Arrays

Our first collections.

Overview of Arrays (7.1)

What Does an Array Do?

Up until now: variables store a single data item (e.g. `59` or `"Hello"`)

Now: *arrays* are special variables that store a list of data items under a single name.

- Each of the data items in the array is called an *element*

```
|-------array--------|
[4, 5, 9, 10, -20, 32]
    ⬆️ one element
```

Arrays Have Indices

Since arrays are collections of data, we need some way of accessing the individual pieces of data inside the array.

Each element's location number is called the **index**.
If an array has `n` elements, then the indices of the array range from `0` to `n-1`

```
ARRAY   :  [39.02, 902.094, 94.2, -29.3]
INDICES:    0       1         2     3
```

Array Elements Can Be Accessed & Set

```
grades <-- [94, 89, 77, 100, 88]
```

Accessing the third element of the array: `grades[2]`

Assigning a new value to the first element: `grades[0] = 98`

Activities 7.1.3 & 7.1.4 in the book give excellent practice on these fundamentals!

Practice with Indices

I have an array with 10 elements called `studentNames`...

1. How can I assign the first element in the list to be `"Han"`?
2. What are the valid indices for elements in `studentNames`?
3. How can I set the last element to be the same as the first element?

Practice with Indices

Answers

1. `studentNames[0] = "Han";`?
2. `0` through `9`
3. `studentNames[9] = studentNames[0];`

Array Syntax in Java

Declaring & Allocating a New Array

```
dataType[] arrayName = new dataType[numElements];
```

This tells Java to declare a new variable called `arrayName`. This variable has type `dataType[]`, which reads aloud like "data type array".

The initial value of `arrayName` is set to be a new array of type `dataType[]` with a length of `numElements`.

e.g. `int[] officeHourAttendance = new int[5];`

More Rules about Indices

- Indexing into an array is an expression
  - `arr[1]` works
  - `arr[i]` works
  - `arr[i * 3 - j + 2]` works
- The type of the index must be `int`
  - `arr["Harry"]` does not work!

Exercise: Our First Array

Write a program that declares an array that will store doubles named `homeworkScores`. The array should have a length of `4`.

The array stores the average homework scores on the first `4` homeworks for a class. The average for HW1 was `89.3`, for HW2 was `99.0`, for HW3 was `77.8`, and for HW4 was `82.84`. Store these values in the correct order in the array `homeworkScores`.

Solution: Our First Array

```java
public class FirstArray {
    public static void main(String[] args) {
        double[] homeworkScores = new double[4];

        homeworkScores[0] = 89.3;
        homeworkScores[1] = 99.0;
        homeworkScores[2] = 77.8;
        homeworkScores[3] = 82.84;
    }
}
```

Printing Arrays

More complicated than we might hope...

```
public class FirstArray {
    public static void main(String[] args) {
        double[] homeworkScores = new double[4];
        homeworkScores[0] = 89.3;
        homeworkScores[1] = 99.0;
        homeworkScores[2] = 77.8;
        homeworkScores[3] = 82.84;
        System.out.println(homeworkScores);
    }
}
```

➡️ `[D@65e579dc` 🤔 🤔 🤔 🤔

Printing Arrays

We can still print the individual elements of the array.

```
public class FirstArray {
    public static void main(String[] args) {
        double[] homeworkScores = new double[4];
        homeworkScores[0] = 89.3;
        homeworkScores[1] = 99.0;
        homeworkScores[2] = 77.8;
        homeworkScores[3] = 82.84;
        System.out.println(homeworkScores[1]);
    }
}
```

➡

```
99.0
```

Loops and Arrays

We can use for loops to print out our arrays!

```
public class FirstArray {
    public static void main(String[] args) {
        double[] homeworkScores = new double[4];
        // homeworkScores elements set as before...

        for (int i = 0; i < homeworkScores.length; i++) {
            System.out.print(homeworkScores[i] + " ");
        }
    }
}
```

➡ `89.3 99.0 77.8 82.84` ✅

Loops & Arrays

All arrays have a field `length` that stores how many elements are inside.

```
String[] names = new String[17];
System.out.println(names.length);
```

➡️ `17`

To loop over every index in an array, start from `0` and increment until the last valid index, `arr.length - 1`.

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

Check-in

```
int NUM_STUDENTS = 55;
int NUM_TAS = 5;
String[] pennkeys = new String[NUM_STUDENTS + NUM_TAS];
for (int i = 0; i < pennkeys.length; i++) {
    System.out.println("Enter Student/TA PennKey:");
    pennkeys[i] = scnr.next();
}
```

1. How many iterations does this for loop run for?
2. What is the value of `pennkeys.length`?
3. How could I change the loop so that I only ask for `55` PennKeys?

Check-in

```
int NUM_STUDENTS = 55;
int NUM_TAS = 5;
String[] pennkeys = new String[NUM_STUDENTS + NUM_TAS];
for (int i = 0; i < pennkeys.length; i++) {
    System.out.println("Enter Student/TA PennKey:");
    pennkeys[i] = scnr.next();
}
```

1. `60`
2. `60`
3. `for (int i = 5; i < pennkeys.length; i++)` (and others)

Alternative Array Initialization

Only at the same time that you declare the array variable, you can write out the elements of the array manually:

```
int[] newArray = {5, 7, 11};
```

- The elements are automatically set
  - `newArray[0] = 5, newArray[1] = 7, newArray[2] = 11`
- The length of the array is automatically interpreted
  - Don't need to provide that the length is `3`
  - `newArray.length == 3` automatically

Exercise: Printing in Reverse

Can you write a loop that prints out the elements of an array in reverse? Think carefully about loop variable! Where should it start? How should it be updated? When should it stop?

```
int[] toPrint = {1, 2, 3, 4, 5};

// your code here
// Should print out 5 4 3 2 1
```

Solution 1: Printing in Reverse

```java
int[] toPrint = {1, 2, 3, 4, 5};

for (int i = toPrint.length - 1; i >= 0; i--) {
    System.out.println(toPrint[i]);
}
```

Iterate backwards from `toPrint.length - 1` down to (and including) `0`, decrementing `i` by `1` every iteration.

Solution 2: Printing in Reverse

```java
int[] toPrint = {1, 2, 3, 4, 5};

for (int i = 0; i < toPrint.length; i++) {
    System.out.println(toPrint[toPrint.length - (i + 1)]);
}
```

Iterate forwards the normal way, but access the indices using the formula `toPrint.length - (i + 1)`. *Why does this work?*

Bad Solution 1: Printing in Reverse

```
int[] toPrint = {1, 2, 3, 4, 5};

for (int i = toPrint.length - 1; i > 0; i--) {
    System.out.println(toPrint[i]);
}
```

What's wrong?

Bad Solution 1: Printing in Reverse

```java
int[] toPrint = {1, 2, 3, 4, 5};

for (int i = toPrint.length - 1; i > 0; i--) {
    System.out.println(toPrint[i]);
}
```

What's wrong? *We never print the first element of the array since we stop before* `i = 0`

Bad Solution 2: Printing in Reverse

```java
int[] toPrint = {1, 2, 3, 4, 5};

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

What's wrong?

Bad Solution 2: Printing in Reverse

```
int[] toPrint = {1, 2, 3, 4, 5};

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

What's wrong? *We try to access `toPrint[toPrint.length - 0]` in the first iteration, which is not a valid index. Only `0` through `toPrint.length - 1` are valid.*

Live Coding: Finding the Biggest Element in an Array

Biggest Element Solution

```java
public static void main(String[] args) {
    Scanner scnr = new Scanner(System.in);
    int NUM_INPUTS = 5;
    double[] inputs = new double[NUM_INPUTS];

    for (int i = 0; i < inputs.length; i++) {
        inputs[i] = scnr.nextDouble();
    }

    double biggestSoFar = inputs[0];
    for (int i = 0; i < inputs.length; i++) {
        if (inputs[i] > biggestSoFar) {
            biggestSoFar = inputs[i];
        }
    }
    System.out.println("Biggest element: " + biggestSoFar);
}
```

Exercise: Smallest Element

```java
public static void main(String[] args) {
    Scanner scnr = new Scanner(System.in);
    int NUM_INPUTS = 5;
    double[] inputs = new double[NUM_INPUTS];

    for (int i = 0; i < inputs.length; i++) {
        inputs[i] = scnr.nextDouble();
    }

    // YOUR SOLUTION HERE

    System.out.println("Biggest element: " + biggestSoFar);
}
```

Solution: Smallest Element

```java
public static void main(String[] args) {
    Scanner scnr = new Scanner(System.in);
    int NUM_INPUTS = 5;
    double[] inputs = new double[NUM_INPUTS];

    for (int i = 0; i < inputs.length; i++) {
        inputs[i] = scnr.nextDouble();
    }
    double smallestSoFar = inputs[0];
    for (int i = 0; i < inputs.length; i++) {
        if (inputs[i] < smallestSoFar) {
            smallestSoFar = inputs[i];
        }
    }
    System.out.println("Smallest element: " + smallestSoFar);
}
```

Exercise: CIT 591 Grade Calculator

Prompt the user for for their homework score, their average exam score, their final project score, and their attendance score. Store these values in an array, and then use that array to calculate their grade in the class. (~5 minutes to plan, then we'll code together.)

Component	Weight
HW	60%
Exam	20%
Final Project	10%
Attendance	10%

Representing Complex Data with Multiple Arrays

Representing a restaurant's menu items.

```java
String[] itemNames = {"Chicken", "Falafel", "Lamb", "Fish"};
int[] itemPrices = {6, 5, 7, 6};

for (int i = 0; i < itemNames.length; i++) {
    System.out.println(itemNames[i] + ": $" + itemPrices[i]);
}
```

Representing Complex Data with Multiple Arrays

Better still: `MenuItem` class to fill a single array

```
public class MenuItem {
    private String name;
    private int price;

    // constructor, getters, etc.
}
```

Representing Complex Data with Multiple Arrays

Better still: `MenuItem` class to fill a single array

```
MenuItem[] menu = new MenuItem[4];
menu[0] = new MenuItem("Chicken", 6);
menu[1] = new MenuItem("Falafel", 5);
menu[2] = new MenuItem("Lamb", 7);
menu[3] = new MenuItem("Fish", 6);
for (int i = 0; i < menu.length; i++) {
    MenuItem curr = menu[i]
    System.out.println(curr.getName() + ": $" + curr.getPrice());
}
```

Fundamental Rules of Arrays

- Once created, an array object's size cannot change
- Copying an array into another variable does not create a new array

```
int[] x = {3, 4, 5};
int[] y = x;
x[1] = 17;
System.out.println(y[1]); // --> Prints 17!!!
```

- In order to duplicate an array, you need to initialize an entirely new array of the same size.

Copying an Array

```java
int[] x = {3, 4, 5};
int[] y = new int[x.length]; // init new array of the same size
for (int i = 0; i < x.length; i++) {
    y[i] = x[i]; // copy each element individually
}
x[1] = 10;
y[1] = -10;


System.out.println(x[1]); // --> prints 10;
System.out.println(y[1]); // --> prints -10
```

Exercise: Reversed Copy

Write a program that copies the values of an array into another new array, but in reverse!

e.g. `{4, 3, 6} -> {6, 3, 4}`

Solution: Reversed Copy

Write a program that copies the values of an array into another new array, but in reverse!

```
int[] x = {3, 4, 5};
int[] reversed = new int[x.length]; // init new array of the same size
for (int i = 0; i < x.length; i++) {
    reversed[reversed.length - i - 1] = x[i]; // copy each element individually
}
```

Put `x[0]` into `reversed[2]`, `x[1]` into `reversed[1]`, and `x[2]` into `reversed[0]`

Swapping

Swapping array elements is a little more complicated than it might seem.

What happens here?

```
String[] names = {"Harry", "Adrian", "Vivian"};
names[0] = names[2];
names[2] = names[0];
```

Swapping

Swapping array elements is a little more complicated than it might seem.

What happens here?

```
String[] names = {"Harry", "Adrian", "Vivian"};
names[0] = names[2];
names[2] = names[0];
```

`names` becomes `{"Vivian", "Adrian", "Vivian"}` since `"Harry"` gets overwritten by `"Vivian"`.

Swapping

Swapping correctly:

```
String[] names = {"Harry", "Adrian", "Vivian"};
String temp = names[0]; // temp is "Harry", this doesn't change with names

names[0] = names[2]; // We get {"Vivian", "Adrian", "Vivian"}
names[2] = temp; // We get {"Vivian", "Adrian", "Harry"}
```

Strings are like Arrays of Characters

If we have a `String s`...

- `s.length()` returns the length of the string.
  - Note the parentheses at the end!
- `s.charAt(int idx)` returns the `char` (a single character) at index `idx` in the string.
- String indexing works exactly like array indexing
  - the first character lives at index `0`, the last at `s.length() - 1`

Example: Checking Title Case

Does the String start with an uppercase letter and then have only lowercase letters?

```java
public boolean isTitleCase(String str) {
    // check if the string is empty to start (return true)

    // check if the first char is not uppercase (return false)

    // check if any following char is not lowercase letter (return false)

    // if all other conditions aren't met, then return true
}
```

Example: Checking Title Case

```
public boolean isTitleCase(String str) {
    if (str.length() == 0) {
        return true;
    }
    if (str.charAt(0) <= 'A' || str.charAt(0) >= 'Z') {
        return false;
    }
    for (int i = 1; i < str.length(); i++) {
        char curr = str.charAt(i);
        if (curr <= 'a' || curr >= 'z') {
            return false;
        }
    }
    return true;
}
```