Reading/Writing to File

Elina, Helena, Setareh, Shaevitha

1.
Writing To File

Why Write to File?

- You can save it for future use
- Allows you to print hard copy results and distribute it
- Makes it more available to others
- Large data can be collected, stored and formatted according to needs

Outputting to a File

- The PrintWriter class can be used to create a file and write data to it
- Necessary imports:
  - import java.io.PrintWriter;
  - import java.io.IOException

    OR
  - import java.io*; //imports the entire I/O library (stands for input/output)
- It is declared similarly to the Scanner class:
  - PrintWriter output = new PrintWriter("place_file_name_here.txt");
- Different types of file can be created for data to be written to, depends on the extension, e.g. ".txt" creates a text file while ".docx" creates a word document

Sample Code

```
import java.io.PrintWriter;
import java.io.IOException;

public class PrintWriterExample {
        public static void main(String[]args) throws IOException{
                PrintWriter output = new PrintWriter("mytextfile.txt");
                //creates a text file

                output.println("hello"):
                                output.println("world!");
                                output.close();
                }
}
```

- It is important to always close the output stream, else the output will not show up on the file
- In the sample code a text file is created for data to be written to

```
import java.io.PrintWriter;

public class PrintWriterExample2 {
        public static void main (String[]args) throws Exception {
                                PrintWriter output = new
PrintWriter("file.txt");

                                output.println("Here is some sample
text.");
                                output.close();  //closes PrintWriter
```

2.
Reading to File

Why Read to File?

- Speed
- Less errors
- Easier to read individual cases
- Able to get data from sources other than the keyboard
- Changing information is easy (only on the file)
- More efficient (less time to test)
- Large amounts of data can be entered quickly
- At the testing stage, data can be carefully chosen to test the program

How to Read to File?

- The File class is Java's representation of a file or directory path
- It is used to identify/locate the file that is to be read
  - File file = new File("place_file_name_here"); //declaration statement
- Scanner class is used to read from file
  - Scanner read = new Scanner(file);
  - Not the same thing as "new Scanner(System.in)" which takes input from the keyboard
- Necessary imports
  - import java.util.Scanner;
  - import java.io.IOException;
  - import java.io.File;

Sample Code

```
import java.util.Scanner;
import java.io.File;
import java.io.IOException;

public class ReadingToFileDemo {
        public static void (String[] args) throws IOException {
                        File file = new File("text.txt"); //the file you are
reading from
                        Scanner read = new Scanner(file);

                        while (read.hasNext() == true) { //checks for tokens
                                        System.out.println(read.nextLine());
                //reads and outputs each line from file
                        }
                read.next(); //moves the cursor down
                        }
                }
```

"Note: When using the boolean class has method, e.g. hasNext(), hasNextLine(), the cursor doesn't actually move

- A loop is used to read the file, typically a while loop
- read.hasNext(), part of the boolean class, checks if there are tokens, a token is essentially a character
- If it returns true a line is read from file and printed to screen
- read.next(), read.hasNextLine() can also be used in substitute depending on the way the text is organized in the file
- For reading files with more than 1 line of text an additional command must be used to move the cursor down to the next line

Sample Code

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

public class ReadingToFileDemo2 {
        public static void main(String[] args) throws Exception
{
                Scanner input = new
Scanner(System.in);
                File file = new File("file10.txt");
                Scanner read = new Scanner(file);

                System.out.print("Please enter a
username: ");
                String username =
input.nextLine();
                while(read.hasNextLine() == true) {
                        read.nextLine();
                }
                read.close();
                read = new Scanner(file);
                boolean isUsername = false;

                while (read.hasNext() == true && isUsername ==
false) {
                        read.next(); //skips the first token
                        "username"
                        String temp = read.next();

                        if (temp.equals(username)) {
                                isUsername = true;
                        }
                }

                if (isUsername == true) {
                        System.out.println("You are logge
                        on.");
                } else {
                        System.out.println("You do not ha
                        valid account.");
                }
                read.close();
```

- In cases where information is read from file at 2 or more separate occasions, the reading to file Scanner must be reopened
- This is because after the read scanner is used the cursor is left at the end of the file with no more text to read
- In order to read info again the read Scanner has to be reopened to set the cursor back to the start of the file

3.
Exceptions

What is an Exception?

- It is a class
- A form of throwable that indicates conditions that a reasonable application might want to catch
- To “throw an exception” is to essentially throw any errors of a certain nature to allow the program to continue to run

Exception vs IOException

```
import java.io.PrintWriter;
import java.util.Scanner;
import java.io.File;
[...]
public static void main (String[] args) throws
Exception
```

```
import java.io.PrintWriter;
import java.util.Scanner;
import java.io.File;
import java.io.IOException;
[...]
public static void main(String[] args) throws IOException
```

- Both “throws Exception” and “throws IOException” are used
- IOException is a subclass of the Exception class, therefore any code that works with IOException will also work with Exception
- Both codes above can be used; however, generally it is better practice to be more specific

4.
Useful Methods

Scanner Class - Read to File

Type	Method	Descriptions
boolean	hasNext()	- Returns true if the scanner has another token in its input - Does not advance past any input
boolean	hasNextLine()	- Returns true if there is another line in the input of this scanner - Does not advance past any input
boolean	hasNextInt()	- Returns true if the next token in this scanner's input can be interpreted as an int value in the default radix using the nextInt() method
String	next();	- Finds and returns the next complete token from this Scanner
String	nextLine()	- Advances this scanner past the current line and returns the input that was skipped - Returns the rest of the current line, excluding any line separator at the end - Position is set to the beginning of the next line after

File Class - Read to File

Type	Method	Description
boolean	canExecute()	Tests whether the application executes the file.
boolean	canRead()	Tests whether the application can read from the file.
boolean	exists()	Checks whether the file or directory exists.
boolean	renameTo(File dest)	Renames the file.
boolean	setReadOnly()	Marks the file or directory named by this abstract pathname so that only read operations are allowed.


Thanks for
Listening!