# WEEK NINE

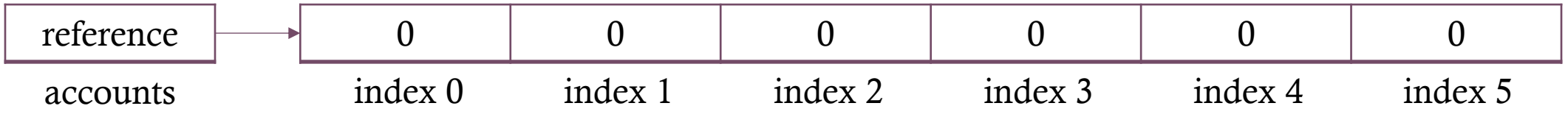
Acknowledgements: Slides created based off material provided by Dr. Travis Doom

# THE ARRAY

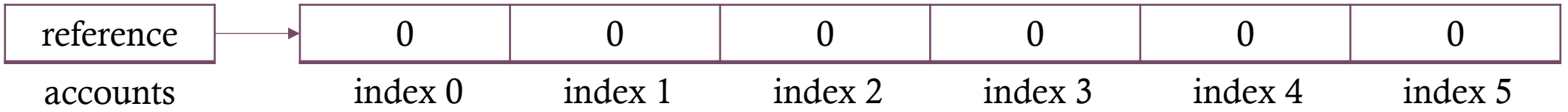
- Data structure
  - Contain groups of related items under one variable name
- Arrays
  - Simplest and most prevalent data structure
  - Object that contains items of the same data type
  - Each item is indexed by their order in the list (starting at 0)
  - Can hold primitive data types or objects
-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

# CREATING AN ARRAY

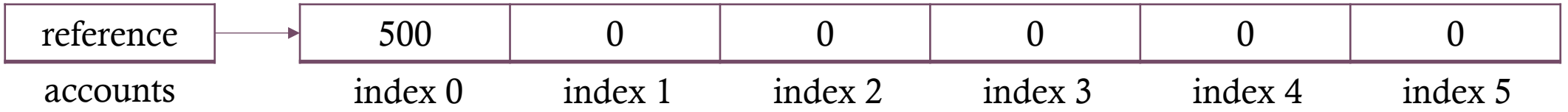
- An array is an object thus it needs an object **reference**
  - The reference is stored in a variable and refers to the place in memory that the object is stored
  - `int[] accounts;`
- When creating an array, we must define it with a permanent size
  - We can never directly change the size of this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



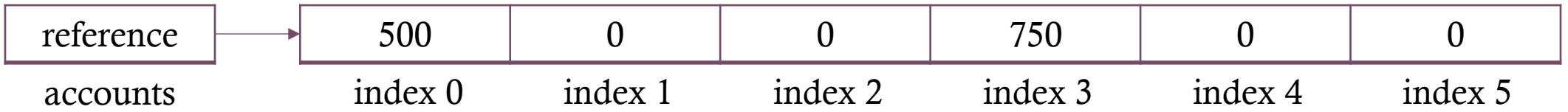
# ACCESSING AND MODIFYING ARRAYS



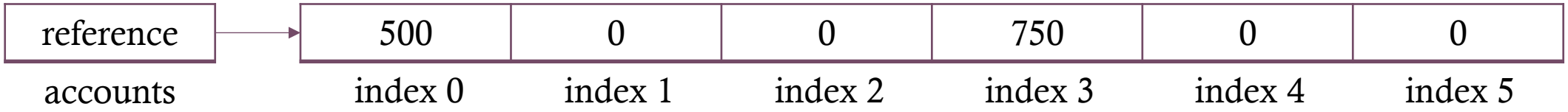

- Say we want to update the value of the first index
  - `accounts[0] = 500;`




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



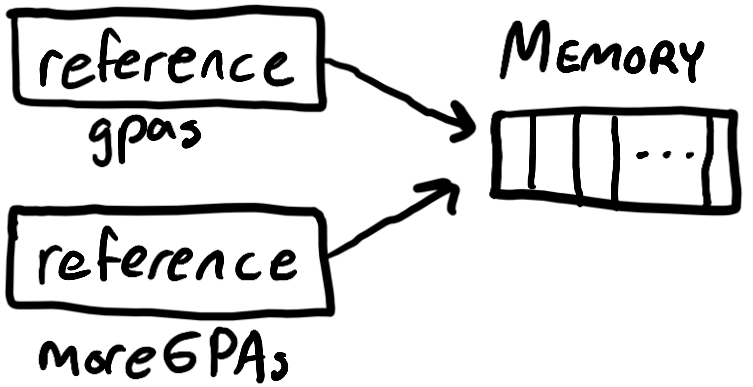
# MORE ABOUT ACCESSING ARRAYS




- 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 triples
- The method will be provided with a starting value, and a number of triples
- The method should then store each triple 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 triples,
  - The array should look like this: [5, 15, 45, 135]

# 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

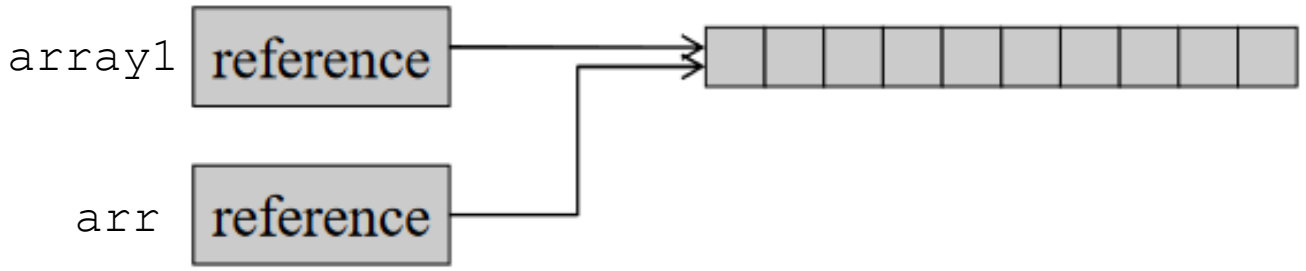
- Write a method that finds and returns the maximum value in an array of integers
- Write a method to find the first location of a specified value in an array

# 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

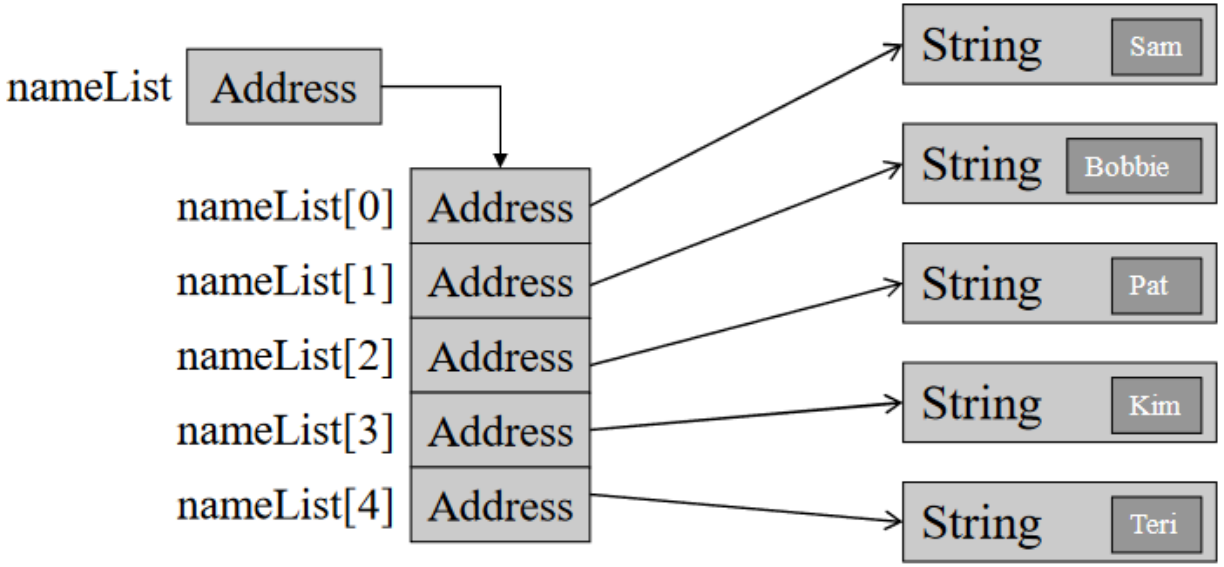
```
public static void main(String[] args) {
        int[] array1 = {1, 2, 3};
        zeroArray(array1);
        System.out.println(Arrays.toString(array1));
    }

    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"};
```

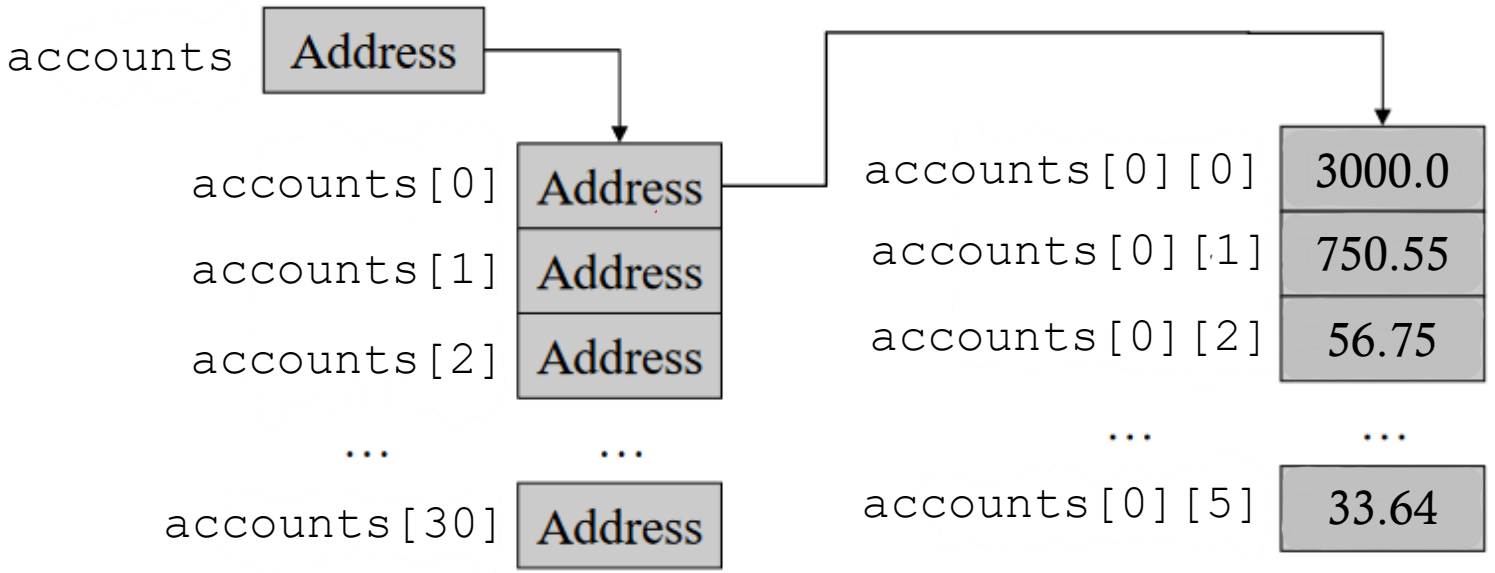
- Since String is an object, each index in the array holds the memory address of the object
- Essentially, we have an array of addresses



# 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];
```



# TWO-DIMENSIONAL ARRAYS

- Can be visualized as a grid of data
- Must declare with a size for number of rows and columns (still constant)

```
double[][] scores = new double[4][4];
```

| | column 0 | column 1 | column 2 | column 3 |
|---|---|---|---|---|
| row 0 | `Scores[0][0]` | `Scores[0][1]` | | |
| row 1 | `Scores[1][0]` | `Scores[1][1]` | | |
| row 2 | | | | |
| row 3 | | | | `Scores[3][3]` |

# RAGGED ARRAYS

- We can store different data types in an array by specifying the type as Object

```
Object[][] array = { {3.0, 5.6}, {true, false}, {3, 5} };
```

- When dealing with multi-dimensional arrays, the arrays within the array can be different lengths

```
int[][] numList = { {1, 2, 3},
                    {4, 5, 6, 7, 8},
                    {9, 10} };
```

OR

```
int[][] array = { new int[3], new int[2], new int[4] };
```

# ARRAYLISTS

- Found in the java.util library
- Similar to an array but with additional functionality
  - Can hold objects of different types in the same list
  - Automatically expands and reduces on demand
  - Still indexed
- Upon creation we can specify a type for the ArrayList or use Object if we want to hold any object

```
ArrayList<String> names = new ArrayList<>();
ArrayList<Object> everything = new ArrayList<>();
```

- Specifying a type helps avoid issues and avoid typecasting

# ARRAYLIST METHODS

- `.size()`: returns the size of the ArrayList
- `.add( object )`: adds the reference to the object to the end of the list
- `.add( index, object )`: inserts the object reference at the specified index
- `.set( index, object )`: overwrites the current index value with the object reference
- `.get( index )`: returns the object reference at that index (not removed)
- `.remove( index )`: returns and removes the object reference
- `.clear()`: removes all elements from the list
- `.contains( object )`: checks if the specified object exists in the list
- `.indexOf( object )`: returns the index of the specified object

# WRAPPER CLASSES

- If we want to create an ArrayList of `int` or `double`, we have to say `Integer` or `Double`
- Both are wrapper classes for the primitive data types
- Surround an existing class to make it an object and add additional functionality
- ArrayLists cannot take in primitive types, so we have to use the wrappers
- `Integer` and `Double` also have useful methods that allow us to cast a String to an int or double and vice versa
  - `int num = Integer.parseInt("3");`
  - `String str = Integer.toString(3);`

# ACTIVITY

- Write code that constructs a two-dimensional array that holds a number of movies and the reviews for each movie
  - This can be done through initialization
  - You can choose the size of the two-dimensional array, but it should be at least 3x3
  - Reviews are on a 10-point scale (e.g. 8 or 5)
- Write a method that finds the average review for a given movie and returns it
- Lastly, print out the movie with the highest average review