Computer Science II
Handout 13

File input & output

- Java allows reading from (and writing to) files on your computer

- These operations are distinct

Files – Reading

- Files can be read using the FileReader class
    - A new object is created by using the filename as a parameter

- However, reading directly from a file this way can be inefficient
    - We always want to *buffer* the stream of data coming from the file

- A buffer is an intermediate location for holding data
    - We *could* read each character from a file by opening the file, copying a single character, then closing the file
    - Reading many characters this way would incur the opening/closing operations each time!
    - These could be costly in terms of processor time used

Files – Reading

- The BufferedReader class allows reading single lines from a file
  - This is done in an efficient way; we don't need to worry about it

- Each line is returned as a String
  - We may need to use the Integer, Double, etc. wrapper classes to convert these to primitive values

- To create a new BufferedReader object, you need to pass a FileReader object:

```
FileReader myFile = new FileReader("InputData.txt");
BufferedReader reader = new BufferedReader(myFile);
```

Files – Reading Example 1

- Create a program that will read lines from a file (given by the user), and print out these lines in sequence

Files – Reading

```
import java.util.Scanner;
import java.io.*;
public class BufferedReaderDemo1 {
      public static void main(String[] args) throws IOException {
            String line = null;
            String filename;
            Scanner kb = new Scanner(System.in);
            System.out.print("Enter the name of the file to read from: ");
            filename=kb.nextLine();










      }
}
```

Files – Reading Example 2

- Create a program that will read lines from a file (given by the user), and store each line in an ArrayList while removing duplicate lines

```
import java.util.ArrayList;
import java.util.Scanner;
import java.io.*;
public class BufferedReaderDemo2 {
        public static void main(String[] args) throws IOException {
                String line = null;
                ArrayList<String> myList = new ArrayList<String>();
                String fn;
                Scanner kb = new Scanner(System.in);
                System.out.println("Enter the name of the file to read from: ");
                fn = kb.nextLine();

                BufferedReader br = new BufferedReader(new FileReader(fn));
                line = br.readLine();







                System.out.println(myList);
        }
}
```

Files – Reading

- Using the BufferedReader class is a good way to get the data quickly into our program, line by line
  - This String data then needs to be processed (i.e., to extract numerical values, or find matching entries)

- We might also want to have more convenience while reading from the file
  - Like we have with the Scanner class!

- We can use the Scanner class if we operate on a File object

```
Scanner s = new Scanner(new File("Input.txt"));
```

Files – Reading

- In fact, the FileReader object could also have worked with a File object
    - It was just more straightforward to use the filename itself

```
FileReader fr = new FileReader(new File(“Input.txt”));
```

- Using the Scanner object on a File will involve the same methods as when we are operating on user/console input
    - However, we do need to make sure we close the File once we are done reading it

Files – Reading Example 3

- Write a program that reads words from a file (only one per line) and prints these to output, each on a separate line, using the Scanner class

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

public class ScannerDemo1 {
      public static void main(String[] args) throws IOException {
            String fn;
            Scanner kb = new Scanner(System.in);
            System.out.println("Enter the name of the file to read from: ");
            fn = kb.nextLine();

            Scanner input = new Scanner(new File(fn));





            input.close();  // Important!
      }
}
```

Files – Reading Example 4

- Write a program that reads words from a single line in a file and prints these to output, all in lowercase, using the Scanner class

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

public class ScannerDemo2 {
      public static void main(String[] args) throws IOException {
            String fn;
            Scanner kb = new Scanner(System.in);
            System.out.println(“Enter the name of the file to read from: ”);
            fn = kb.nextLine();

            Scanner input = new Scanner(new File(fn));





            input.close();  // Important!
      }
}
```

Files – Reading Example 5

- Write a program that reads class grades from a text file and computes the average grade and the highest grade
  - Assume that the text file will consist of a name (one word) and a numerical grade, one pair on each line
  - Print the name of the highest graded student

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

public class ScannerDemo3 {
        public static void main(String[] args) throws IOException {
                Scanner kb = new Scanner(System.in);
                System.out.println("Enter the name of the file to read from: ");
                Scanner input = new Scanner(new File(kb.nextLine()));
                int count = 0;
                double highest = -1, sum = 0, curGrade;
                String hName, curName;

                while(input.hasNext()) {







                }
                // .. Continued ..
```

```
            double avg = 0.0;




            System.out.println("\nAverage = " + avg);
            System.out.println("Highest scoring student was " + hName);


        }
    }
```

Files – Writing

- Writing to files is accomplished in a similar manner

- Use a BufferedWriter object
    - This in turn makes use of a FileWriter object

```
BufferedWriter writer = new BufferedWriter(new FileWriter("Output.txt"));
```

- Use the **`write`** method to add a String to the output file

Files – Writing Example 1

- Write a program that writes names (given by the user), one per line, to an output file (specified by the user)

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

public class BufferedWriterDemo1 {
        public static void main(String[] args) throws IOException {
                String line = null;
                Scanner kb = new Scanner(System.in);
                System.out.println("Enter the name of the file to write to: ");
                BufferedWriter bw = new BufferedWriter(new FileWriter(kb.nextLine()));

                System.out.println("Enter name (quit to end): ");
                String name = kb.nextLine();
                while(!name.equals("quit")) {






                }

                System.out.println("Data written to file!");
                bw.close();
        }
}
```

Files – Writing

- Another option is to use the **`PrintWriter`** class
  - This uses the same print and println statements that we are familiar with

```
PrintWriter writer = new PrintWriter(“Output.txt”);
```

Files – Writing Example 2

- Write a program that writes names (given by the user), one per line, to an output file (specified by the user)

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

public class BufferedWriterDemo2 {
        public static void main(String[] args) throws IOException {
                String line = null;
                Scanner kb = new Scanner(System.in);
                System.out.println("Enter the name of the file to write to: ");
                PrintWriter pw = new PrintWriter(kb.nextLine());

                System.out.println("Enter all names (quit to end): ");
                String name = kb.nextLine();
                while(!name.equals("quit")) {



                }

                System.out.println("Data written to file!");
                pw.close();
        }
}
```