored.

R7.6

A checked exception indicates something beyond your control has gone wrong. Methods that throw these types of exceptions must surround a call with a try/catch block. An example is `FileNotFoundException`. These exceptions need to be declared with the `throws` reserved word if the method that throws them does not handle them.

An unchecked exception indicates an error in the code and does not need to be explicitly handled with a try/catch block. An example is `IndexOutOfBoundsException`.

R7.7

Because `IndexOutOfBoundsException` is an unchecked exception, and by definition unchecked exceptions do not need to be declared by the `throws` keyword. (These exceptions are caused by errors in code that a programmer should have detected during testing.)

R7.8

The next line of code processed after a `throw` statement is the first line in the first `catch` block hierarchy that processes that type of exception.

R7.9

If an exception does not have a matching `catch` clause, the current method terminates and throws the exception to the next higher level.

R7.10

Your program can do anything with the exception object that it chooses to. It can print it to `System.out`, ignore it, or anything else that is legal to program into the `catch` block.

R7.11

Not always, it's possible that the type of the exception handled in the `catch` block is an "ancestor" of the exception type that is actually passed into it. This is legal in Java since a superclass can always reference a subclass object.

R7.12

The `finally` clause is executed whether or not an exception is thrown. It is used whenever there is some cleanup necessary, like closing a file.

R7.13

The exception thrown by the `finally` block needs to be caught by a matching catch clause. In this example, the exception thrown by the `finally` block is caught by a second `catch` clause:

```
try
{
```



```
   try
   {
      Scanner in = new Scanner(new File("foo.txt"));
   }
   catch (FileNotFoundException e)
   {
      System.out.println("Can't find file.");
   }
   finally
   {
      throw new IllegalArgumentException("A different exception");
   }
}
catch (IllegalArgumentException e)
{
   System.out.println("It's illegal!");
}
```

R7.14

`next` can throw a `NoSuchElementException` if no more tokens are available or an `IllegalStateException` if the scanner is closed.

`nextInt` can throw everything that `next` can plus an `InputMismatchException` if the next token does not match the Integer regular expression, or is out of range.

These are all unchecked exceptions.

R7.15

Because the first line of the file is a 1, `readData` constructs an array to hold one value and, after the next line is read from the file, the program throws an `IOException` because it expected to reach the end of the file. The error report could be improved by printing the number of elements that the program expected to read. Then it becomes obvious that the input file does not indicate the correct number.

R7.16

As the program is set up, no. But `readFile` can throw a `NullPointerException` if it is passed a `null` value for the `String` filename argument. However, the program prevents this from occurring because the filename is initialized from `in`, which always returns a non-null value.

