File I/O

This section will describe how to read from a text file and how to write to a text file. In both cases, you will need to include the lines:

```
import java.io.*;
import java.util.*;
```

You will also need to add:

```
throws IOException
```

to the end of the main method. Thus, the main method will look like this:

```
public static void main(String[] args) throws IOException {

}
```

This means that interacting with files can cause some unexpected problems (like if the file isn't there), but that we're not going to handle them. Later in the course we will discuss how to handle these problems.

Reading from a File

First, we need to open a connection to the file. We do this by creating a `Scanner` (like we do to get user input). This time, however, we'll connect the `Scanner` to a file:

```
Scanner s = new Scanner(new File(filename));
```

Here, `filename` is a string that is the name of the input file. This file should be stored in the same directory as the class that's using it. Alternatively, you can specify the absolute path of the file (e.g., "`C:\Documents and Settings\...`").

To read a single line in the file, do this:

```
String line = s.nextLine();
```

The first line of the file will be stored in the `line` variable. If we call `nextLine()` again, it will read the second line, and so on.

To see if you've reached the end of the input file, use the `hasNext()` command. This will return false if we're at the end of the file, and true otherwise.

When you're done reading from a file, you need to close it:

```
s.close();
```

Here's an example that opens the file "`in.txt`", and prints every line from the file to the screen:

```
Scanner s = new Scanner(new File("in.txt"));
while (s.hasNext()) {
     String line = s.nextLine();
     System.out.println(line);
}
s.close();
```

If `in.txt` looked like this:

Hi
Testing
example

Then this would print to the command line:

Hi
Testing
example

You will often read files that have a bunch of information on each line – say, a bunch of names separated by spaces. To process each name, read each line (as shown above), and then use a `StringTokenizer` to break apart each line.

Writing to a File

First, we need to open a connection to the file. Here, `filename` is the name of the file we want to write to. If this file doesn't exist, it will be created in the same directory as your code. If it does exist, everything in the file will be overwritten.

```
PrintWriter pw = new PrintWriter(new FileWriter(filename));
```

Now we can write to the file. Here's how to write a single line of text (and advance to the next line):

```
pw.println(stuff);
```

Here, `stuff` is the `String`, `int`, `char`, or `double` that you want to write to the file. This function works exactly like `System.out.println`. To print a line without advancing to the next line, do:

```
pw.print(stuff);
```

When you're done writing to the file, close it:

```
pw.close();
```

Closing the file is especially important after you've written information to it. When you write to a file, the information doesn't get immediately written. Instead, it gets stored in a temporary buffer. When the buffer gets full, the entire buffer is written to the file at once. When you close the file, it writes whatever is in the buffer to the file. If you forget to close it, then whatever information was still in the buffer will not get written to the file.

Example
These arrays will hold people's names and ages, respectively:

```
String[] names;
int[] ages;
```

Suppose those arrays have been initialized and filled with data. Suppose also that they are the same length, and that `names[i]` is a person's name, and `ages[i]` is that person's age. We want to print

name: age

for each person to the file `info.txt`. Here's how:

```
//open a connection to the file
PrintWriter pw = new PrintWriter(new FileWriter("info.txt"));

//print name: age for each person
for (int i = 0; i < names.length; i++) {
      pw.println(names[i] + ": " + ages[i]);
}

//close the file
pw.close();
```