Revised: 8/23/2025

# Basic File I/O

Michael C. Hackett

Associate Professor, Computer Science

COMMUNITY COLLEGE OF PHILADELPHIA

# Lecture Topics

- File Basics
- File Objects
- Reading text files
- Writing text files
- Directories

# Colors/Fonts

- Local Variable Names – Brown
- Primitive data types – Fuchsia
- Literals – Blue
- Keywords – Orange
- Object names – Green
- Operators/Punctuation – Black
- Field Names – Lt Blue
- Method Names – Purple
- Parameter Names – Gold
- Comments – Gray
- Package Names – Pink

Source Code – **Consolas**
Output – Courier New

Boolean expression is false

Boolean expression is true

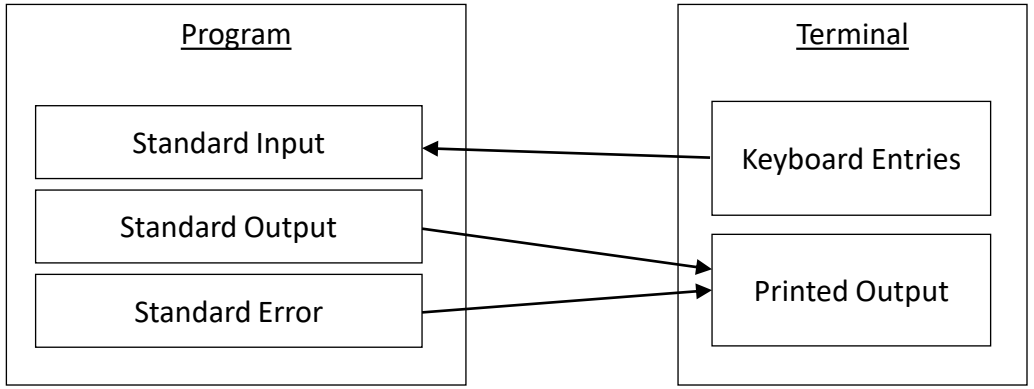
# File Basics

- A ***file*** is stream of binary, digital information typically kept on a long-term storage device.
  - Word documents, Powerpoint presentations, and PDFs are all examples of different types of files.

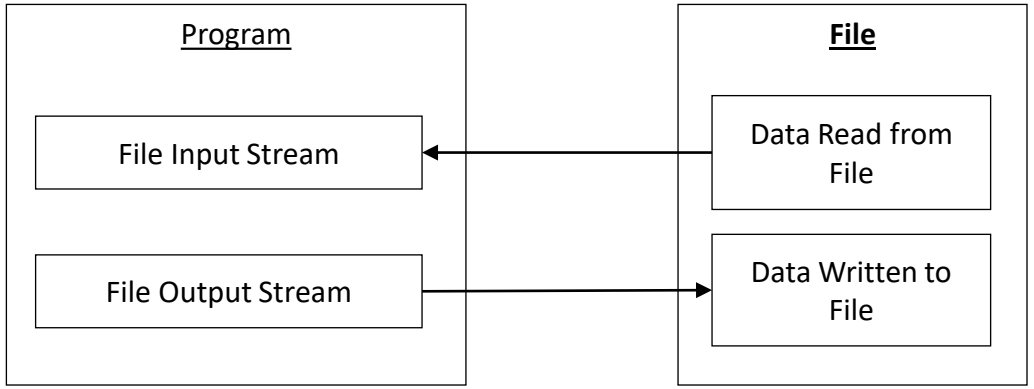
- Can be used as an input data stream. (“Reading a file”)

- Can be used as an output data stream. (“Writing to a file”)

# Standard Data Streams



# File Data Streams



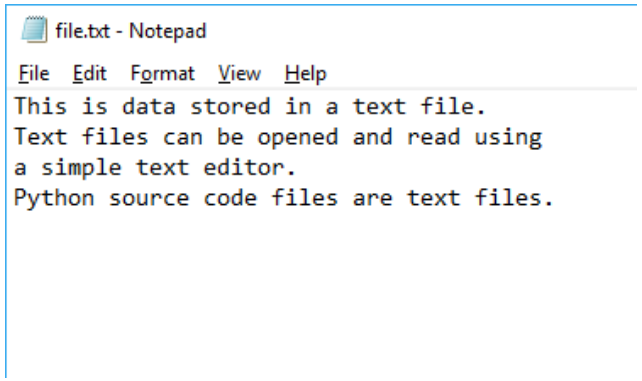
# File Basics - Extensions

- A file has a name which normally includes an extension.
  - Textfile**.txt**
  - WordDocument**.docx**

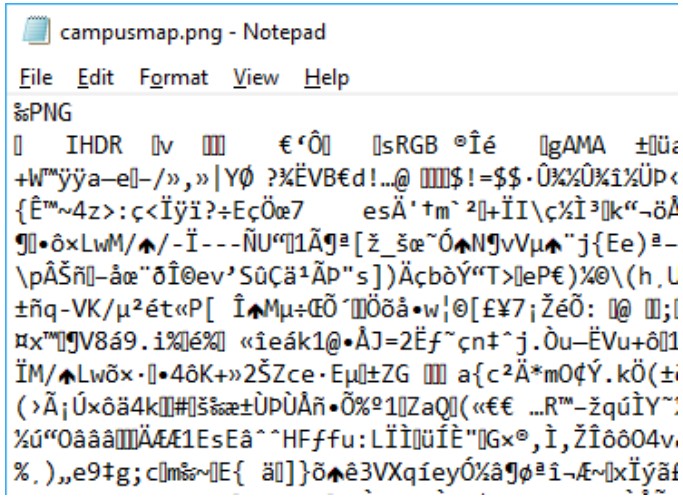
- You can have files without extensions.
  - Extensions are primarily used by the operating system, so it knows what program to use to open and read the file.
  - Some programs will only accept files with certain extensions.

# File Basics – File Types

- Text Files
  - The binary information contained in the file is encoded with ASCII plaintext.
  - Can be opened in any text editor (like Notepad.)
  - “Human readable”



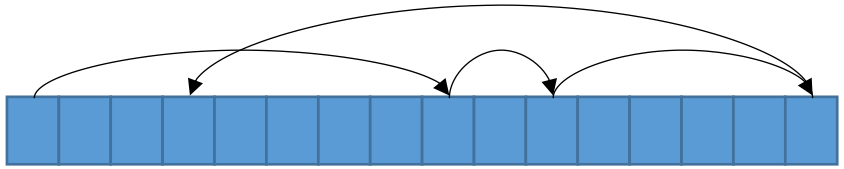

- Data Files
  - Files that are not stored in plaintext, like images and compiled programs.
  - Normally cannot be opened in any text editor.
  - Raw binary- “Computer readable”



# File Basics – File Access

- Using ***sequential access***, data is read/accessed from the beginning of the file through the end of the file.

- Using ***direct access***, data can be accessed from any location in the file.
  - A topic discussed in CSCI 112



# File Objects

- The File object provides a number of methods that gives us information about a file.

- It must be imported.

```
import java.io.File;
```

# File Objects

- There are a number of constructors for a File object, but we'll be using the constructor with one String parameter- the path to the file.

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
```

# Determining if a file exists

- To check if a file exists before you try to read or write to it, use the exists method.
  - This prevents FileNotFoundExceptions when trying to read a file that doesn't exist.
  - You can check to see if a file exists because you might not want to overwrite an existing file.

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
if(myTextFile.exists()) {
    //Do Stuff
}
```

- The exists() method returns true or false.

# Determining if a file is writable

- To check if permissions allow you to change/overwrite it, use the canWrite method.
  - This prevents IOExceptions when trying to change a file that you don't have permission to modify.

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
if(myTextFile.canWrite()) {
    //Do Stuff
}
```

- The canWrite() method returns true or false.

# Determining if a file is readable

- To check if permissions allow you to read the file, use the canRead method.
  - This prevents IOExceptions for trying to read a file that you don't have permission to read.

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
if(myTextFile.canRead()) {
    //Do Stuff
}
```

- The canRead() method returns true or false.

# Determining the file's name

- The File object keeps the entire path of the file.
  - To get the file's name, use the getName method.

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
System.out.println(myTextFile.getName());
```

```
file.txt
```

- The getName() method returns a String- the file name.

# Determining the file's path

- To get the directory path of the file, use the getParent method.

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
System.out.println(myTextFile.getParent());
```

```
C:\path\to\my
```

- The getParent() method returns a String- the file's directory path.

# Determining the file's full path

- To get the full file path, use the getPath() method.

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
System.out.println(myTextFile.getPath());
```

```
C:\path\to\my\file.txt
```

- The getPath() method returns a String- the full path, including name.

# Determining the file's full path

- If the file is in the same directory as the compiled class file, you don't need to specify the file's path for Java to find it.

```
File myTextFile = new File("file.txt");
System.out.println(myTextFile.getPath());
```

```
file.txt
```

- However, the file's path will only be the file name.

# Determining the file's absolute path

- The File object's getAbsolutePath method will always return the full path to the file, even if we only gave the File object the file's name.

```
File myTextFile = new File("file.txt");
System.out.println(myTextFile.getAbsolutePath());
```

```
C:\path\to\my\file.txt
```

# Reading files in Java using a Scanner

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
Scanner fileReader = new Scanner(myTextFile);
```

- Pass your file to a new Scanner object.
  - This is the same Scanner object we have been using to get keyboard input.
  - Now, we are using the file as the input source instead of System.in

# Reading lines from a file

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
Scanner fileReader = new Scanner(myTextFile);
String line = fileReader.nextLine();
```

- The nextLine method will return the next line of the file as a String.
  - In this example, it would return line 1 from file.txt, since this is the first time we called the nextLine method.
- The example only reads a single line. How can we read through an entire file with an unknown number of lines?

# Reading the contents of an entire file

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
Scanner fileReader = new Scanner(myTextFile);

while(fileReader.hasNextLine() == true) {
    System.out.println(fileReader.nextLine());
}
```

- The Scanner's hasNextLine method returns true if there are more lines to be read and false if it reached the end of the file.
- The above while loop will iterate as long as there are still lines to be read.

# Closing the file

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
Scanner fileReader = new Scanner(myTextFile);

while(fileReader.hasNextLine() == true) {
    System.out.println(fileReader.nextLine());
}

fileReader.close();
```

- The Scanner's close method releases it's hold on the file.
- Only call this method when you are done using the resource/file.

# Exception Handling for Files

- Exception handling isn't covered until CSCI 112.
- Java has two types of exceptions, checked and unchecked.
- An ***unchecked exception*** is an exception that the compiler does not check for.
  - Executing the code below will result in an ArrayIndexOutOfBoundsException.

```
char[] letters = {'a', 'b', 'c'};
System.out.println(intArray[3]);
```

  - The compiler doesn't check to see if the index used is valid, yet the code will still compile.
  - This is an example of an unchecked exception.

# Exception Handling for Files

- A ***checked exception*** is an exception that the compiler *does* check for.
  - Checked exceptions have to be *handled* or the source code won't compile.

- Checked exceptions are caused by a method or constructor explicitly stating in their header that they may throw an exception.

- As previously stated, we won't be covering exceptions and exception handling in detail.
  - However, we will now need to (at least) handle checked exceptions.

# Exception Handling for Files

- The Scanner constructor that takes a File object as the parameter is one such constructor that explicitly says it may throw an exception.
  - Specifically, a FileNotFoundException (meaning the file doesn't exist.)

- To handle **any** checked exceptions in your main method, put the clause "throws Exception" on the end of the main method header.

```
public static void main(String[] args) throws Exception {
```

# Writing to files in Java

- The PrintWriter object provides an easy way to write data to a new or existing file.

- The PrintWriter object must be imported.

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

# Writing to files in Java

```
public static void main(String[] args) throws Exception {

  File myTextFile = new File("C:\\path\\to\\my\\file.txt");
  PrintWriter fileWriter = new PrintWriter(myTextFile);

}
```

- The above example initializes a new PrintWriter object with a File object.
  - The file the PrintWriter will write to.
- Like the Scanner constructor, this PrintWriter constructor may also throw a FileNotFoundException.
  - Remember to include the “throws Exception” clause.
  - If the file doesn’t exist, the PrintWriter will create it.
    - You would most likely get a FileNotFoundException because the file or directory wasn’t writable.

# Writing to files in Java

```
public static void main(String[] args) throws Exception {

  File myTextFile = new File("C:\\path\\to\\my\\file.txt");
  PrintWriter fileWriter = new PrintWriter(myTextFile);

}
```

- If the file already exists, it will be completely overwritten.
  - There will be an example of how to append to files later in the lecture.

# Writing to files in Java

```
public static void main(String[] args) throws Exception {

  File myTextFile = new File("C:\\path\\to\\my\\file.txt");
  PrintWriter fileWriter = new PrintWriter(myTextFile);
  fileWriter.println("This will be printed to my file.");

}
```

- The println, print, and printf methods work in an identical fashion to System.out.println/print/printf.
  - The “out” in System.out **is** a PrintWriter! It just prints to the console instead of a file.

# Closing the file

```
public static void main(String[] args) throws Exception {

  File myTextFile = new File("C:\\path\\to\\my\\file.txt");
  PrintWriter fileWriter = new PrintWriter(myTextFile);
  fileWriter.println("This will be printed to my file.");
  fileWriter.print("This will also be printed to my file.");
  fileWriter.println("And so will this.");
  fileWriter.close();

}
```

- If you do not close your PrintWriter after you are finished writing to the file, your changes will **not** be made permanent and the file will be empty.

# Appending to files in Java

```
import java.io.BufferedWriter;
import java.io.FileWriter;
```

```
public static void main(String[] args) throws Exception {
     PrintWriter fileWriter = new PrintWriter(new BufferedWriter(new FileWriter("SampleOutputFile.txt", true)));
     fileWriter.println("This will be printed to my file.");
     fileWriter.close();
}
```

- A few more steps than using just PrintWriter and File objects.
- The above will append the line “This will be printed to my file.” to the end of SampleOutputFile.txt instead of overwriting existing data.
- This is just for your reference. All assignments will have you create new files/overwrite old ones.

# Directories

- A ***directory*** (or folder) contains files and other directories (or subdirectories.)
- The File object can be a file or a directory.
- Both examples below are valid.

```
File myTextFile = new File("C:\\path\\to\\my\\file.txt");
File myDirectory = new File("C:\\path\\to\\my");
```

# Determining if a File object is a directory

- To determine if a File object is a directory, use the isDirectory method.
  - You'll only need to use this if you're not sure whether or not a File object is a directory.

```
File myDirectory = new File("C:\\path\\to\\my");
if(myDirectory.isDirectory()) {
    //Do Stuff
}
```

- The isDirectory method returns true if the File Object is a directory or false if the File object is a file.
- Alternatively, you can also use the isFile method, which returns true if the File object is a file or false if the File object is a directory.

# Directory Permissions and Names/Paths

- All of the File object methods seen earlier in the lecture can be used for directories.
  - exists method
  - canRead method
  - canWrite method
  - getName method
  - getParent method
  - getPath method
  - getAbsolutePath method

# Getting a list of files in a directory

- The listFiles method returns an array of File objects.
  - These File objects are only the files and directories contained in the directory.

```
File myDirectory = new File("C:\\path\\to\\my");
if(myDirectory.isDirectory()) {
  File[] contents = myDirectory.listFiles();
}
```

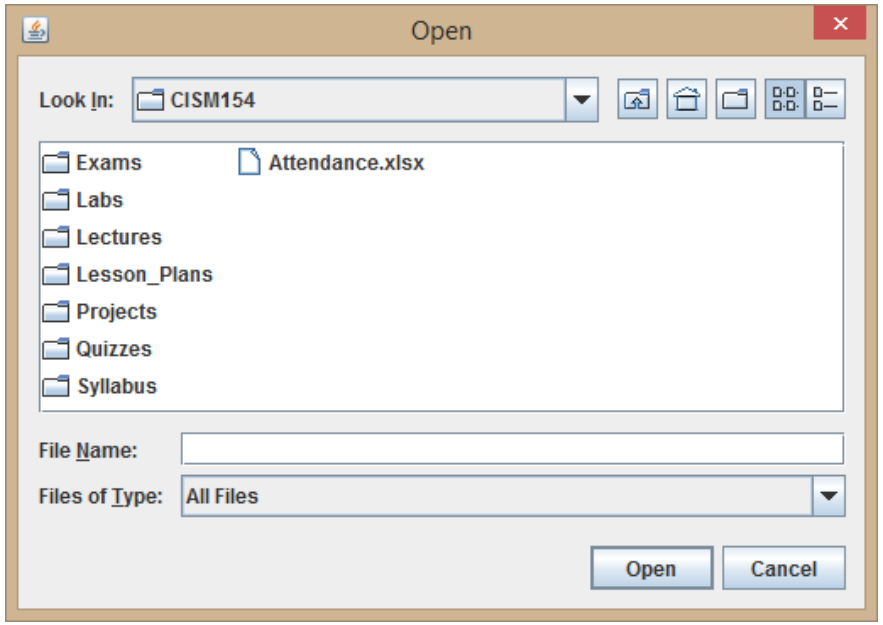
# Getting a list of the files and directories in a directory

- The list method returns an array of Strings.
  - These Strings are the **names** of the files and directories contained in the directory.

```
File myDirectory = new File("C:\\path\\to\\my");
if(myDirectory.isDirectory()) {
  String[] contents = myDirectory.list();
}
```

# JFileChooser

- A JFileChooser is a graphical interface for navigating a file system to select a file or directory.



# Creating an instance of a JFileChooser

```
import javax.swing.JFileChooser;
```

A new instance named “exampleChooser”

```
JFileChooser exampleChooser = new JFileChooser();

int result = exampleChooser.showOpenDialog(null);
```

The chooser window will return an int value; The value depends on whether the user clicked OK or Cancel. More about that on the next slide.

showOpenDialog method handles displaying the chooser window.

It requires one parameter, but you can set it to null. The parameter refers to what other graphical component the chooser window will be displayed relative to. “null” will just force it to display in the middle of the screen.

# Getting the JFileChooser's selected file

```
JFileChooser exampleChooser = new JFileChooser();
int result = exampleChooser.showOpenDialog(null);

File selectedFile = null;
if (result == JFileChooser.APPROVE_OPTION) {
    selectedFile = exampleChooser.getSelectedFile();
}
```

Declare a File object that we can use for the file the user selected.

We can store the File object returned by the getSelectedFile method into the File object we previously declared.

getSelectedFile method returns the selected file, as a File object.

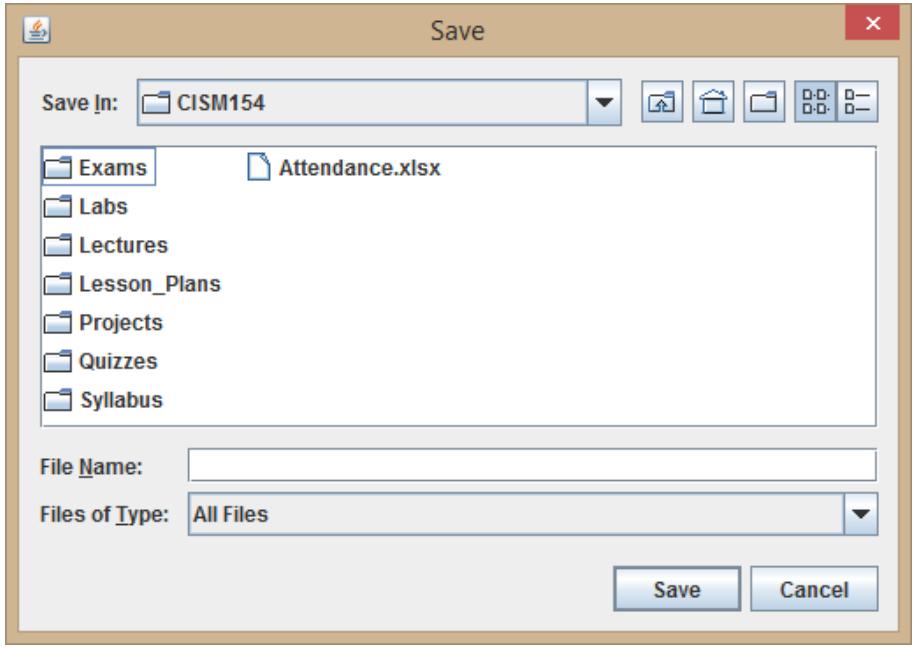
APPROVE_OPTION is a constant field of the JFileChooser object.

The value of this constant is the int value for when the user selected a file and pressed Open in the Chooser window.

We compare this constant value to the value we received from the showOpenDialog method call. If they match, then we know the user selected a file.

# JFileChooser – Saving Files

- JFileChoosers can also be used for selecting a location to save a file, just as it is used for opening files.



# JFileChooser – Saving Files

```
JFileChooser exampleChooser = new JFileChooser();
int result = exampleChooser.showSaveDialog(null);

File selectedFile = null;
if (result == JFileChooser.APPROVE_OPTION) {
    selectedFile = exampleChooser.getSelectedFile();
}
```

- Works identically as opening files.
- You just call the showSaveDialog method instead of the showOpenDialog method.