Northeastern University

Lecture 5: Fundamentals of Programming - 5

Prof. Chen-Hsiang (Jones) Yu, Ph.D.

College of Engineering

Materials are edited by Prof. Jones Yu from Prof. Charlie Wiseman's materials.

Outline

- Introduction to Computation and Programming
- Variables, I/O, Types and Strings
- Control Flow and Conditions
- Methods
- Arrays
- File I/O

Outline

- Introduction to Computation and Programming
- Variables, I/O, Types and Strings
- Control Flow and Conditions
- Methods
- Arrays
- File I/O

I/O

- I/O stands for Input/Output.
- So far, we've used a Scanner object based on System.***in*** for all input (from the user's keyboard) and System.***out*** for all output (to the user's screen).
- System.***in*** and System.***out*** are predefined I/O objects that are available automatically in every Java program.

File I/O

- Files can also be used for input and output.
- Files can store large data sets for input into a program, to save the need to type in all the data values individually.
- Files store data that need to be available after the program ends.
  - » All values in memory or displayed on the screen will be lost when the program terminates.

File Input

`File` Objects

- In Java, files on your computer are represented with File objects.
- New File objects are created for each file that you want to read from or write to.
- For example: `File f = new File("test.txt");`
- There are many methods you can use with File objects, but we are going to focus on how to use them for input and output.

File Input

- Reading from a file is done with a Scanner object, the same as we've been doing for keyboard input.
- When you create a Scanner for file input, use the File object instead of System.***in***
- Example:

```
File f = new File("test.txt");
Scanner fin = new Scanner(f);
```

- Or, these can be combined into a single statement:

```
Scanner fin = new Scanner(new File("test.txt"));
```

`FileNotFoundException`

- When you create the Scanner with a File object, Java will open that file for reading.
- If the file doesn't exist, a `FileNotFoundException` will be thrown.
- You must use **`try`**/**`catch`** to check for and handle that exception.
  - Or declare that the method containing the Scanner will throw the exception.
  - Be sure to handle it somewhere in your program.

Closing Files

- When the file is no longer needed in the program, it's important that the file is closed.
- There are several ways to handle closing files.
- Newer versions of Java support a convenient mechanism to automatically close files.
  - » Based on **try** blocks
  - » Called try-with-resource

Example: File-based Scanner

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

public class ClassExamples {
    public static void main(String[] args) {
        try (Scanner fin = new Scanner(new File("test.txt"))) {
            // process the file here
        }
        catch (FileNotFoundException ex) {
            System.out.println("File test.txt not found!");
            System.exit(0);
        }
    }
}
```

Adding the Scanner creation here will cause it to be automatically closed after the try/catch block

Files in Eclipse

- You can add files to your Eclipse project directly.
- Right-click on the project (in Package Explorer), go to New, and Select File.
- Give the file a name (e.g., test.txt)
- The file will show up under the project heading and you will be able to access it directly in your programs in that project.

Reading from a File

- Once the file is opened via a Scanner object, read from it the same way that you do with any Scanner.
  - » Using the `nextInt()`, `nextDouble()`, `nextLine()`, and `next()` methods
- You will still need to catch `InputMismatchException` when calling `nextInt()` and `nextDouble()`.
- Don't forget to close the Scanner as soon as you are done with the file. (`input.close()`)

Example: File Reading

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

public class ClassExamples {
    public static void main(String[] args) {
        try (Scanner fin = new Scanner(new File("test.txt"))) {
            String firstLine = fin.nextLine();
            String secondLine = fin.nextLine();
            System.out.println(firstLine);
            System.out.println(secondLine);
        }
        catch (FileNotFoundException ex) {
            System.out.println("File test.txt not found!");
            System.exit(0);
        }
    }
}
```

Exercise

- Write a program that opens a file named integers.txt, then reads 5 integers from the file and prints each one out
- You will have to create the integers.txt file manually first and put at least 5 integers into it.

Answer

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

public class ClassExamples {
    public static void main(String[] args) {
        try (Scanner fin = new Scanner(new File("integers.txt"))) {
            for (int i = 1; i <= 5; i++) {
                int nextInt = fin.nextInt();
                System.out.println(nextInt);
            }
        }
        catch (FileNotFoundException ex) {
            System.out.println("File integers.txt not found!");
            System.exit(0);
        }
    }
}
```

NoSuchElementException

- If you try to read a value that isn't there, then a `NoSuchElementException` will be thrown.
- You can catch this exception as normal.
- Or you can use the `hasNextInt()`, `hasNextDouble()`, `hasNextLine()`, and/or `hasNext()` methods to check if there is another value left in the file BEFORE you do the read.

File Output

Writing to Files

- A Scanner can only read values out of a file.
- In order to write values into a file, use a `PrintWriter` object.
- Example:

```
File f = new File("testOut.txt");
PrintWriter fout = new PrintWriter(f);
```

- Or the two can be combined into a single statement:

```
PrintWriter fout = new PrintWriter(new File("testOut.txt"));
```

- Creating a `PrintWriter` object will automatically create the file if it doesn't already exist and will remove all existing data in the file if it does exist.
- The file will show up in Eclipse under the project entry (might need to refresh the project view).

Using a `PrintWriter`

- Creating a new `PrintWriter` might throw a `FileNotFoundException`
  - » You either catch it or declare that your method throws it.
- You can use the `print()`, `printf()`, and `println()` methods on a `PrintWriter` object.
  - » The same as with `System.out`
- Use the same modified `try` block to ensure that the `PrintWriter` will be closed as soon as it is done being used.

Example: Writing to a File

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

public class ClassExamples {
    public static void main(String[] args) {
        try (PrintWriter fout = new PrintWriter(new File("testOut.txt"))) {
            double value = 42.42;
            fout.println("Hello File World!");
            fout.print("value: ");
            fout.printf("%.2f%n", value);
        }
        catch (FileNotFoundException ex) {
            System.out.println("File testOut.txt not found!");
            System.exit(0);
        }
    }
}
```

Exercise

- Write a program that writes the numbers from 1 to 100 to a file named "numbers.txt"

Answer

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

public class ClassExamples {
    public static void main(String[] args) {
        try (PrintWriter fout = new PrintWriter(new File("numbers.txt"))) {
            for (int i = 1; i <= 100; i++) {
                fout.println(i);
            }
        }
        catch (FileNotFoundException ex) {
            System.out.println("File numbers.txt not found!");
            System.exit(0);
        }
    }
}
```

Wrap Up - File I/O Summary

- You can read from and write to files just like getting input and output with System.***in*** and System.***out*** .
- Use a `File` object to represent a file in your program
- Use a `Scanner` with a `File` to read from the file
  - » Use the `next()` methods to read values and `hasNext()` methods to check for more values
- Use a `PrintWriter` with a `File` to write to the file
  - » Use the `print()` or `printf()` methods to write values

More Exercises of File I/O
(Self-Study Materials)

Exercise

- Write a program that reads every line (one entire line at a time) from a file named "jediCode.txt".
- For each line, print both to the screen and to another file named "lineCounts.txt" how many characters were on that line.
- Note that you'll need to create the jediCode.txt file yourself before you run the program.

Answer

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;
public class ClassExamples {
     public static void main(String[] args) {
          try (
               Scanner fin = new Scanner(new File("jediCode.txt"));
               PrintWriter fout = new PrintWriter(new File("lineCounts.txt"));
          ) {
               while (fin.hasNextLine()) {
                    String nextLine = fin.nextLine();
                    System.out.println(nextLine.length());
                    fout.println(nextLine.length());
               }
          }
          catch (FileNotFoundException ex) {
               System.out.println("File not found!");
               System.exit(0);
          }
     }
}
```

Exercise

- Write a program that copies the contents of one file into another file.
- In particular, ask the user for the names of both the original (input) file and the new (output) file.
- Write a method that is passed the already created `Scanner` and `PrintWriter` objects to do all of the copying (reading and writing).

Answer

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.Scanner;
public class ClassExamples {
	public static void main(String[] args) {
		Scanner keyboardInput = new Scanner(System.in);
		System.out.print("Enter the input file name: ");
		String inputFileName = keyboardInput.next();
		System.out.print("Enter the output file name: ");
		String outputFileName = keyboardInput.next();
		try (
			Scanner fin = new Scanner(new File(inputFileName));
			PrintWriter fout = new PrintWriter(new File(outputFileName));
		) {
			copyFile(fin, fout);
		}
		catch (FileNotFoundException ex) {
			System.out.println("File not found!");
			System.exit(0);
		}
	}
	public static void copyFile(Scanner original, PrintWriter copy) {
		while(original.hasNextLine()) {
			String line = original.nextLine();
			copy.println(line);
		}
	}
}
```