Chapter 6 Handout

a)

```
1 // Report some basic information about a file.
2
3 import java.io.*; // for File
4
5 public class FileInfo {
6     public static void main(String[] args) {
7         File f = new File("hamlet.txt");


8         System.out.println("exists returns " + f.exists());
9         System.out.println("canRead returns " + f.canRead());
10        System.out.println("length returns " + f.length());
11        System.out.println("getAbsolutePath returns "
12                           + f.getAbsolutePath());
13    }
14 }
```

```
exists returns true
canRead returns true
length returns 191734
getAbsolutePath returns C:\data\hamlet.txt
```

If the program is unable to locate the specified input file, it will generate an error by throwing what is known as a `FileNotFoundException`. This particular exception is known as a *checked exception.*

Checked Exception

An exception that *must* be caught or specifically declared in the header of the method that might generate it.

Because `FileNotFoundException` is a checked exception, you can't just ignore it. Java provides a construct known as the `try/catch` statement for handling such errors (described in Appendix D), but it allows you to avoid handling this error as long as you clearly indicate the fact that you aren't handling it. All you have to do is include a *`throws` clause* in the header for the `main` method to clearly state the fact that your `main` method might generate this exception.

`throws` Clause

A declaration that a method will not attempt to handle a particular type of exception.

Now that we've introduced some of the basic issues involved in reading an input file, let's explore reading from a file in more detail. One way to process a file is token by token.

Token-Based Processing

Processing input token by token (i.e., one word at a time or one number at a time).

Recall from Chapter 3 the primary token-reading methods for the `Scanner` class:

- `nextInt` for reading an `int` value
- `nextDouble` for reading a `double` value
- `next` for reading the next token as a `String`

b)

For example, you might want to create a file called `numbers.dat` with the following content:

```
308.2 14.9 7.4
2.8


3.9 4.7 -15.4
2.8
```

```
1 // Program that reads five numbers and reports their sum.
2
3 import java.io.*;
4 import java.util.*;
5
6 public class ShowSum1 {
7     public static void main(String[] args)
8             throws FileNotFoundException {
9         Scanner input = new Scanner(new File("numbers.dat"));
10
11        double sum = 0.0;
12        for (int i = 1; i <= 5; i++) {
13            double next = input.nextDouble();
14            System.out.println("number " + i + " = " + next);
15            sum += next;
16        }
17        System.out.println("Sum = " + sum);
18    }
19 }
```

This program uses a variation of the cumulative sum code from Chapter 4. Remember that you need a `throws` clause in the header for `main` because there is a potential `FileNotFoundException`. The program produces the following output:

```
number 1 = 308.2
number 2 = 14.9
number 3 = 7.4
number 4 = 2.8
number 5 = 3.9
Sum = 337.19999999999993
```

c)

```
1 // Counts the number of words in Hamlet.
2
3 import java.io.*;
4 import java.util.*;
5
6 public class CountWords {
7     public static void main(String[] args)
8             throws FileNotFoundException {
9         Scanner input = new Scanner(new File("hamlet.txt"));
10        int count = 0;
11        while (input.hasNext()) {
12            String word = input.next();
13            count++;
14        }
15        System.out.println("total words = " + count);
16    }
17 }
```

```
total words = 31956
```