WEEK NINE

Acknowledgements: Slides created based off material provided by Dr. Travis Doom

FILES

- Sequence of binary digits
  - May represent integers, text characters, etc.
- Files have many different types that define how to read the information inside
  - Text file: ASCII/UNICODE characters
  - Binary file: pretty much everything else

FILE I/O AND THE OS

- Operating System (OS) handles file operations for programs
  - Interacts with the storage device
  - Polices who can access/write to a file
  - Handles file properties (size, permissions, name)
- OS must open files so a program can use them
  - Programs use method calls to invoke OS routines
    - Create and open a new file to write to it (output)
    - Open an existing file to read it (input)
    - Open an existing file and write/append information to it (output)
    - Destroy an existing file
  - If the OS runs into a problem, it throws (creates) an exception

EXCEPTIONS

- Describes a problem that occurs in the code (or in this case with the OS)
- This allows the program to respond accordingly to an unexpected issue
- Some exceptions, (particularly file I/O exceptions) are **checked exceptions**
  - We must deal with these in some way, otherwise we will get an error
  - EX: `IOException`, `FileNotFoundException`
- When we encounter an exception, we must either:
  - Handle the exception (try/catch: we will discuss this later) OR
  - Pass the exception up a level
- To pass it up a level, we need to add a throw clause to the method header

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

FILE BUFFERS

- Program calls a method to ask the OS to open the file
    - OS creates an area in memory (a buffer) that the program can access
    - OS provides the program with a reference to the buffer (a file handle)
    - OS checks if the program is permitted to receive the file handle
        - Permissions, is the file already open?
- Buffer improves performance
    - Memory is faster than accessing storage device where the file is stored
    - If file is opened, OS copies file contents into buffer
    - If file write occurs, change occurs in buffer
    - Eventually, OS will copy buffer back into the file
    - OS will flush the buffer and close the file
    - Open files are closed when program exits, thus we need to explicitly close files that are open

FILE POINTERS

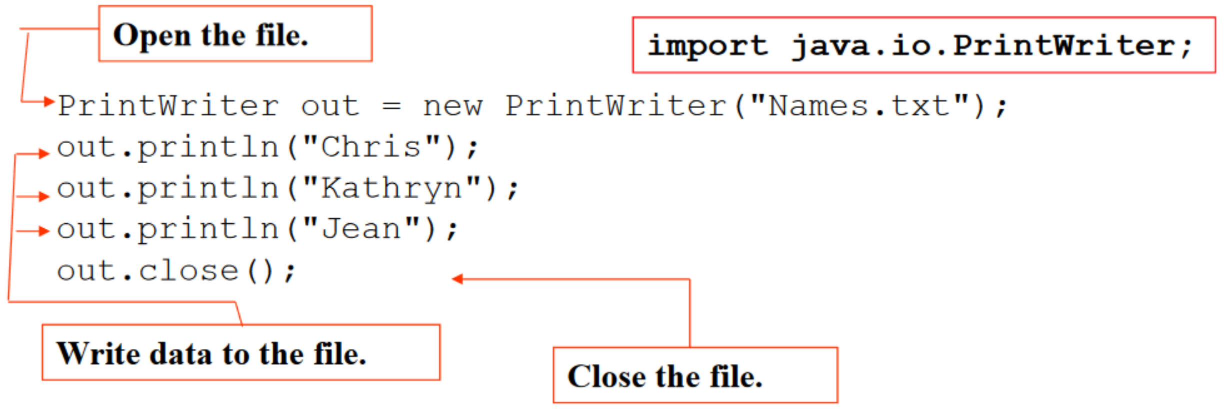
- File pointer indicates where the next read or write operation will take place
- Each file is treated as a one-dimensional sequence of characters
  - Non-printing characters for newlines
- Read position indicates what characters are returned on the next read operation
  - Read position is updated to the next character each time a character is read
  - Most languages also provide methods to move the read pointer
  - EX: ‘1501 245’
  - nextInt() would read in ‘1501’
  - Read pointer is now on the white space between the two numbers
  - nextLine() would read in ‘ 245’

FILENAME CONVENTIONS

- Dependent on the OS
- Two different ways to reference file locations
  - Relative reference: specified from a default working directory (no absolute path)
    - `String filename = "Data.txt";`
  - Absolute reference: entire path specified from the root directory
    - `String filename = "C:\\Users\\ClarissaMilligan\\Documents\\FA24\\data.txt";`
    - Since '\' is the escape character in ASCII/UNICODE, we must use '\\'
    - Unix-type OSs use forward slash '/'

THE PRINTWRITER CLASS

- Under the java.io library
- PrintWriter class allows for writing to files using `print` and `println` methods, like we use for the console
- Constructor takes in a filename (`String`) or file handle (`FileWriter`)
    - WARNING: If PrintWriter is just given a filename, it will always overwrite that file completely



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

```
PrintWriter out = new PrintWriter("Names.txt");
out.println("Chris");
out.println("Kathryn");
out.println("Jean");
out.close();
```

THE FILEWRITER CLASS

- Also, in the java.io library
- FileWriter is used to avoid erasing an existing file
  - `FileWriter fwriter = new FileWriter(“filename.txt”, true);`
  - Boolean argument indicates whether or not we will be appending data to the file
  - If we choose true, buffer will be created in a way so output will be appended to the end of the file
- FileWriter object can be passed into a PrintWriter

THE FILE CLASS

- Also in the java.io library
- Used to create a file handle
  - `File fileHandle = new File(“filename.txt”);`
- Scanner object can be used to parse the associated buffer (read the contents)
  - `Scanner inputFile = new Scanner(fileHandle);`
- Data can then be read with the same methods we use for the console:
  - `nextLine(), nextInt(), nextDouble(),` etc.
  - `hasNextLine(), hasNextInt(), hasNextDouble(),` etc.

FILE CLASS METHODS

- Sanity checking methods
  - `exists(), canWrite(), canRead(), canExecute()`
- Other useful methods
  - `createNewFile(), delete(), getPath(), getAbsolutePath()`
- Set OS properties
  - `setExecutable(), setReadable(), setWritable(), setLastModificationDate()`

HANDLING EXCEPTIONS

- Using a try/catch block allows us to handle issues without our program crashes
- Try block
  - Should surround any of the code that could generate an exception
  - Any code that is dependent on the code above also needs to be in the try block
  - Once an exception is thrown from the try block the execution halts and jumps to the corresponding catch block
- Catch block
  - Can be more than one for different exceptions
- Finally block
  - Executes last regardless of whether an exception is thrown

TRY/CATCH/FINALLY

```
try {
      // code that can cause an exception
} catch (Exception e) {
      // code that happens if that exception occurs
} finally {
      // code that happens regardless of what happens above
}
```

ACTIVITY

- Write code that:
    - Reads in 5 numbers from a file called “nums.txt”
    - Stores them in an array
    - Prints out the sum of the numbers
- File will look like this:

```
4
23
17
16.5
-8.3
```

FILEOUTPUTSTREAM

- Also, part of java.io
- FileOutputStream can be used in place of a FileWriter

```
FileOutputStream fileStream = new FileOutputStream("data.txt‶, true);
PrintWriter fileOut = new PrintWriter(fileStream);

fileOut.println("Hello File!");
fileOut.close();
fileStream.close();
```

FILEOUTPUTSTREAM

- Also, part of java.io
- FileInputStream can be used in place of File

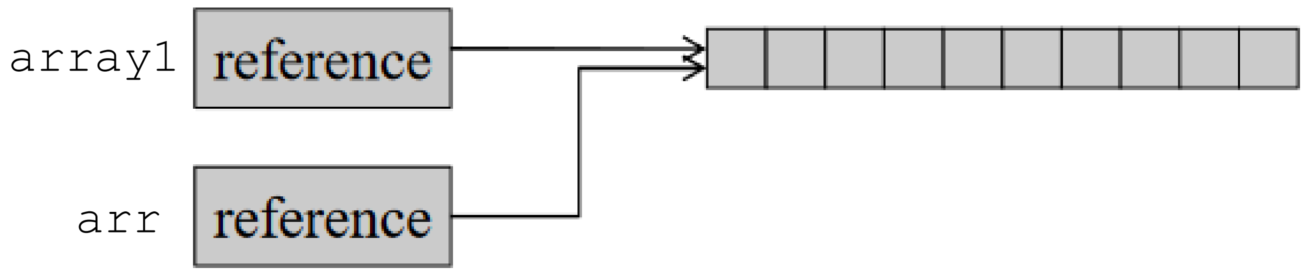
```
FileInputStream fileByteStream = new FileInputStream("data.txt");
Scanner fileInput = new Scanner(fileByteStream);

while (fileInput.hasNextLine()) {
       System.out.println(fileInput.nextLine());
}
fileInput.close();
fileByteStream.close();
```

MORE ON ARRAYS

- Remember, arrays are objects
- Variable name points to a memory address where the array is stored
- To print an array,
  - Loop through the array and print each index
  - Use `Arrays.toString(array1);`
- To test equality of two arrays,
  - Loop through one array and check equality index by index
  - Use `Arrays.equals(array1, array2);`

PASSING ARRAYS AS ARGUMENTS

- Applies to any object not just arrays
- Objects are passed by reference in Java
- Thus, they can be modified in a method and the actual object being passed in will also be modified
- This is different from when we pass primitive types into a method (pass by value)
- `array1` is the name in main
- `arr` is the argument name



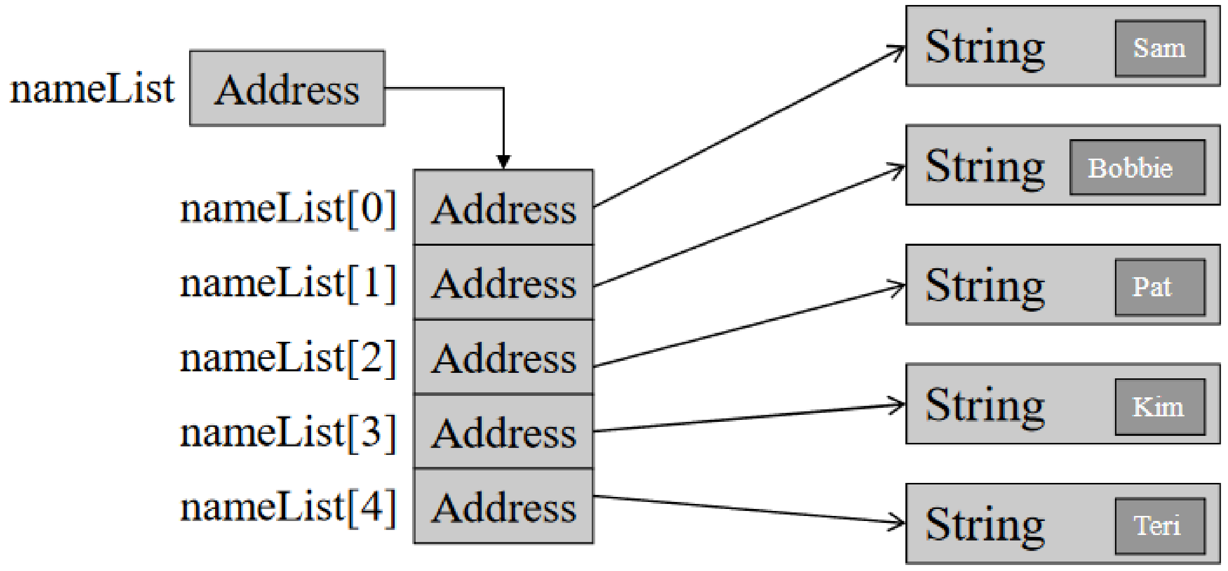
PASSING ARRAYS AS ARGUMENTS

```
public static void main(String[] args) throws IOException {
        int[] array1 = {1, 2, 3};
        zeroArray(array1);
        System.out.println(Arrays.toString(array1));
    }

    public static void zeroArray (int[] arr) {
        for (int i = 0; i < arr.length; i++) {
            arr[i] = 0;
        }
    }
```

ARRAYS OF OBJECTS

```
String[] nameList = {“Sam”, “Bobbie”, “Pat”, “Kim”, “Teri”};
```

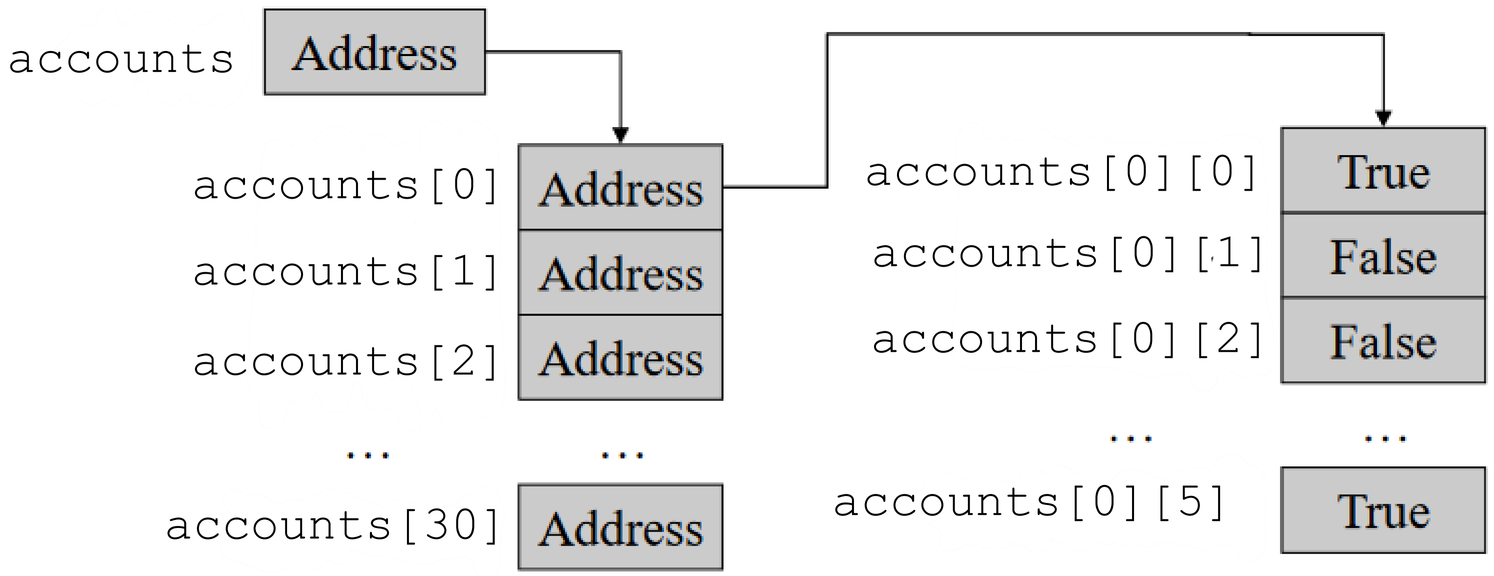
- Since String is an object, each index in the array holds the memory address of the object
- Essentially, we have an array of addresses



MULTI-DIMENSIONAL ARRAYS

- An array can also contain another array (which could contain another array and so on…)

```
double[][] accounts = new double [NUM_ACC][NUM_DEPOSITS];
```



accounts
Address
accounts[0]
Address
accounts[1]
Address
accounts[2]
Address
…
…
accounts[30]
Address
accounts[0][0]
True
accounts[0][1]
False
accounts[0][2]
False
…
…
accounts[0][5]
True