CIS 110: Introduction to Computer Programming

Lecture 17

All hail the mighty array

(§ 7.1)

Outline

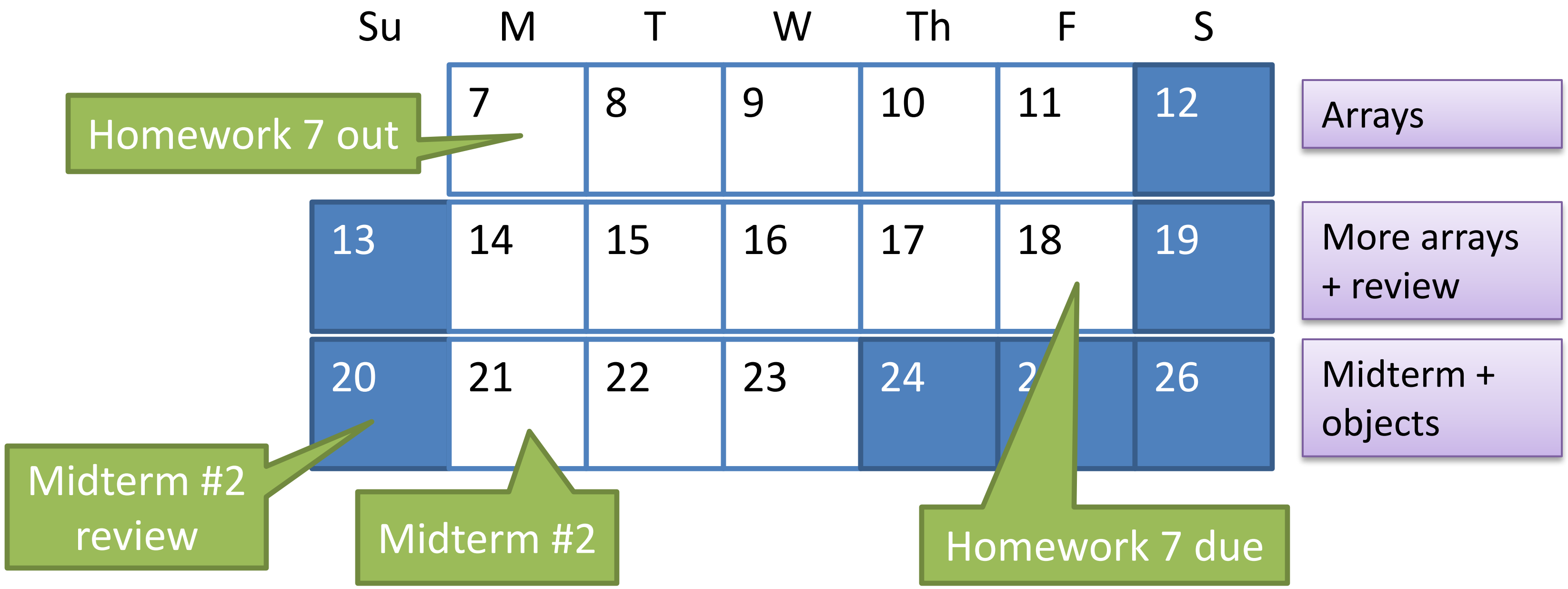
- Catch-up from last week: file output
- Introduction to Arrays

About me

- My name: Michael-Peter Osera.
  - Call me Michael-Peter.

- I am a
  - 4th year Ph.D. student (*not* a professor).
  - Ethnomusicology researcher.
  - Die-hard supporter of clubs in and around Philly!

Now to midterm #2



File output

The PrintStream class

- PrintStreams allow us to write to files!

```
public static void main(String[] args)
    throws FileNotFoundException {
  PrintStream out = new PrintStream(new File("helloworld.txt"));
  out.println("Hello World!");
}
```

- PrintStreams have:
  - print
  - println
  - printf
- Sound familiar?

helloworld.txt

```
Hello World!
```

System.out is a PrintStream!

```java
public class System {
  public static PrintStream out = /* ... */;
}
```

- System.out is a PrintStream that outputs to the console.
- PrintStreams that we make function identically but output goes to a File instead!

Example: AllCapsWriter

```
public class AllCapsWriter {
  public static void main(String[] args)
      throws FileNotFoundException {
    Scanner in = new Scanner(
      new File("in.txt"));
    PrintStream out = new PrintStream(
      new File("out.txt"));
    while (in.hasNextLine()) {
      out.println(in.nextLine().toUpperCase());
    }
  }
}
```

Arrays

Remembering lots of stuff

- We can't store arbitrary amounts of data!

```java
Scanner in = new Scanner(System.in);
System.out.println("Enter 10 numbers: ");
double current = 0;
for (int i = 0; i < 10; i++) {
  System.out.print("Enter a double: ");
  current = in.nextDouble();
  // ...
}
```

- We necessarily "forget" each double the user enters after each iteration of the loop.

Introducing the array

- Arrays allow us to store lots of data at once!

```
Scanner in = new Scanner(System.in);
System.out.println("Enter 10 numbers: ");
double[] values = new double[10];
for (int i = 0; i < 10; i++) {
  System.out.print("Enter a double: ");
  values[i] = in.nextDouble();
}
```

- The `values` array contains all 10 doubles entered by the user!

Declaring arrays

- An array is an *object* that contains multiple values.

A variable *values* that holds an array of doubles.

```
double[] values = new double[10];
```

An *array type*. "An array of doubles"

Array *creation.* Instantiates a new array containing 10 doubles.

- Arrays are *homogenous*: all elements have the same type.

Accessing Arrays

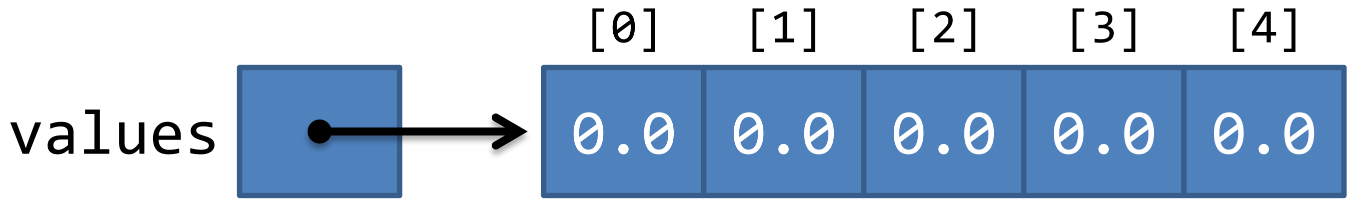
- We *index* into arrays to access individual elements.

```
values[i] = in.nextDouble();
```

Index into the *ith* position of the array and store the next double from the user there.

Array operation examples: initialization

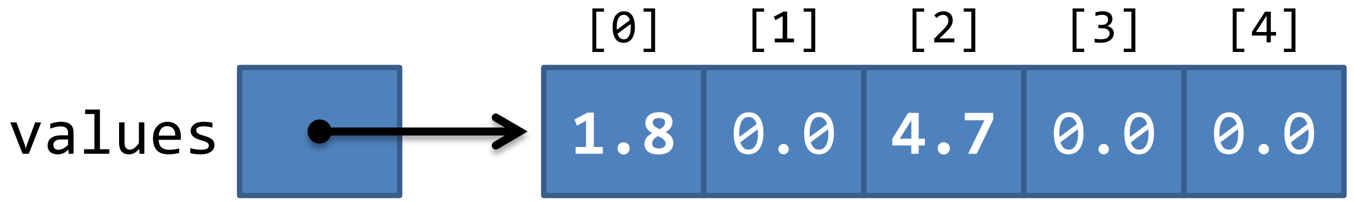
```
double[] values = new double[5];
```



- Array variables containing *references* to arrays.
- Array elements are *auto-initialized* to "zero values".

Array operation examples: assignment

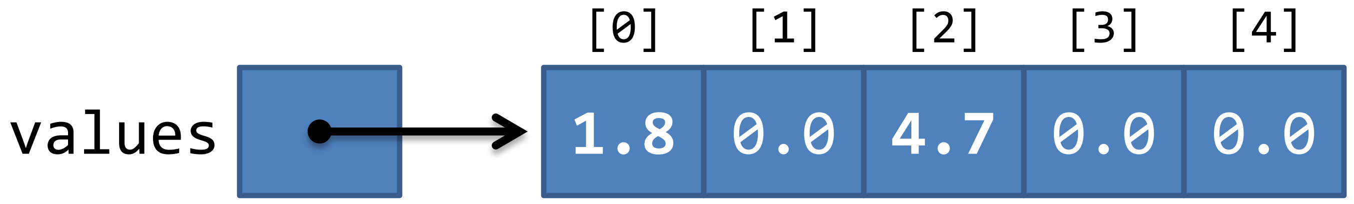
```
values[2] = 4.7;
values[0] = 1.8;
```



- Arrays have zero-based indices.
- An *indexed array* is just a storage location we can assign into and access like a variable.

Array operation examples: usage

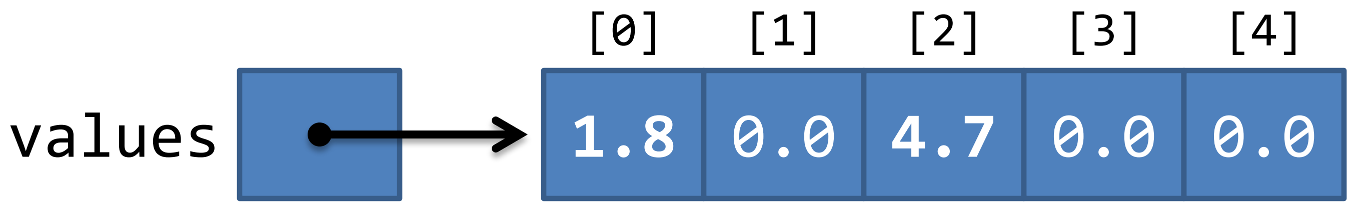
```
System.out.println(values[0] + values[2]);
> 6.5
```



- To use an element of the array, we simply index into the array at the desired position.

Array operation examples: length

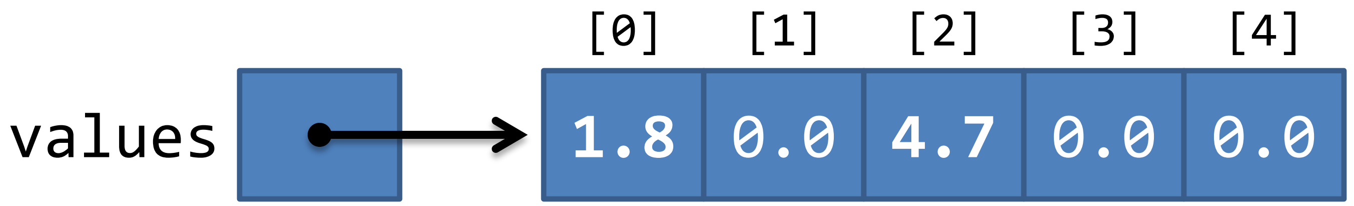
```
System.out.println("Length = " + values.length);
> 5
```



- The *length* field of the array object contains that arrays length.
- Note: no parenthesis after length unlike Strings!
  - It really is a *variable* rather than a *method*.

IndexOutOfBoundsExceptions

```
values[10] = 4.1;
> Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 10
```



- We get IndexOutOfBounds exceptions if we try to access an index that doesn't exist.
- We can't change the size of an array once it is made!

For-loop traversal template

- We can put this all together to write a useful traversal pattern:

```
for (int i = 0; i < arr.length; i++) {
  /** ...arr[i]...*/
}
```

- *For each element of arr, do something to it.*

Enhanced for-loop

- "Doing something" to each element is so common there's special syntax to do this:

```
int arr[] = new int[10];
for (int i : arr) {
  /** ...i... */
}
```

- Upside: succinct captures *for each element…*
- Downside: don't have access to *current index*.

Random access

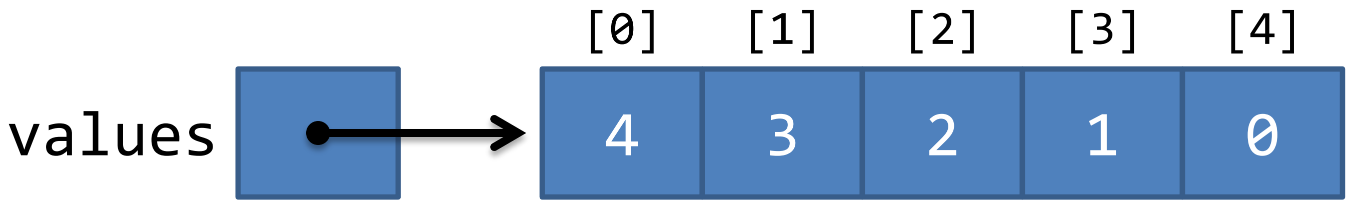
- With Scanners, we process data *sequentially*.
- Arrays allow us to have *random access* to data.

```
Scanner in = new Scanner(System.in);
System.out.println("Enter 10 lines: ");
String[] lines = new String[10];
for (int i = 0; i < 10; i++) {
  System.out.print("Enter a line: ");
  lines[i] = in.nextLine();
}
System.out.print("Which line do you want? ");
int index = in.nextInt();
System.out.println("Index " + index + " = " + lines[index]);
```

Alternative array initialization

- There's special syntax for initializing an array with non-default values.

```
int[] values = { 4, 3, 2, 1, 0 };
```



References, arrays, and methods

- Methods can change the value of arrays unlike with variables!
  - Due to *reference semantics* that we'll talk about next time.

```java
public static void initialize(int[] values) {
  for (int i = 0; i < values.length; i++) {
    values[i] = values.length - i - 1;
  }
}

public static void main(String[] args) {
  int[] values = new int[3];
  // Before: values = { 0, 0, 0 }
  initialize(values);
  // After: values = { 2, 1, 0 }
}
```