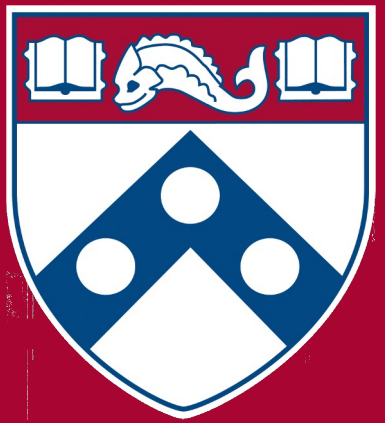
Arrays

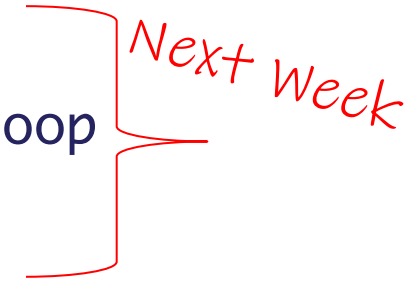


Overview

- Often, we need to store and manipulate several variables; a program will use an array to store a collection of variables of the same type
- In this module we will learn how to store several variables inside an array
- You can think of an array as a collection of variables in which each variable occupies a specific position
- Example:
    - Store your friends' phone numbers inside an array named *besties*

Learning Objectives

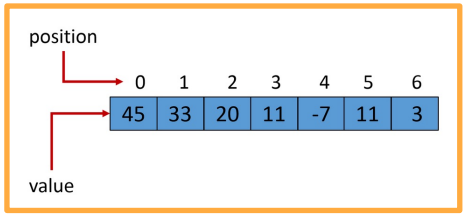
- To be able to declare an array
- To be able to initialize an array with the `new` keyword
- To be able to initialize an array with an initializer list
- To be able to access array values
- To be able to modify array values
- To be able to traverse an array using the `for-loop`
- To be able to traverse an array using the `enhanced for-loop`
- To be able to solve problems using arrays



Modeling with arrays

Element type decided with variable declaration

- `Arrays` are used to **store several elements of the same type**
- `Arrays` are a type of data structure
- An `array` is like a list in real life: list of students, list of songs (playlist), etc.
- An array has a **fixed length** (the number of elements that can be stored in it)
- Each element in an array has a **position or index**:
  - The first element is at `index 0`
  - The last element is at `index (length of the array) - 1`



Index numbering similar to strings

Declaring and Creating Arrays

- `Arrays` are an `object` type
- To declare an array, you write

```
TypeOfElements[] arrayName = new TypeOfElements[length];
```

Example: `int[] myArray = new int[6]`

Create an array of `integers` of length `6`

`Student[] studentsArray = new Student[10]`

Create an array of `Student` of length `10`

Default Values

- When we initialize an array with **new**, Java will make each element some "default" value depending on the type

Example: `int[] myArray = new int[6]`
`integers` in the array will start of as `0`

`Student[] studentsArray = new Student[10]`
`Strings` (or any other object type) in the array will start as `null`

`doubles` start as `0.0`, `Booleans` start as `false`.

Initializer list

Only works on same line you are declaring the array

- To declare an array using **an initializer list**, you write

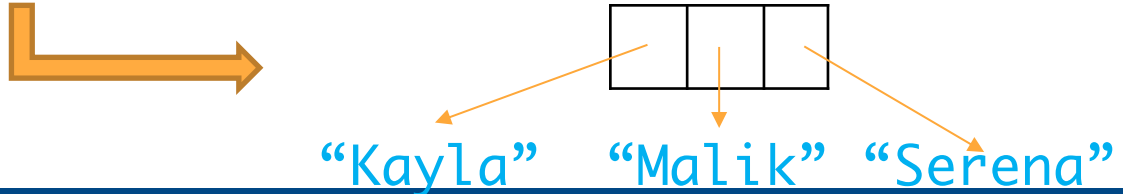
```
TypeOfElements[ ] arrayName = {element1, element2, …};
```

What is the `length` of `myArray`?

Example:

```
int[] myArray = {1, 2, 4, 5, 6};
```


```
String[] names = {“Kayla”, “Malik”, “Serena”};
```



Array length

- The **length** of the array **cannot be changed after initialization**
- To get the length you write

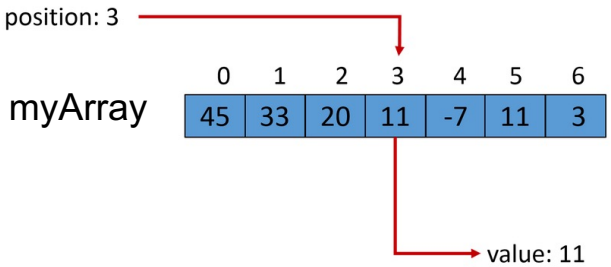
```
arrayName.length
```

```
int[] myArray = {1, 2, 4, 5, 6};
myArray.length ➔ 5

String[] names = new String[10];
names.length ➔ 10
```

Accessing Array Values

- Elements inside an array are **accessed** based on their **position** or **index**
- To access the element at position `i` you write

```
arrayName[i]
```




```
myArray[3] ➔ 11
myArray[0] ➔ 45
myArray[myArray.length -1] ➔ 3
```

- Trying to access an element at a `position < 0 or >= array.length` will raise an `Error` (`ArrayIndexOutOfBoundsException`)

Modifying Array Values

- To assign a new value at the position `i` you write

```
arrayName[i] = new_value;
```

```
int[] myArray = {1, 2, 4, 5, 6};
myArray[0] = 5; ➔ {5, 2, 4, 5, 6}
myArray[4] = 10; ➔ {5, 2, 4, 5, 10}
```

- Trying to modify an element at a `position < 0 or >= array.length` will raise an `Error` (`ArrayIndexOutOfBoundsException`)

What’s the problem here?

```
int[] numbers;

int doubled = numbers[0] + numbers[0];
```

What’s the problem here?

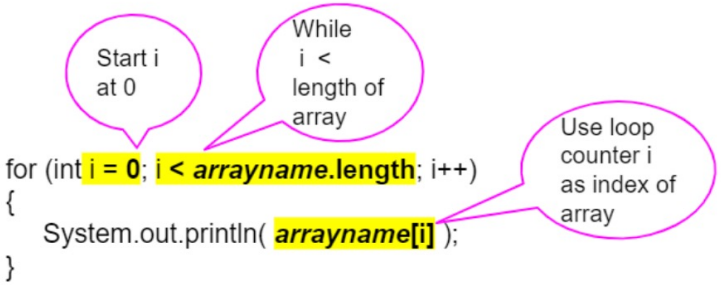
```
int[] numbers = new int[3];

int doubled = numbers[8] + numbers[8];
```

Traversing Arrays

- Often, we need to iterate through all the elements inside an array:
  - To find a specific value
  - To perform a computation: for example, the sum of all the elements
  - Many more reasons
- We use a **loop** to **iterate** through an array

Array iteration with a `For-loop`

- Start the **loop control variable** at `0` (the smallest position)
- Keep looping as long as the control loop variable is within the bounds of the array (`< array.length`)
- Increment the loop control variable between each iteration
- Use the **loop control variable** to **access** the **elements** inside the array



What do you think this code will do?

Parallel Arrays

- As we want our programs to solve more complicated problems, we have to handle more complicated representations of data.
- **Example**: To calculate weekly revenues from movie theater ticket sales, we need to know:
    - Tickets sold per day
    - Ticket prices per day

Parallel Arrays

- Parallel Arrays allow us to store multiple data points that belong to the same entity along several arrays:

```
int[] ticketSales        = {301,  408,  201,  202,  680,   710,  590};
double[] ticketPrices  = {7.00, 7.00, 12.0, 12.0, 12.0, 13.0, 13.0};
```

- Both of these arrays have seven entries, one for each day of the week! (Day 0 → Monday, Day 1 → Tuesday, ... Day 6 → Sunday)

Parallel Arrays

- Parallel Arrays allow us to store multiple data points that belong to the same entity along several array

```
int[] ticketSales        = {301, 408, 201, 202, 680,  710, 590};
double[] ticketPrices  = {7.00, 7.00, 12.0, 12.0, 12.0, 13.0, 13.0};
```

- How can we calculate Monday's revenue?

Parallel Arrays

- Parallel Arrays allow us to store multiple data points that belong to the same entity along several array

```
int[] ticketSales        = {301, 408, 201, 202, 680,  710, 590};
double[] ticketPrices  = {7.00, 7.00, 12.0, 12.0, 12.0, 13.0, 13.0};
```

- How can we calculate Monday's revenue?
    - ticketSales[0] * ticketPrices[0];

Parallel Arrays

- Parallel Arrays allow us to store multiple data points that belong to the same entity along several array

```
int[] ticketSales        = {301, 408, 201, 202, 680,  710, 590};
double[] ticketPrices  = {7.00, 7.00, 12.0, 12.0, 12.0, 13.0, 13.0};
```

- How can we calculate revenue for the *i*th day of the week?
    - ticketSales[i] * ticketPrices[i];

Parallel Arrays

- Parallel Arrays allow us to store multiple data points that belong to the same entity along several array

```
int[] ticketSales        = {301, 408, 201, 202, 680,  710, 590};
double[] ticketPrices    = {7.00, 7.00, 12.0, 12.0, 12.0, 13.0, 13.0};
```

- How can we calculate average daily revenue from this week?
    - Iteration!

Command Line Arguments

- It’s annoying to change the values of variables in the program whenever you want the behavior to change
- **Command Line Arguments** allow the user to pass in values when *running* the program

Command Line Arguments

- Accessing the values within the program:
    - In **main**, we have access to the command line arguments inside of the **String[] args**
- All arguments are stored as Strings by default:
    - String name = args[0]; // read the first CLA as a String
- We can **parse** ints and doubles
    - int age = Integer.parseInt(args[3]); // read the fourth CLA as an int
    - double height = Double.parseDouble(args[1]); // read 2nd CLA as double

Array iteration with the Enhanced `For-loop`

- The enhanced `for loop` is called the `for each loop`
- The `for each loop` does not use an index to traverse an array
- The `for each loop` uses a variable that will refer to each value in the array from the first position (`0`) to the last (`arrayname.length -1`)
- The iteration stops after the variable reaches the last element
- To use the `for each loop` you write

```
for (TypeOfElements variable : arrayName){
    //body
}
```

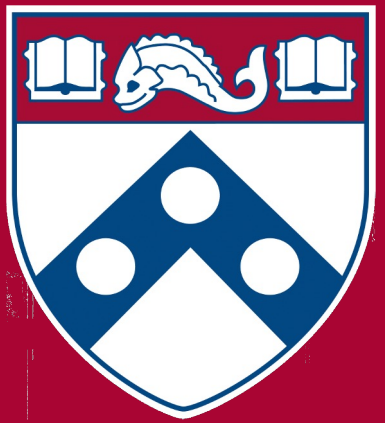
Array iteration with the Enhanced For-loop

```
String[] names = {“Kayla”, “Malik”, “Serena”};

for(String name : names){

    System.out.println(name);

}
```

- Will print:

```
“Kayla”     // first value of name
“Malik”     // second value of name
“Serena”    // third value of name
```

In.java



Purpose

- **In** is another library (like **PennDraw**) that provides a series of functions you can use to read information from a file!
    - Instead of putting all of our data directly in a program, we can direct the program to read data from a file elsewhere on our computer.
- We'll use this throughout the course, starting in HW02 (NBody)

Getting Started with In

- To open up a file and get it set up for reading, we do the following:

```
In inStream = new In(filename);
```

The In Toolkit

- Once you have initialized an **In** object and stored it in a variable, you can use any of these functions to read data from the file!

```
inStream.isEmpty(); // boolean value that is true if there are no more values,
false otherwise
inStream.readInt(); // reads in an int from inStream
inStream.readDouble(); // reads in a double from inStream
inStream.readBoolean(); // reads in a boolean from inStream
inStream.readString(); // reads in a string from inStream
inStream.readLine(); // reads in an entire line from inStream (as a String)
inStream.readAll(); // reads in the entire file from inStream (as a String)
```

The Reading Model

- `inStream` will start reading from the beginning of the file (top left)
- Each time a function (like `readDouble()`) is called, `inStream` attempts to read the next (unread) data from the file and interpret it as the type that you asked for
    - an error will occur if the data cannot be parsed to the requested type.
- The next time a read function is called, `inStream` moves to the next item in the file.

The Reading Model

sample.txt:

```
4 5
Hello true 3.0
32
342
4 39 491 1
12 1239 1
12 1239 110210 1
 293 1.493
193 true
23901 391 30 3
Yes green orange
Think
```

- After creating the inStream, the file is set to start reading from the top left. (The cursor is the orange line there)

In inStream = new In(“sample.txt”);

The Reading Model

sample.txt:

```
4 5
Hello true 3.0
32
342
4 39 491 1
12 1239 1
12 1239 110210 1
 293 1.493
193 true
23901 391 30 3
Yes green orange
Think
```

- Reading an int causes the **4** to be read and moves the reading position forward.

inStream.readInt(); → evaluates to **4**

The Reading Model

sample.txt:

```
4 5
Hello true 3.0
32
342
4 39 491 1
12 1239 1
12 1239 110210 1
 293 1.493
193 true
23901 391 30 3
Yes green orange
Think
```

- Reading an *another* int causes the **5** to be read and moves the reading position forward.

inStream.readInt(); → returns the int **5**

The Reading Model

sample.txt:

```
4 5
Hello true 3.0
32
342
4 39 491 1
12 1239 1
12 1239 110210 1
 293 1.493
193 true
23901 391 30 3
Yes green orange
Think
```

- Reading an *yet another* int causes the program to crash! The next data in the file is the character 'H', which can't be the start of an int.

inStream.readInt(); → ***CRASH!***

The Reading Model

sample.txt:

```
4 5
Hello true 3.0
32
342
4 39 491 1
12 1239 1
12 1239 110210 1
 293 1.493
193 true
23901 391 30 3
Yes green orange
Think
```

- Instead, we could have tried to read it as a String, which reads one word at a time and moves the reading position again.

inStream.readString(); → **“Hello”**