Western Engineering

WESTERN UNIVERSITY
ELECTRICAL & COMPUTER ENGINEERING

ES1036B
PROGRAMMING FUNDAMENTALS FOR ENGINEERS

WINTER 2019

Lab06- Arrays

Instructor:
DR. QUAZI RAHMAN

Prepared by:
RILEY BLOOMFIELD

1 Deliverables

To earn full marks for this assignment you must submit the following file(s) on OWL before 11:55PM on the day of your lab session:

- Username_lab06_q1.java
- Username_lab06_q2.java
- Username_lab06_q3.java

where *username* is the beginning part of your email (before @uwo.ca).

It is mandatory that you submit the .java file(s) containing the high-level Java code you have typed. You may see other files types (such as .class) that are generated as you complete the lab but these files will not be accepted.

Additionally, it is **mandatory** that you demonstrate your codes to your TA and obtain their approval before leaving the lab to earn your marks. Codes completed and submitted without a demonstration will not be graded.

Because codes must be demonstrated to earn marks, it is expected that you will enter the lab session with working or almost working codes that can be completed before the end of the session. If you have any difficulties with the labs during the week, you are encouraged to contact your TA for help.

2 Introduction

If any of the instructions presented are not clear, please do not hesitate to contact your lead teaching assistant Riley Bloomfield at rbloomf@uwo.ca.

2.1 Arrays

For detailed instruction on arrays in Java please see lecture unit 5. For specific examples on declaring and accessing arrays, please see slides 5-7, 20, and 39-43.

Up until now we have stored variables in memory one-by-one. For example, if we wish to store an integer in memory, we would declare an integer variable and assign a single value to the new variable.

```
1     int x = 8;
```

This statement has the effect of reserving four bytes in the memory for this single integer variable (Figure 1).

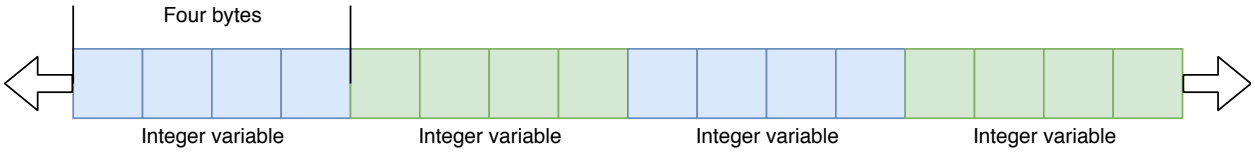

Figure 1: Individually declaring four integer variables in memory. We can think of our computer memory as an infinitely long chain of bits that continues both left and right.

Using a new structure we call an *array*, we can reserve multiple memory locations at the same time. With this larger chunk of reserved memory, we can store multiple variables and access them using the same variable name. The following statement will create an array of 4 integers.

```
int myArray[] = new int[4];
```

We can access each of the four items in the array using the square bracket operators "[]" where the integer value inside the brackets is the index of the element we desire. For example, myArray[0] will give us the first element in the array and myArray[2] will give us the third. Note arrays in Java are **0-indexed** which means the first element is always at the index 0 and the last index is always the array length minus 1 (Figure 2).

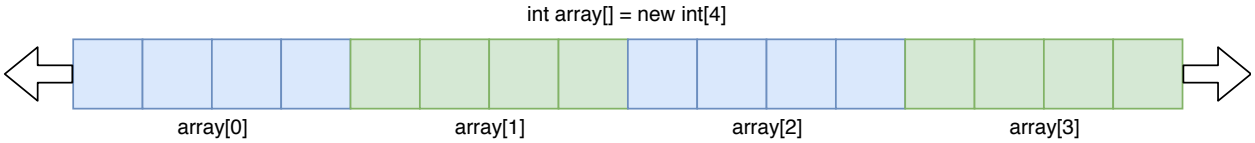

Figure 2: An integer array of four elements in the memory.

We can assign values to each of the array elements individually,

```
myArray[0] = 56;
myArray[1] = 23;
```

and we can also access each array element similarly,

```
System.out.println(myArray[0]);
// Output is 56
```

We can have arrays of different data types other than integers and also we can have arrays of objects. **All elements of the same array must be the same type** and we cannot mix data types or objects in a single array. Consider a problem where a program asks the user to input the names of 10 courses. We could declare 10 individual variables and ask the user to input a name and store it in each variable. What if there were 50 courses? What if we want to print each one? Normally that would mean too many variable declarations and too many print statements.

```
public static void main(String[] args) {
    // Declaring 10 variables
    String name1 = "";
    String name2 = "";
```

```
        // ...
        String name9 = "";
        String name10 = "";

        Scanner scanner = new Scanner(System.in);

        // Asking the user to input the name 10 times
        System.out.println("Please enter the name of course 1: ");
        name1 = scanner.next();

        System.out.println("Please enter the name of course 2: ");
        name2 = scanner.next();

        // ...

        System.out.println("Please enter the name of course 9: ");
        name9 = scanner.next();

        System.out.println("Please enter the name of course 10: ");
        name10 = scanner.next();

        // Calling print statements 10 times to print each course.
        System.out.println("The name of course 1 is " + name1);
        System.out.println("The name of course 2 is " + name2);
        // ...
        System.out.println("The name of course 9 is " + name9);
        System.out.println("The name of course 10 is " + name10);
}
```

Let us instead use an array of strings that can be declared in place of the 10 variable declarations.

```
String names[] = new String[10];
```

Instead of writing statements over and over again to ask the user to input the correct name, we can iterate over the declared *names* array using a loop structure. Note that arrays in Java have a *length* property which tells us the number of elements in the array. This is **not** the same as calling the *length()* method of the string class, since the array property is not a method (no parentheses).

```
for (int i = 0; i < names.length; i++) {
    System.out.println("Please enter the name of course " + i + ": ");
    names[i] = scanner.next();
}
```

Similarly we can iterate over the array once more to print the names of all the courses.

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

```
2       System.out.println("The name of course " + i + " is " + names[i]);
}
```

Take note of how much shorter the code has become after using an array to store the course names. Now, if there are instead 50 courses to be printed instead of 10, we must only change a single number in our code.

```
1 String names[] = new String[50];
```

3 Good Programming Practices

Below are several programming practices that should be followed in order to write and maintain quality codes.

- It is proper convention for class names to begin with a capital letter. Make sure that your class names are always capitalized. The naming convention of the submitted files has been changed, please see the section above (section 1).
- Include comments in your program: This will help you remember what each part of your code does, especially after long breaks from your work.
- Choose meaningful and descriptive names for your variables. There is a balance between descriptive names and code readability but always err on the side of descriptive.
- It is recommended that you follow a lower camel-case naming strategy for variables. Since descriptors cannot contain white-space characters (spaces or tabs) words should be separated by capital letters.
- Initialize variables when declaring them. This means giving them initial values which are easier to track in your program if logical errors are present with your output.
- Indent and properly format your code. You should write your codes so that you and your teaching assistant can read and understand your code easily: You will be marked on this.

4 Lab Assignment Questions

You must complete the three assignment questions below for full credit. For each question, create a new class file named with the appropriate naming convention outlined at the start of this lab.

Question 1 - Introduction to Arrays

4.0.1 Requirement

Write a program that will populate an array and print it out according to the specification below.

4.0.2 Specifications

- At the start of the main method, declare an integer array of size 10.
- Use a for loop to populate the array with values 0 to 9.
- Print the contents of the array on the console using a separate for loop.

4.0.3 Sample Output

Console Output

```
0
1
2
3
4
5
6
7
8
9
```

Question 2 - Dice Roll Statistics

4.0.4 Requirement

Write a program (according to the specifications outlined below) that implements a simple dice statistics program.

4.0.5 Specifications

- For this question, you will be simulating the rolling of dice. This can be accomplished by using the random function from the previous lab. You will use this function to randomly generate a number between 1 and 6 inclusive to simulate rolling dice. Create a method named *rollDice* that returns an integer result of the rolled dice. The function will have the following prototype:
  - public static int rollDice()
- Inside your main method, you must ask the user how many times they would like to roll dice. After storing this integer, create an integer array in the main that can store this many integer values.
- Inside of a loop structure, roll dice the appropriate number of times using the *rollDice()* method and store each result in the integer array.

- Once the array has been populated with values, you must compute and display the following information to the user:
  - The total number of rolls (specified by the user).
  - The number of times each of the values (1-6) were rolled.
  - The mean value of all rolls.
  - The standard deviation of all values.
- The standard deviation of a set can be computed as follows:

$$\sigma = \sqrt{\frac{\sum_{n=1}^{N} (x_n - \bar{x})^2}{N}}$$

  where x_n is the nth element in the integer array, $\bar{x}$ is the mean of all values in the array, and σ is the standard deviation.
- You should round your results to integer precision since dice rolls are whole number entities.

Hints

- You can computer the number of rolls for each integer and the mean of all rolls during array population or by running through the array an additional time and accessing all elements once it has been populated completely.
- The standard deviation computation will require a separate loop over the array because we need to know the mean of all inputs beforehand. On this loop, we must take a summation of the squared difference between each element and the array mean. Once we have this value, we divide it by the number of integers in the array, and take the square root of this value.

4.0.6 Sample Output

Console Output

```
How many dice would you like to roll?
26

Here are the statistics:
1's: 5
2's: 7
3's: 3
4's: 6
5's: 3
6's: 2
Mean: 3
StdDev: 1
```

Question 3 - Simple Card Game

4.0.7 Requirement

Write a program according to the specifications below that allows the user to play a simple card game by guessing if a random card will be higher than the previous.

4.0.8 Specifications

- At the start of your program, you must call a *printHeader()* method from the main to print the mission of this question and your details. This will can be the same method from the previous lab. This method must have the following signature:

```
public static void printHeader(int labNum, int questionNum,
                    String fName, String lName, String mission)
```

  The *printHeader()* method must find which of the string inputs will be longest, and print that many asterisks (*) as the header borders (shown in the sample output). If the combination of the first and last names (with a space in between) is larger than the mission, the name length will determine the number of border characters printed.

- You must then declare an array locally in the main() method. This array must be able to store 52 String objects that will represent card values.

- You must also declare a String array in the main to store 4 characters representing the four suits of a deck of cards. This array should be initialized with the following: ”H”, ”C”, ”D”, ”S”.

- From the main() you must call a populateDeck() method that will populate the array with card values. This method will have the following signature:

```
void populateDeck(String[] deck, String[] suits)
```

- Inside this method, you must iterate over the four suits passed in the suits array and for each value, store a string of a denomination (integer 1-13) concatenated with the suit to create a ”deck” of 52 (13 cards of each suit). After this method is called, the deck array should be populated with all denominations for all suits.

- You must also implement a method named showCard() that will take a string card value as a parameter and return a converted string card value. You must implement some conditional logic or a switch statement inside this method to change:
  - 1’s into A for ace
  - 11’s into J for Jack
  - 12’s into Q for Queen
  - 13’s into K for King

and return a converted card value back to the user. This method will be called every time the raw card values must be displayed to the user. This method will have the following signature:

```
public static String showCard(String cardValue)
```

- You must also create one additional method named *drawCard()* that will take the deck array as a parameter and return a random and *non-repeating* card from the deck. This will be accomplished by generating a random value between 0 and 51 as an array index and fetching the card at that location. We can ensure cards are non-repeating by ”marking” used locations as a special value and checking for this special value upon returning a card. If a random location is chosen but a special marker is retrieved, we can use another random number and continue doing this until a valid card is drawn. This method will have the following signature:

```
public static String drawCard(String[] deck)
```

- In the main method, you must implement a game where a card is drawn randomly using the drawCard method and returned to the main. It will be shown to the user using the showCard method. The user will then be asked if the next card drawn will be higher in value than the previous. Following the user response, another card will be drawn and shown to the user indicating if they were correct or not.
- Your program must ask the user if they would like to play again, and if they choose yes, loop back around, repopulate the deck and proceed to play once more.

Hints

- For more information on declaring arrays, please see lecture unit 6 slides 4-8.
- For examples of accessing a deck of cards using arrays, see lecture unit 6 slides 44-49.

4.0.9 Sample Output

Console Output

```
*****************************
Simple card game using arrays
*****************************
Lab 6
Question 3
Name: User Name
*****************************

The first card drawn was 2H.
Will the next card be higher?
n

The second card drawn was 9S.
Too bad!

Would you like to play again? (y/n)
y

The first card drawn was 8D.
Will the next card be higher?
y

The second card drawn was KD.
You were correct!

Would you like to play again? (y/n)
n

Good bye!
```