WEEK EIGHT

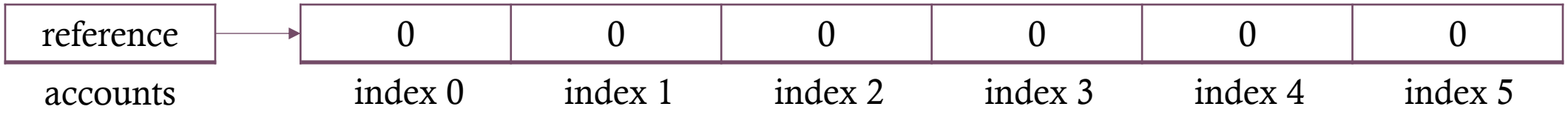
Acknowledgements: Slides created based off material provided by Dr. Travis Doom

THE ARRAY

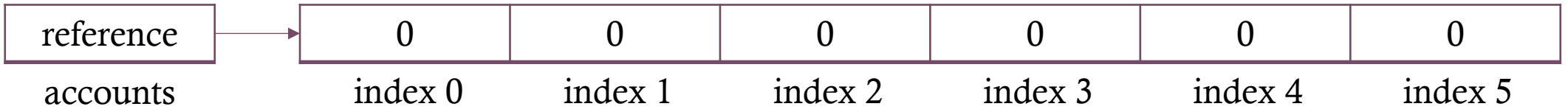
- Data structure
  - Contain groups of related items under one variable name
- Arrays
  - Simplest and most prevalent data structure
  - Object that contains items of the same data type
  - Each item is indexed by their order in the list (starting at 0)
  - Can hold primitive data types or objects
- String is essentially an array of characters

CREATING AN ARRAY

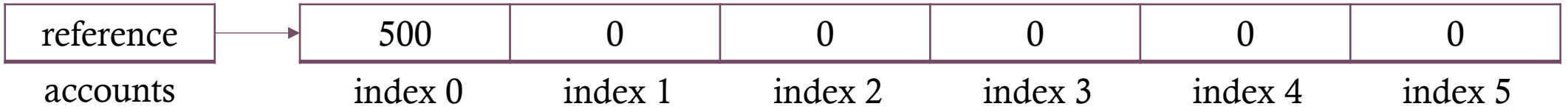
- An array is an object thus it needs an object **reference**
  - The reference is stored in a variable and refers to the place in memory that the object is stored
  - `int[] accounts;`
- When creating an array, we must define it with a permanent size
  - We can never directly change the size of this array after it is created
  - `accounts = new int[6];`
  - `int[] accounts = new int[6];`



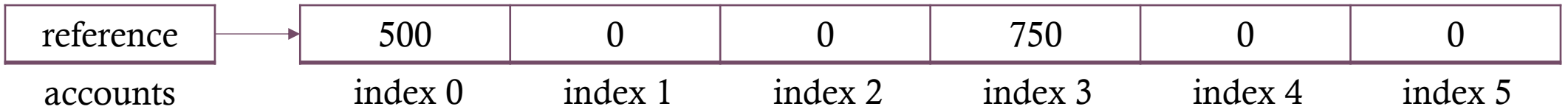
ACCESSING AND MODIFYING ARRAYS



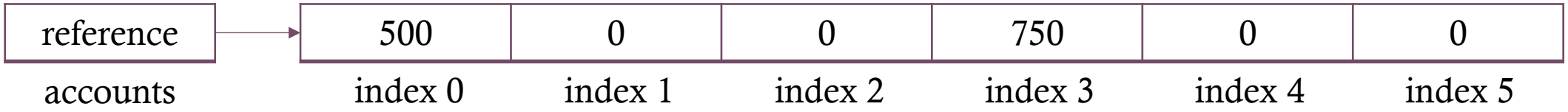
- Say we want to update the value of the first index
  - `accounts[0] = 500;`



- We can also reference an existing array value when modifying another
  - `accounts[3] = accounts[0] + 250;`



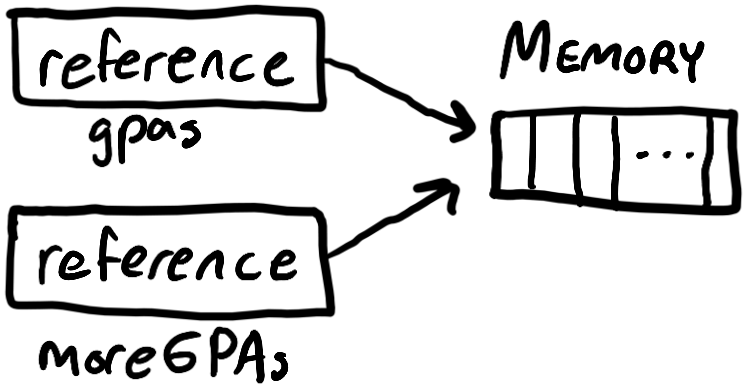
MORE ABOUT ACCESSING ARRAYS



- What happens if we try:
  - `int num = accounts[6];`
  - ArrayIndexOutOfBoundsException
- What if we try:
  - `int index = 3;`
  - `int value = accounts[index];`
  - value will equal 750

CREATING AN ARRAY WITH DEFAULT VALUES

- If you want your array to have some default values other than zero,
  - `double[] gpas = {2.7, 3.4, 4.0, 3.6};`
  - `gpas[2]` is equal to `4.0`
- Remember, arrays are objects
  - What happens if we do:
  - `System.out.println(gpas);`
  - `[D@7b23ec81`
  - What if we do:
  - `double[] moreGPAs = gpas;`
  - `moreGPAs` now referenes the same place in memory as `gpas`
  - If one changes, they both change



ADDITIONAL ARRAY FUNCTIONALITY

- `String[] weekDays = {“Mon”, “Tue”, “Wed”, “Thu”, “Fri”, “Sat”, “Sun”};`
- Because arrays are objects, they have some built in fields and methods
  - The length *field*:
    - `int size = weekDays.length; // 7`
  - Useful methods:
    - `Arrays.toString();`
    - `Arrays.equals();`
    - `Arrays.sort();`
    - `weekDays.clone();`
- Array objects have access to all the methods of that object
  - `String allCapsMon = weekDays[0].toUpperCase();`

ACTIVITY

- Write a method that uses an array to keep track of a certain number of doubles
- The method will be provided with a starting value, and a number of doubles
- The method should then store each double in an index in the array and then return the array
- For example,
  - If the method is given 5 as a starting value and 4 as the number of doubles,
  - The array should look like this: [5, 10, 20, 40]

FOR-EACH LOOPS

- Enhanced for-loops for arrays or array-like structures
- Simplify code

```
int[] ages = new int[15];
for (int age : ages){
      System.out.println(age);
}
```

- Versus:

```
int[] ages = new int[15];
for (int i=0; i < ages.length; i++){
      System.out.println(ages[i]);
}
```

ACTIVITY

- Write a method that finds and returns the maximum value in an array of integers
- Write a method to find the first location of a specified value in an array

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

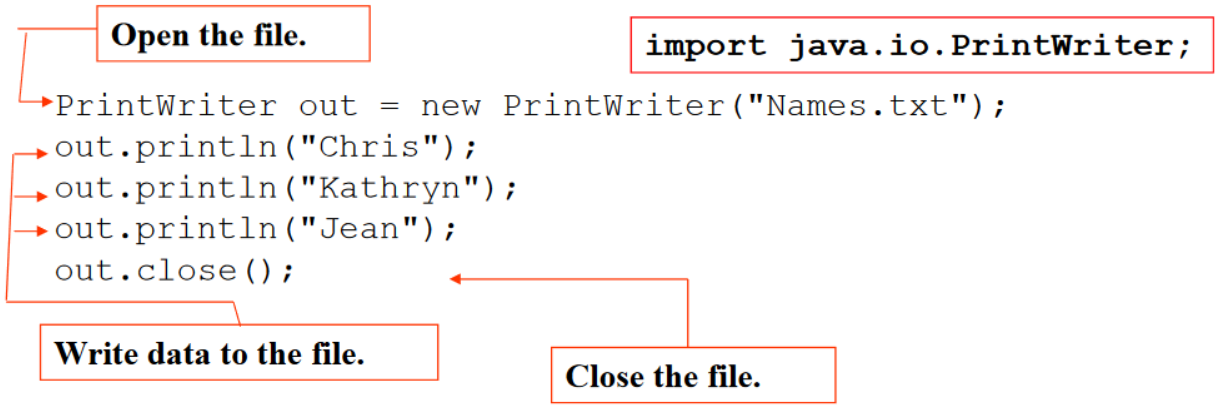
- File pointer indicates where the next read or write operation will take place
- Each file is treated as a one-dimensional sequence of characters
  - Non-printing characters for newlines
- Read position indicates what characters are returned on the next read operation
  - Read position is updated to 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



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
  - Stores them in an array
  - Prints out the sum of the numbers
- File will look like this:

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