CHAPTER 7.1-7.3

INPUT/OUTPUT and FILES

Portions taken from slides by Donald W. Smith
TechNeTrain.com

Contents

- Reading and Writing Text Files
- Text Input and Output
- Command Line Arguments

Text File Input

If you don't add the character encoding it uses the default. OK for class, but might bite your later.

- Create an object of the `File` class
  - Pass it the name of the file to read in quotes

```
File inputFile = new File("input.txt");
```

- Then create an object of the `Scanner` class
  - Pass the constructor the new `File` object

```
Scanner in = new Scanner(inputFile, "UTF-8");
```

- Then use `Scanner` methods such as:
  - `next()`
  - `nextLine()`
  - `hasNextLine()`
  - `hasNext()`
  - `nextDouble()`
  - `nextInt()...`

```
while (in.hasNextLine())
{
   String line = in.nextLine();
   // Process line;
}
```

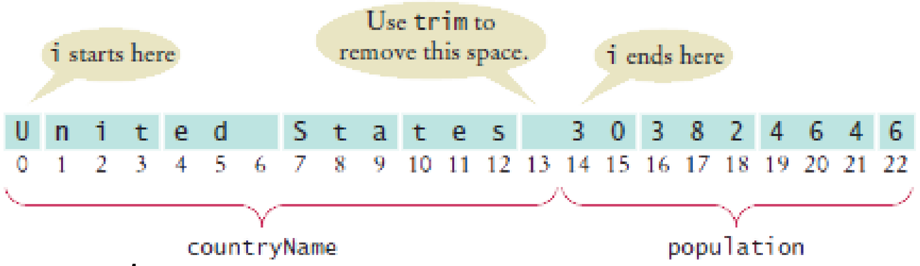
Scanning Text Input

- You have several ways to read text input
  - `nextLine()` – reads entire line
  - `next()` – reads one word/token
  - `nextInt()`, `nextDouble()` – reads token and then converts to the appropriate type. Stops when a character can't be converted.
  - `useDelimiter()` – regular expression that determines which characters are used to surround tokens
    - `useDelimter("[^A-Za-z]+");` // only letters allowed
    - `useDelimter("");` // reads a single character
- Converting text to numbers?
  - Use `Integer.parseInt()`, `Double.parseDouble()`

Classifying Characters

- `nextLine`, `next` return String
- For that matter, `substring` returns String
- Should you ever use char?
- In general, no. In almost all cases, people who use char do it wrong
- Instead, use strings of length 1
  - Or length 2 for those Unicode code points that require two char values
  - Like this cat with heart-shaped eyes 😻
- char is useful in the rare case that you need to classify characters
  - isDigit, isLetter, isUpperCase, isLowerCase, isWhiteSpace
  - char ch = in.next().charAt(0); if (Character.isLetter(ch)) ...

3 Ways to Process Lines



- Read entire line and then process tokens in the line
  - 1. Use **`lastIndexOf`**, **`substring`**, etc
  - 2. Use a secondary **`Scanner`** on line

    -
      ```
      Scanner strIn = new Scanner(line);
      String country = strIn.next();
      while (! strIn.hasNextInt()) {
          country += "  " + strIn.next();
      }
      int population = strIn.nextInt();
      ```
  - 3. Use the **`split(regex)`** method to convert to array
    - `str.split("  ");` // spaces (some tokens may be spaces)
    - `str.split("\\s+");` // multiple spaces collapsed
    - `str.split("[^A-Za-z0-9]");` // only letters & numbers

Converting to Numbers

- `Integer.parseInt()`, `Double.parseDouble()`
- Can apply to substrings of a line
- You may need to trim
  - `Integer.parseInt(" 13")` throws an exception
- Beware of currency symbols and decimal separators
  - `Double.parseDouble("$10.95")` throws an exception
  - `Integer.parseInt("1,000,000")` throws an exception

Lecture 4 Clicker Question 1

- An input file contains lines such as

```
Fred 40
Wilma 28
Mary Ann 30
```

- Consider this code to process a line:

```
String line = in.nextLine();
int n = line.lastIndexOf(" ");
String name = line.substring(0, n);
int age = Integer.parseInt(line.substring(n));
```

- What is the problem with this code?
  1. The name includes the trailing space
  2. The call to `Integer.parseInt` throws an exception
  3. Double names (such as “Mary Ann”) are not stored in name
  4. The code won't work with characters such as 😻 that require two char values

Split

- `str.split(regex)` splits the string into an array
- Example: `line.split(" ")` splits along spaces
  - If line is `"Mary Ann 30"`, get an array `["Mary", "Ann", "30"]`
- What if there is more than one space between tokens?
- Use regular expression `line.split("\\s+")`
  - `\s` matches any whitespace
  - `+` means one or more
- Or split along anything that's not a letter or number: `"[^A-Za-z0-9]"`
  - You've seen this with `Scanner.useDelimiter`
  - `[A-Z]` means all letters from A to Z
  - `^` means “not”

Lecture 4 Clicker Question 2

- Complete this program to find the sum of the numbers in the second to last column. Use `split`.

```
Abraham Lincoln          6 ft 4 in       193 cm
Lyndon B. Johnson        6 ft 3 1/2 in   192 cm
Thomas Jefferson         6 ft 2 1/2 in   189 cm
...
```

- What result do you get?
  1. 179.81
  2. 7732
  3. 7842
  4. Something else

Caution: mixing next and nextLine

- Input file

```
1729
Mary Ann
1730
Wilma
```

- Initially, the input contains

```
1 7 2 9 \n H a r r y
```

- Call

```
int studentID = in.nextInt();
```

- Now the input contains

```
\n H a r r y
```

- If you call `nextLine`, you don't read `Harry`. Reads an empty string!
- Remedy: a call to `nextLine` after reading the ID:

```
int studentID = in.nextInt();
in.nextLine(); // Consume the \n
String name = in.nextLine();
```

Lecture 4 Clicker Question 3

- Suppose the input contains the characters `6,995.00 12`. What is the value of price and quantity after these statements?

```
double price = in.nextDouble();
int quantity = in.nextInt();
```

1. price is `6995.0` and quantity is `12`
2. price is `"6,995.00"` and quantity is `12`
3. price is `6`, then an exception is thrown because the comma is not a valid part of an integer
4. an exception is thrown because a comma is not a valid part of a floating-point number

Text File Output

- Create an object of the `PrintWriter` class
  - Pass it the name of the file to write in quotes

```
PrintWriter out = new PrintWriter("output.txt");
```

  - If output.txt exists, it will be emptied
  - If output.txt does not exist, it will create an empty file

`PrintWriter` is an enhanced version of `PrintStream`

  - `System.out` is a `PrintStream` object!

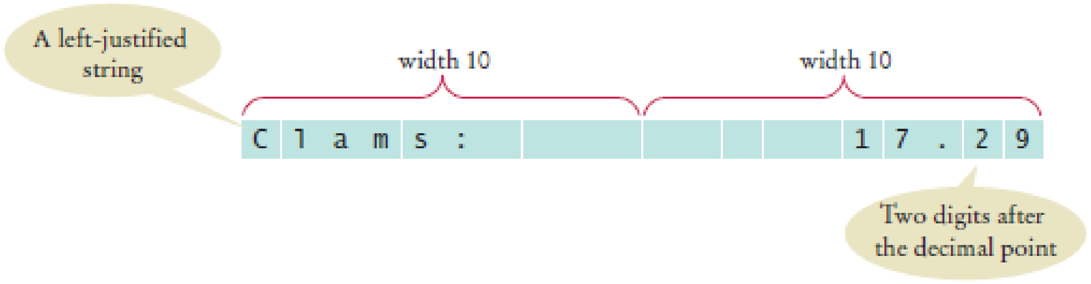
```
System.out.println(“Hello World!”);
```

- Then use `PrintWriter` methods such as:
  - `print()`
  - `println()`
  - `printf()`

```
out.println("Hello, World!");
out.printf("Total: %8.2f\n", totalPrice);
```

Useful Format Specifiers

- out.printf and String.format are your friends
- Alignment



  - `out.printf("%-10s%10.2f", items[i] + ":", prices[i]);`
  -
```
Clams:         19.95
Lobsters:     109.95
```

- x prints hexadecimal: `String.format("%4x", s.charAt(0));`
- Exotic flags
  - %,.2f prints decimal separators 100,000.00
  - %04d prints leading zeroes 0001

Lecture 4 Clicker Question 4

- Have another look at the example with the clams. Why can't you use the simpler form

```
out.printf("%-10s:%10.2f", items[i], prices[i]);
```

1. A colon is not a valid flag
2. The colon would be at the wrong place
3. You can't format items[i] as a string with %s
4. You can't have a negative width of -10

Closing Files

- You must use the `close` method before file reading and writing is complete
  - Closing a `Scanner`

```
while (in.hasNextLine())
{
   String line = in.nextLine();
   // Process line;
}
in.close();
```

Your text may not be saved to the file until you use the `close` method!

- Closing a `PrintWriter`

```
out.println("Hello, World!");
out.printf("Total: %8.2f\n", totalPrice);
out.close();
```

Exceptions Preview

- One additional issue that we need to tackle:
  - If the input or output file for a Scanner doesn’t exist, a FileNotFoundException occurs when the Scanner object is constructed.
  - The PrintWriter constructor can generate this exception if it cannot open the file for writing.
    - If the name is illegal or the user does not have the authority to create a file in the given location

Exceptions Preview

- Add two words to any method that uses File I/O

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

  - Until you learn how to handle exceptions yourself

And an important `import` or two..

- Exception classes are part of the `java.io` package
  - Place the `import` directives at the beginning of the source file that will be using File I/O and exceptions

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

public class LineNumberer
{
   public void openFile() throws FileNotFoundException
   {
      . . .
   }
}
```

7.3 Command Line Arguments

- Text based programs can be 'parameterized' by using command line arguments
  - Filename and options are often typed after the program name at a command prompt:

```
>java ProgramClass -v input.dat
```

```
public static void main(String[] args)
```

  - Java provides access to them as an array of `Strings` parameter to the main method named `args`

```
args[0]: "-v"
args[1]: "input.dat"
```

  - The `args.length` variable holds the number of args
  - Options (switches) traditionally begin with a dash '-'

Command-Line Arguments

- Common to use the command line for automating tasks
- Consider a program Total.java
- What if we have a whole bunch of input files?
- If we can specify the arguments on the command line, this would be easy:

```
for %%F in (*.txt) do java Total %%F %~nF.out
```

- Get arguments in args array
- Now you finally know what this means:

```
public static void main(String[] args)
```

- Common to use flags that start with -
  - -d for decoding in CaesarCipher.java (book example)

Example: Total.java (1)

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

/**
   This program reads a file with numbers, and writes the numbers to another
   file, lined up in a column and followed by their total.
*/
public class Total
{
   public static void main(String[] args) throws FileNotFoundException
   {
      // Prompt for the input and output file names

      Scanner console = new Scanner(System.in);
      System.out.print("Input file: ");
      String inputFileName = console.next();
      System.out.print("Output file: ");
      String outputFileName = console.next();

      // Construct the Scanner and PrintWriter objects for reading and writing

      File inputFile = new File(inputFileName);
      Scanner in = new Scanner(inputFile);
      PrintWriter out = new PrintWriter(outputFileName);
```

More import statements required! Some examples may use `import java.io.*;`

Note the throws clause

Example: Total.java (2)

```
        // Read the input and write the output

        double total = 0;

        while (in.hasNextDouble())
        {
           double value = in.nextDouble();
           out.printf("%15.2f\n", value);
           total = total + value;
        }

        out.printf("Total: %8.2f\n", total);

        in.close();
        out.close();
     }
  }
```

Don’t forget to close the files before your program ends.

Let's Try It Out

- Complete the **`FileAnalyzer`** program.
- Your task is to
  - Open a text file for reading
  - Read all of the words in the file
  - Find (and return) the longest word in the file
  - If there are multiple words with the same length, return the first of the maximum length words.
- Check your code by running **`TestFileAnalyzer`**
  - You'll find it in the tests package
  - Paste the results into your IC document

Let's Try It Out

- Complete the `WordCounter` program.
- Your task is to complete `countWords` method
  - Count the number of words per line
  - Print out each line of the poem preceded by number of the number of words in that line.
  - Note that method does NOT open files
- Check your code by running `TestWordCounter`
  - You'll find it in the tests package
  - Paste the results into the IC document