Java System: Files

CPSC 233: Introduction to Computer Science for Computer Science Majors II
Winter 2022

Jonathan Hudson, Ph.D.
Instructor
Department of Computer Science
University of Calgary

Monday, 8 November 2021

UNIVERSITY OF CALGARY

Files – Are Objects in Python

Once made like the following we can now interact with the file (import java.io.File;)

```
File file = new File(filename);
```

https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/File.html

Creating a file

- Working file reading program

```
public static void main(String[] args) {
    if (args.length != 1) {
        System.err.println("Invalid number of arguments!");
        System.exit(1);
    } else {
        String filename = args[0];
        File file = new File(filename);
    }
}
```

Files – Are Objects in Python

Once made like the following we can now interact with the file (import java.io.File;)

```
File file = new File(filename);
```

```
System.out.println(file.getName());          //Name of file as provided
System.out.println(file.getPath());          //Path of file as provided
System.out.println(file.getAbsoluteFile());    //Name of file on OS
System.out.println(file.getAbsolutePath());    //Path of file on OS
System.out.println(file.isFile());           //Is this a file
System.out.println(file.isDirectory());       //Is this a directory
System.out.println(file.exists());           //Does this file exist
System.out.println(file.length());           //Size of file
System.out.println(file.canRead());          //Can file be opened for reading
System.out.println(file.canWrite());         //Can file be opened for writing
```

Files – Are Objects in Python

input.txt entered but does not exist

```
File file = new File(filename);
```

```
System.out.println(file.getName());
System.out.println(file.getPath());
System.out.println(file.getAbsoluteFile());
System.out.println(file.getAbsolutePath());
System.out.println(file.isFile());
System.out.println(file.isDirectory());
System.out.println(file.exists());
System.out.println(file.length());
System.out.println(file.canRead());
System.out.println(file.canWrite());
```

```
input.txt
input.txt
C:\Users\jonat\IdeaProjects\Example\input.txt
C:\Users\jonat\IdeaProjects\Example\input.txt
false
false
false
0
false
false
```

Files – Are Objects in Python

input.txt entered and exists!

```
File file = new File(filename);
```

```
System.out.println(file.getName());
System.out.println(file.getPath());
System.out.println(file.getAbsoluteFile());
System.out.println(file.getAbsolutePath());
System.out.println(file.isFile());
System.out.println(file.isDirectory());
System.out.println(file.exists());
System.out.println(file.length());
System.out.println(file.canRead());
System.out.println(file.canWrite());
```

```
input.txt
input.txt
C:\Users\jonat\IdeaProjects\Example\input.txt
C:\Users\jonat\IdeaProjects\Example\input.txt
true
false
true
18
true
true
```

Files – Are Objects in Python

local directory "." entered

```
File file = new File(filename);
```

```
System.out.println(file.getName());
System.out.println(file.getPath());
System.out.println(file.getAbsoluteFile());
System.out.println(file.getAbsolutePath());
System.out.println(file.isFile());
System.out.println(file.isDirectory());
System.out.println(file.exists());
System.out.println(file.length());
System.out.println(file.canRead());
System.out.println(file.canWrite());
```

```
.
.
C:\Users\jonat\IdeaProjects\Example\.
C:\Users\jonat\IdeaProjects\Example\.
false
true
true
4096
true
true
```

Checking properties of file before reading

```
public static void main(String[] args) {
    if (args.length != 1) {
        System.err.println("Invalid number of arguments!");
        System.exit(1);
    } else {
        String filename = args[0];
        File file = new File(filename);
        if (file.exists() && file.isFile() && file.canRead()){

        }else{
            System.err.println("Cannot access the file to read it!")
            System.exit(1);
        }
    }
}
```

Reading from a File

Files - Reading From a File – Treat file like user input

- Scanner scanner = new Scanner(file);

- Everything that worked earlier, will work on this file, except that once we run out of data our request for input will give use an error!

Files - Reading From a File

- FileReader file_reader = new FileReader(file);
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/FileReader.html
- File readers let us request input from the created file handle

Files - Reading From a File

```
public static void main(String[] args) {
    if (args.length != 1) {
        System.err.println("Invalid number of arguments!");
        System.exit(1);
    } else {
        String filename = args[0];
        File file = new File(filename);
        if (file.exists() && file.isFile() && file.canRead()){
            FileReader file_reader = new FileReader(file);
        }else{
            System.err.println("Cannot access the file to read it!");
            System.exit(1);
        }
    }
}
```

Files - Reading From a File (need to handle exception)

```
public static void main(String[] args) {
    if (args.length != 1) {
        System.err.println("Invalid number of arguments!");
        System.exit(1);
    } else {
        String filename = args[0];
        File file = new File(filename);
        if (file.exists() && file.isFile() && file.canRead()){
            FileReader file_reader = new FileReader(file);
        }else{
            System.err.println("Cannot access the file to read it!")
            System.exit(1);
        }
    }
}
```

Files - Reading From a File (need to handle exception)

```
public static void main(String[] args) {
  …
        try {
           FileReader file_reader = new FileReader(file);
        } catch (FileNotFoundException e) {
            //Handle error (More on this later)
        }
  …
}
```

Files - Reading From a File – Want text functionality

- BufferedReader buffered_reader = new BufferedReader(file_reader);
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/BufferedReader.html
- BufferedReaders allow us to do a lot more with our input

Files - Reading From a File

```
try {
    FileReader file_reader = new FileReader(file);
    BufferedReader buffered_reader = new BufferedReader(file_reader);
} catch (FileNotFoundException e) {
    //Handle error
}
```

Files - Reading From a File – Loop Through Lines

```
try {
    FileReader file_reader = new FileReader(file);
    BufferedReader buffered_reader = new BufferedReader(file_reader);
    String line = buffered_reader.readLine();
    while (line != null) {
        System.out.println(line);
        line = buffered_reader.readLine();
    }
} catch (FileNotFoundException e) {
    //Handle error
}
```

Files - Reading From a File - IOException

```
try {
    FileReader file_reader = new FileReader(file);
    BufferedReader buffered_reader = new BufferedReader(file_reader);
    String line = buffered_reader.readLine();
    while (line != null) {
        System.out.println(line);
        line = buffered_reader.readLine();
    }
} catch (FileNotFoundException e) {
    //Handle error
}
```

Files - Reading From a File – deal with IOException

```
try {
    FileReader file_reader = new FileReader(file);
    BufferedReader buffered_reader = new BufferedReader(file_reader);
    String line = buffered_reader.readLine();
    while (line != null) {
        System.out.println(line);
        line = buffered_reader.readLine();
    }
} catch (FileNotFoundException e) {
    //Handle error
} catch (IOException e) {
    //Handle other error type
}
```

Files - Reading From a File – Scanner Style

```
try {
    Scanner scanner = new Scanner(file);
    while (scanner.hasNextLine()) {
        String line = scanner.nextLine();
        System.out.println(line);
    }
} catch (FileNotFoundException e) {
    //Handle error
}
```

Scanner versus BufferedReader

- Scanner will only read file each time you make request
  - Less memory stored at one time
  - Slower because it makes I/O request to hard drive each loop (farther from CPU)

- BufferedReader
  - Will pre-load file into memory (as much as VM lets it), more memory
  - Faster because requests are now in RAM (random-access memory closer to CPU)

Writing to a File

Checking properties of file before writing

```
public static void main(String[] args) {
  …
     if(!file.exists()){
        try {
           file.createNewFile();
        } catch (IOException e) {
           //Handle error
        }
     }
     if (file.exists() && file.isFile() && file.canWrite()) {

     } else {
        System.out.println("Cannot access the file to write it!");
     }
  …
}
```

Files - Writing To a File

- FileWriter file_writer = new FileWriter(file);
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/FileWriter.html
- File writers let us send output from the created file handle

Files - Reading From a File – Want buffer functionality

- BufferedWriter buffered_writer = new BufferedWriter(file_writer);
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/BufferedWriter.html
- BufferedWriters allow us to do more with our output

Files - Writing To a File (need to handle exception)

```
if (file.exists() && file.isFile() && file.canWrite()) {
    try {
        FileWriter file_writer = new FileWriter(file);
    } catch (IOException e) {
        //Handle file open error
    }
} else {
    System.out.println("Cannot access the file to write it!");
}
```

Files - Writing To a File (write)

```
if (file.exists() && file.isFile() && file.canWrite()) {
    try {
        FileWriter file_writer = new FileWriter(file);
        BufferedWriter buffered_writer = new BufferedWriter(file_writer);
        for (int i = 0; i < 10; i++) {
            buffered_writer.write(i + "\n");
        }
    } catch (IOException e) {
        //Handle write error
    }
} else {
    System.err.println("Cannot access the file to write it!");
    System.exit(1);
}
```

Files - Writing To a File (write and flush!!!!)

```
if (file.exists() && file.isFile() && file.canWrite()) {
    try {
        FileWriter file_writer = new FileWriter(file);
        BufferedWriter buffered_writer = new BufferedWriter(file_writer);
        for (int i = 0; i < 10; i++) {
            buffered_writer.write(i + "\n");
        }
        buffered_writer.flush();       //makes sure your text reaches file
    } catch (IOException e) {
        //Handle write error
    }
} else {
    System.err.println("Cannot access the file to write it!");
    System.exit(1);
}
```

Files - Writing To a File (PrintWriter)

```
if (file.exists() && file.isFile() && file.canWrite()) {
    try {
        FileWriter file_writer = new FileWriter(file);
        PrintWriter print_writer = new PrintWriter(file_writer);
        for (int i = 0; i < 10; i++) {
            print_writer.println(i);
        }
        print_writer.flush();
    } catch (IOException e) {
        //Handle write error
    }
} else {
    System.err.println("Cannot access the file to write it!");
    System.exit(1);
}
```

PrintWriter versus BufferedWriter

- PrintWriter is essentially a BufferedWriter with a lot more useful commands
- https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/io/PrintWriter.html
- println, print commands for quite a number of types int,char,bool,String, etc.
- For text many prefer to use Scanner for file input, PrintWriter for file output

Closing a File

Files - Closing a File

- **If you are writing to a file:** it is important to know that writing does not occur instantly. Java maintains a buffer where it stores the strings that are ought to be written to the file. Java and the OS coordinate the best time to **flush** this data and push it into the actual file area in memory.
- If your program crashes or terminates without closing a file, the **flush** action may not have occurred; Java will close files but may not flush buffers when terminating normally.

Files - Closing a File

- Also, opening files, reading from them and writing to them consume memory. Having unnecessarily-opened files means some memory is being used when it can be freed.

- CLOSE YOUR FILES!!!

- You can request a flush (does not immediately occur, but will happen before exit)
  buffered_writer.flush()
  print_writer.flush()
- Or a close (does not force flush, closing the file_reader, file_writer, will close these)
  buffered_reader.close()
  buffered_writer.close()
  print_writer.close()

Standard Input/Output/Error

Files - Standard Input, Output and Error

- We have been using files since the first program that we wrote.
- When you run a Java program, the interpreter opens up three streams, standard output, input, and error. When a program ends, it closes these streams.
- These streams are used for display output, and accepting inputs from/to Java programs.

Files - Standard Input, Output and Error

- **System.out**: is the stream for the Standard **Out**put
  - It is opened and closed automatically when the program starts and ends.
  - Any values written to it are displayed on the screen.
  - The stream is accessible via **System.out**
- **System.in**: is the stream for the Standard **In**put.
  - It is opened and closed automatically when the program starts and ends.
  - The file handler is accessible via **System.in**
- **System.err**: is the stream for the Standard **Err**or.
  - It is opened and closed automatically when the program starts and ends.
  - Any values written to it are displayed on the screen.
  - Intended for displaying error messages instead of program output
  - Allows us to redirect program output separately from error messages
  - Useful for debugging
  - The file handler is accessible via **System.err**

Onward to … Exceptions.

Jonathan Hudson
jwhudson@ucalgary.ca
https://pages.cpsc.ucalgary.ca/~jwhudson/

UNIVERSITY OF CALGARY