COMP90041
Programming and Software Development
2020 - Semester 1
Lab 6 - Week 7

Yuhao(Beacon) Song

yuhsong1@unimelb.edu.au

Introduction

- Timetable
  - Tue(11) 14:15-15:15 (Melbourne Time) Join URL: https://unimelb.zoom.us/j/490084146
  - Tue(07) 16:15-17.15 (Melbourne Time) Join URL: https://unimelb.zoom.us/j/291505765

- Contact
  - yuhsong1@unimelb.edu.au
  - yuhsong@student.unimelb.edu.au
  - Github: https://github.com/Beaconsyh08/COMP90041_Programming_and_Software_Development_Tutorials.git

Outline

- Lecture Review
- Exercise & demo
- Project B

Declaring and Creating an Array

- An array is declared and created in almost the same way with objects:

  BaseType[] ArrayName = new BaseType[size];

  int[] arr1 = new int[5];

  - The size may be given as an expression that evaluates to a **nonnegative integer**

- An array can be initialized when it is declared
  - Values for the indexed variables are enclosed in **braces**, and separated by commas
  - The array size is automatically set to the number of values in the braces

    int[] age = {2, 12, 1}; //no need to new

- Given age above, age.length has a value of 3

The length Instance Variable

- An **array** is considered to be an **object**
- Since other objects can have instance variables, so can **arrays**
- Every array has exactly one instance variable named **length**
  - When an array is created, the instance variable length is automatically set equal to its size
  - The value of length **cannot be changed** (other than by creating an entirely new array with new)

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

- Given score above, score.length has a value of 5

Square Brackets "[]" with an Array Name

- to create a type name:

  double[] score;

- to create a new array:

  score = new double[5];

- be used to name an indexed variable of an array:

  max = score[0];

Value 1, 2, 3, 4, 5

Index 0, 1, 2, 3, 4

Pitfall: Array Index Out of Bounds

- Array indices always start with 0, and always end with the integer that **is one less than the size of the array(arr.length - 1)**
  - The most common programming error made when using arrays is attempting to use a **nonexistent array index**
- When an index expression evaluates to some value other than those allowed by the array declaration, the index is said to **be out of bounds**
  - An out of bounds index will cause a program to terminate with **a run-time error** message
  - Array indices get out of bounds most commonly at the first or last iteration of a loop that processes the array: Be sure to test for this!

The "for each" Loop

- The general syntax for a **for-each loop** statement used with an array is

  for (ArrayBaseType VariableName : ArrayName)

  Statement

```
int[] arr1 = new int[5];
```

- The above for-each line should be read as "for each VariableName in ArrayName do the following:"
  - Note that **VariableName** must be declared within the foreach loop, not before
  - Note also that a **colon** (not a semicolon) is used after VariableName

Arrays with a Class Base Type

- The **base type** of an array can be a **class type**

```
Date[] holidayList = new Date[20];
holidayList[0] = new Date(xxx);
```

- The above example creates 20 **indexed variables** of type Date
  - It does NOT create 20 objects of the class Date
  - Each of these indexed variables are automatically initialized to **null(placeholder)**
  - Any attempt to reference any them at this point would result in a "**null pointer exception**" error message
- Like any other object, each of the indexed variables requires a separate invocation of a constructor using new (singly, or perhaps using a for loop) to create an object to reference

```
for (int i = 0; i < holidayList.length; i++)
    holidayList[i] = new Date();
```

Use of “==” with Arrays

- For the same reason, the **equality operator (==)** only tests two arrays to see if they are stored in the same location in the computer's **memory**
  - It does not test two arrays to see if they contain the same values (a == b)
  - The result of the above boolean expression will be true if a and b share the same **memory address** (and, therefore, reference the same array), and false otherwise
- Use Arrays.equals to test if they are stored same value

```
Arrays.equals(arr1, arr2)
```

Exercise

```
// Read in the array from keyboard
readArray(numbers);

// Display an array
display(numbers);

// Get maximum value of an array
int max = getMax(numbers);
System.out.println("Max value is: " + max);

// Get the sum of all elements in an array
int sum = getSum(numbers);
System.out.println("Sum is: " + sum);

// Sort array elements in descending order
sortArrayDescendingly(numbers);

// Display an array
display(numbers);

// Find the element with the largest number of appearances
// If there is a tie then return the smaller element
int mostFrequent = getMostFrequent(numbers);
System.out.println("Most frequent value is: " + mostFrequent);
```