Java for Non Majors

CGS 3416: Spring 2017
Department of Computer Science, Florida State University

Reading Strings using the Scanner Class

February 13, 2017

The Scanner class provides the easiest way to obtain console input in Java. It is a combination of a stream reader and a parser. When asked to read a particular type of data, it scans the input stream, skipping over whitespace until it reaches the right kind of data. It then reads all the tokens until it reaches the next whitespace character. Finally, it converts the text it just read into the data type requested and returns the value.

For example, consider the following stream. The Scanner is asked to read in an integer.

3	6	1	9	\n	H	e	l	l	o		W	o	r	l	d	\n

Once the integer is read in, the stream looks like this.

\n	H	e	l	l	o		W	o	r	l	d	\n

1. What happens if we try to read a type that doesn't match what is on the input stream?

This will throw an `InputMismatchException`. That means the program will crash unless we have dealt with the exception.

2. What happens to the newlines?

The newline that is put into the stream when the user presses the Enter key after input is left on the stream when the Scanner is done reading. This doesn't normally impact the behavior of the Scanner, since most Scanner methods ignore the whitespace on the stream until they see some kind od data. The only exception to this is Scanner.nextLine().

3. What is the deal with nextLine()

The nextLine method has a slightly different action. It reads all data in as text until it sees a newline character, and consumes the trailing newline as well. When we have used another Scanner method before using nextLine(), the newline from the previous entry is still on the stream. This causes the nextLine() method to read in an empty string.

Consider the stream above (the second one). Now, consider the following line of code:

```
String st = input.nextLine();
```

↓ sees this newline and terminates.

\n	H	e	l	l	o		W	o	r	l	d	\n

4. How do we solve this problem?

Whenever we use nextLine() after any other input statement, we will encounter this problem. The way to deal with this is to read in the preceding newline into a dummy String and then continue as normal. So, if we wanted to fix the previous line of code, we would write:

```
String junk = input.nextLine();
String st = input.nextLine();
```

↓ is read into "junk".

\n	H	e	l	l	o		W	o	r	l	d	\n

↑ "st" starts here.