SIENA*college*
Computer Science

Computer Science 120
Introduction to Programming
Siena College
Spring 2012

Topic Notes: Input/Output

We have seen some examples of input/output (I/O) from the keyboard, to the terminal window, and using files, mainly in the Hangman example. We will review and expand upon those ideas here.

The Java `main` method

- A `main` method may be used to test the operation of a class or to run a program.
- Its use for testing is usually informal, since it lacks the features of a more complete testing tool such as JUnit.
- When used for testing and testing is complete, `main` can be deleted, or it may be commented out and kept for future needs.
- In the Java language, when you execute a class with the Java interpreter, the runtime system starts by calling the class's `main` method. The `main` method then calls all the other methods required to run your application.

Input/Output

- A monitor is normally considered the standard output device.
- A keyboard is considered the standard input device.
- `System.out` sends output to the standard output device (*i.e.*, the monitor).
- `System.in` references the standard input device. Unfortunately `System.in` is not as simple and straightforward as `System.out`. `System.in` reads input only as byte values, which is usually not very useful since we usually need data in other formats. We can work around this using a combination of the `System.in` object and the `Scanner` class. For example:

```
int number;
Scanner keyboard = new Scanner(System.in);
System.out.print("Enter an integer value: ");
number = keyboard.nextInt();
```

- The `Scanner` class has methods for reading strings, bytes, integers, long integers, short integers, floats and doubles.
  - `nextFloat` – returns input as a `float`
  - `nextDouble` – returns input as a `double`
  - `nextInt` – returns input as an `int`
  - `nextLine` – returns entire line of input as a `String`
- In our Hangman example, we saw that we needed to add the lines

  ```
  import java.io.FileNotFoundException;
  import java.util.Scanner;
  ```

  to be able to make use of the class, and we had to handle the `FileNotFoundException`. In Hangman, we used a `try..catch` block, but we could also add the command `throws FileNotFoundException` to our method header to essentially tell Java we're not worried about that and to crash our program if the `Scanner` has trouble finding the file we requested.

A painfully simple example:

See Example: Payroll

Sometimes you want to read just a single character from the keyboard (*e.g.*, a y/n response). How do you do this?

```
String input;
char answer;

Scanner keyboard = new Scanner(System.in);
System.out.print("Are you having fun? (y=yes n=no) ");
input = keyboard.nextLine();
answer = input.charAt(0);
```

Reading from a file

Reading from a file rather than the keyboard does not add much complexity.

Instead of passing `System.in` to the `Scanner` constructor, you can pass a `File` object:

```
File f  = new File("hamlet.txt");
Scanner input = new Scanner(f);
```

which we can shorten to a single line:

```
Scanner input = new Scanner (new File("hamlet.txt"));
```

A simple program to count the words in a file:

See Example: CountWords

Tokenizing

We often will want to process input token by token (one word or number at a time). This is called *tokenizing*.

For example, let's look at a program that processes a data file with employees' work information on different lines of input. Each line consists of an employee id, name and then the number of hours worked each day:

```
101 Erica 7.5 8.5 10.25 8 8.5
783 Erin 10.5 11.5 12 11 10.75
114 Simone 8 8 8
238 Ryan 6.5 8 9.25 8
```

A `main` method to do this might look like this:

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

  Scanner input = new Scanner (new File("hours.dat"));
  while (input.hasNext()) {
    int id = input.nextInt();
    String name = input.next();
    double sum = 0.0;
    while (input.hasNextDouble()) {
      sum += input.nextDouble();
    }
    System.out.println("Total hours worked by " + name +
  " (id#" + id + ") = " + sum);
  }
}
```

Unfortunately, this program would result in an error...

We need to get the program to stop reading data for the current employee when it gets to the end of an input line. Reading the file line by line guarantees that you dont accidentally combine data for two employees.

- To read a whole line, we can use `input.nextLine()`, but it returns a `String`.
- We can write a method, we'll call it `processLine` that takes a `String` as a parameter, that can pull apart the `String`.
- Fortunately, `Scanner`s are very flexible, and you can even attach them to a `String`:

```
Scanner input = new Scanner("18.4 17.9 8.3 2.9");
```

A working program that does this:

See Example: HoursWorked