Principles of Programming

Module 7: Input/Output and Exception Handling

Reading and Writing Text Files, Text Input and Output

Today's Objectives

- Topics
  - 7.1 Reading and Writing Text Files
  - 7.2 Text Input and Output
  - *Games!*

Last Lecture *on a Page*

Medals.java
RockPaperScissors.java

- Java allows arrays with more than one dimension.

```
final int COUNTRIES = 7;
final int MEDALS = 3;
int[][] counts = new int[COUNTRIES][MEDALS];
```

rows cols

```
int[][] counts =
{
  { 1, 0, 1 },
  { 1, 1, 0 },
  { 0, 0, 1 },
  { 1, 0, 0 },
  { 0, 1, 1 },
  { 0, 1, 1 },
  { 1, 1, 0 }
};
```

rows

cols

- Print the data, with totaled rows:

```
for (int i = 0; i < COUNTRIES; i++) {
   int total = 0;
   for (int j = 0; j < MEDALS; j++) {
      System.out.printf("%8d", counts[i][j]);
      total = total + counts[i][j];
   }
   System.out.println(total);
}
```

- `ArrayList` provides a more functional List.

```
ArrayList<String> names = new ArrayList<String>();
```

Its a *generic* class, so you have to tell it what type will be in the ArrayList.

- `add`: add an element
- `get`: return an element
- `remove`: delete an element
- `set`: change an element
- `size`: current length

Reading and Writing Text Files

- Text files are commonly used to store information.
  - Both numbers and words can be stored as text.
  - They are the most ‘portable’ types of data files.

- The `Scanner` class is used to read text files.
  - We have used it to read from the keyboard (`System.in`).
  - Use familiar `nextInt`, `hasNextDouble`, etc. methods.
  - Reading from a file requires using the `File` class.

- The `PrintWriter` class is used to write text files.
  - We have used a similar object, `PrintStream`, to write to the console (`System.out`).
  - Use familiar `print`, `println` and `printf` methods.
  - Writing to a <u>does not</u> require using the `File` class.

Text File Input

java.io.File

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

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

java.util.Scanner

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

```
Scanner in = new Scanner(inputFile);
```

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

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

Text File Output

java.io.PrintWriter

- 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.

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

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

Closing Files

- You must use the `close` method when 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

`java.io.FileNotFoundException`

- One additional issue that we need to tackle:
  - If the input 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.

- 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.

An Important `import` or Two...

- As we remember, the `Scanner` class is in the `java.util` package.
- The `File`, `PrintWriter` and `FileNotFoundException` classes are part of the `java.io` package.

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

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

Total.java

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

public class Total {

   public static void main(String[] args) throws FileNotFoundException {
      Scanner console = new Scanner(System.in);

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

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

      double total = 0;
      while (in.hasNextDouble()) {
         double value = in.nextDouble();
         out.printf("%15.2f\n", value);
         total = total + value;
      }
      out.printf("Total: %15.2f\n", total);

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

Input File (line #'s shown):

```
1 12.38
2 376.76
3 47.4
4 8.37
5 0.36
6 23.874
```

Output File (line #'s shown):

```
1            12.38
2           376.76
3            47.40
4             8.37
5             0.36
6            23.87
7 Total:           469.14
```

Common Error

- Backslashes in File Names
  - When using a `String` literal for a file name with path information, you need to supply each backslash twice:

```
File inputFile = new File("c:\\homework\\input.dat");
```

  - A single backslash inside a quoted string is the *escape character*, which means the next character is interpreted differently (for example, '`\n`' for a newline character).

Processing Text Input

- There are times when you want to read input by:
  - Each Word
  - Each Line
  - One Number
  - One Character

- Java provides methods of the `Scanner` and `String` classes to handle each situation.
  - It does take some practice to mix them though!

Reading Words

- In the examples so far, we have read text <u>one line at a time</u>.

```
while (in.hasNextLine()) {
  String input = in.nextLine();
  ...
}
```

- To read <u>one word at a time</u> in a loop, use:
  - the `Scanner` object's `hasNext()` method to test if there is another word.
  - the `Scanner` object's `next()` method to read one word.

```
while (in.hasNext()) {
  String input = in.next();
  System.out.println(input);
}
```

- Input:

Mary had a little lamb

Output:

Mary
had
a
little
lamb

White Space

- The `Scanner`'s `next()` method has to decide where a word starts and ends.
- It uses simple rules:
  - It consumes all *white space* before the first character.
  - It then reads characters until the first *white space* character is found or the end of the input is reached.
- What is *white space*?
  - Characters used to separate:
    - Words
    - Lines

"Mary had a little lamb,\n
her fleece was white as\tsnow"

Common White Space

' '	Space
`\n`	NewLine
`\r`	Carriage Return
`\t`	Tab
`\f`	Form Feed

Reading Characters (a *trick*)

- There are no `hasNextChar()` or `nextChar()` methods in the `Scanner` class.
  - Instead, you can set the `Scanner` to use an 'empty' delimiter (`""`).

```
Scanner in = new Scanner(. . .);
in.useDelimiter("");

while (in.hasNext()) {
  char ch = in.next().charAt(0);
  // Process each character
}
```

ABCDEF...

`in.next()` returns `"A"`

`char ch = 'A'`

  - `next` returns a one character `String`
  - Use `charAt(0)` to extract the character from the `String` at index 0 to return a single `char`.

Classifying Characters

- The Character class provides several useful methods to classify a character:
  - Pass them a `char` and they return a `boolean`.

```
if ( Character.isDigit(ch) ) …
```

Character
+isDigit(c: char): boolean +isLetter(c: char): boolean +isUpperCase(c: char): boolean +isLowerCase(c: char): boolean +isWhiteSpace(c: char): boolean +toUpperCase(c: char): char +toLowerCase(c: char): char

Converting Strings to Numbers

- Strings can contain *digits*, not *numbers.*
  - They must be converted to numeric types.
  - ‘Wrapper’ classes provide a `parseInt` method.

A "wrapper" class is a Class defined by Java that works like one of the primitive types. Example: `Character`

‘3’	‘0’	‘3’	‘8’	‘2’	‘4’	‘6’	‘4’	‘6’

```
String population = "303824646";
int populationValue = Integer.parseInt(population);
```

java.lang.Integer
java.lang.Double

‘3’	‘.’	‘9’	‘5’

```
String priceString = "3.95";
double price = Double.parseInt(priceString);
```

Reading Other Number Types

- The `Scanner` class has methods to test and read almost all of the primitive types.

Data Type	Test Method	Read Method
`byte`	`hasNextByte`	`nextByte`
`short`	`hasNextShort`	`nextShort`
`int`	`hasNextInt`	`nextInt`
`long`	`hasNextLong`	`nextLong`
`float`	`hasNextFloat`	`nextFloat`
`double`	`hasNextDouble`	`nextDouble`
`boolean`	`hasNextBoolean`	`nextBoolean`

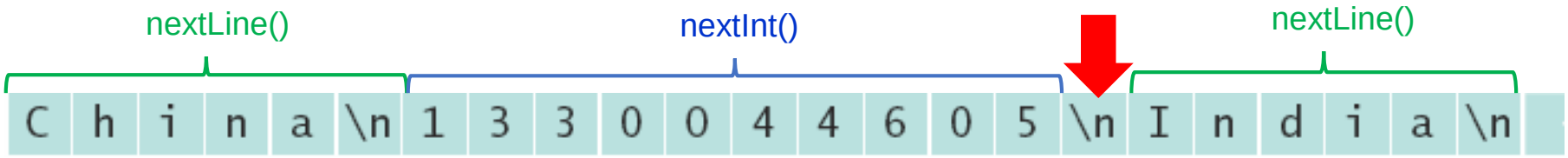
- What is missing?
  - Right, no `char` methods!
  - Set the `Scanner` to use an ‘empty’ delimiter (`" "`).

Mixing Number, Word and Line Input

- `nextDouble` (and `nextInt`…) do not "consume" white space following a number.
  - This can be an issue when calling `nextLine` after reading a number.
  - There is a ‘newline’ at the end of each line.

China
1330044605
India

```
while (in.hasNext()) {
  String countryName = in.nextLine();
  int population = in.nextInt();
  in.nextLine();   // Consume the newline
}
```



Hangman.java

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

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

      ArrayList<String> words = readWords();
      String word = pickWord(words);
      String guesses = "";

      Scanner in = new Scanner(System.in);

      System.out.println("\nHangman - E. Eckert");

      do {
         printWord(word, guesses);
         guesses = guesses + getUniqueGuess(in, guesses);
      } while (!wordGuessed(word, guesses));

      printWord(word, guesses);

      System.out.println("\nYou guessed it in " + guesses.length() + " tries.");

      in.close();
   }

   // Other methods
}
```

Pseudocode (stepwise refinement)

- Pick a random word.
- Continue until the word is guessed:
  - Print the word, showing correctly guessed letters and blanks.
  - Get another unique guess.

Hangman.java

```java
private static ArrayList<String> readWords() throws FileNotFoundException {
   ArrayList<String> words = new ArrayList<String>();

   File inputFile = new File("words.txt");
   Scanner in = new Scanner(inputFile);

   while (in.hasNext()) {
      words.add(in.next());
   }

   System.out.println("I know " + words.size() + " words.");

   in.close();
   return words;
}

static String pickWord(ArrayList<String> list) {
   int choice = (int) (Math.random() * list.size());
   return list.get(choice).toUpperCase();
}
```

63998 English words!

words.txt

```
aardvark
aardvarks
abaci
aback
abacus
abacuses
abaft
abalone
abalones
abandon
abandoned
abandoning
abandonment
abandons
abase
abased
```

Note: Lab8 gives detailed instructions on setting up a text file so that you can easily read it from your Java program.

The file `words.txt` is available on Canvas in Module 7.

CSE110 – Principles of Programming

Eric Eckert

eric.eckert@asu.edu