GERMANNA
ACADEMIC CENTER
FOR EXCELLENCE

Java Programming: File Handling Handout

Working with files in programming is a common practice that enables reading, writing, and accessing content stored externally. Students in ITP 120 (Java Programming) often find file handling difficult when it comes to setting up their program and using essential functions and classes.

This handout is created to provide simple steps for accessing, reading, and writing files in Java programming. Each topic has been broken into sections with Java coding steps and examples.

Please note there are multiple ways to read files in Java, but this handout will focus solely on the content included in ITP 120.

Use the following links to easily navigate between the sections:

Accessing and Reading Files

In programming, it is important to know how to access and use files. This is useful since it allows the user to gather stored information and more details about the documents being used. In ITP 120, the *java.io.FileInputStream* must first be imported, along with the *java.io.IOException* class, to access and use any file. The Scanner class will also be included as it will read the information from the file.

```
Console
import java.util.Scanner;
import java.io.FileInputStream;
import java.io.IOException;
```

The FileInputStream class allows the user to read the "ones" and "zeros" that make up files, also called their byte stream, to gain information from the document. The IOException class is included to catch potential errors that could occur when reading a file, such as:

- The file doesn't exist
- Issues with reading the file
- Connection errors
- Issues with the code file

When using the FileInputStream class, a new object must be created and constructed with the file name as a String parameter. The FileStream object is then passed to the Scanner as a parameter; this process allows the Scanner to have access to the data on the document.

```
Console
FileInputStream object = new FileInputStream("myfile.txt");
Scanner scnr = new Scanner(object);
```

The file contains the information, and the scanner retrieves the data from inside the file. This data can be used to assign values to variables.

```
Disney.txt

Mickey Mouse 1928
Snow White 1937
Donald Duck 1934
```

When reading files and accessing the information on them, the programmer should have knowledge of the file’s contents and understand what to look for before creating the program. Otherwise, they risk causing an error.

```
Console
public class Disney_Characters{
Public static void main(String [] args){
FileInputStream file = new FileInputStream(“Disney.txt”);
Scanner scnr = new Scanner(file);

while scrn.hasNext(){

String fname = scnr.next();
String lname = scnr.next();
int year_created = scnr.nextInt();

System.out.println(“Character’s name: ” + fname + “ “ + lname);
System.out.println(“Creation Year: ” + year_created);
}

file.close();
}
```

The key line in the code is that the while loop uses the ‘hasNext’ method to check each line in the document for content. Once it reaches the end of the file, the document is closed with the ‘close’ method.

Creating and Writing Files

While there are multiple ways Java can create files, the FileOutPutStream class is the primary tool used in ITP 120. This class opens a pre-existing file or creates a new file when used.

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

The user will also need to import the PrintWriter class to write to the file. This class is used to take values and transfer them to the document.

The IOException is included in the program, unlike when reading files, and it will trigger if the file is unable to be opened.

To create a file, the programmer must first create an object for the FileOutputstream class and give it the name and extension value of a non-existing or preexisting file. In the example below, the object 'file' is assigned the value of the empty file Logbook.txt.

Console
```
File OutputStream file = new FileOutputSteam("Logbook.txt");
PrintWriter pen = new PrintWriter(file);

```

Once the file is linked to the object, the PrintWriter class will be assigned an object as well. This new object will take the file object created in the previous line as its parameter. A parameter acts as a placeholder variable that can hold different values during program execution.

In the example, when the object from the OutputStream class, “file,” is passed to the PrintWriter object, it is providing the name of the file that it is going to alter, “Logbook.txt.”
To write the contents of the file, the object ‘Pen’ from the PrintWriter class is called along with an output method, ‘print’ or ‘println,’ which handles line placement, and writes what is inside the parentheses to the file.

Console

```
File OutputStream file = new FileOutputSteam(“Logbook.txt”);
PrintWriter Pen = new PrintWriter(file);
    Pen.println(“Day 1.”);
    Pen.println(“Today starts the beginning of something new.”)
    Pen.close();
```

The “Pen” object is closed, and once the program is executed, the contents will be written to the file.

Example of Logbook:

Logbook.txt

```
Day 1

Today starts the beginning of something new.
```

The Academic Center for Excellence (ACE) offers free on-campus and online tutoring appointments for software design. For further assistance with programming concepts, please call ACE at (540) 891-3017 or email ACE@germanna.edu.