



Western
Engineering

WESTERN UNIVERSITY
ELECTRICAL & COMPUTER ENGINEERING

ES1036B
PROGRAMMING FUNDAMENTALS FOR ENGINEERS

WINTER 2019

Lab07- Arrays and Classes

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_lab07_q1.java
- Username_lab07_q2.java
- Username_lab07_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 and also review the previous lab document.

2.2 Two-Dimensional Arrays

Just like we can declare an array of objects in Java, we can also declare an array of arrays. For more details and examples on these 2D arrays, please see lecture unit 5, slides 104-109. See the example below outlining the declaration of a 2D array. Note the identifier "dataType" is a placeholder for any primitive data type or object.

```

1 public class MyClass {
    public static void main(String args[]) {
3         // Declare array ref var
        dataType[][] refVar;

5         // Create array and assign its reference to variable
6         refVar = new dataType[10][10];

9         // Combine declaration and creation in one statement
        dataType[][] refVar = new dataType[10][10];

11        // Alternative syntax
12        dataType refVar[][] = new dataType[10][10];
13    }
14 }

```

Two-dimensional array elements can be accessed similarly to single dimension arrays, but we must index both dimensions:

```

1 System.out.println(refVar[i][j]);

```

2.3 Classes

Up to this point we have created all of our codes in a single Java class. We have learned that we can create multiple methods inside of a class containing local variables. We also know that the *main* method of our main class is the starting point of our programs.

Java is an object-oriented language and this means that we create objects and manipulate them. So far we have reserved primitive type variables such as `int`, `float`, `double`, or `char`, and we know the memory associated with the creation of these variables. Another object we are familiar with is a `String`. We know that strings contain more than a primitive data type variable: we can call methods on a string such as `.length()` or `charAt()`. Strings contain some primitive characters and also methods to mutate or access properties of the string. We refer to strings as objects, and objects are instances of a class.

A class is a blueprint, or template, that specifies the composition of an object.

When the following statement is executed:

```

1 String myString = "This is a string";

```

we are creating a string object and storing its instance in memory. If we look at this statement inside a method:

```

1 public static void main(String []args) {
    String myString = "This is a string";
3 }

```

we must note that memory for the string object named "myString" has been allocated in the heap. A local reference variable is stored on the stack, which indicates the memory location of the heap reservation. To summarize, this statement has the effect of storing a memory address as a local variable (on the stack) and a string object in the heap.

This string object is created using the *String* class, which specifies the structure of the object (its properties) and defines any methods associated with it. Many string objects can be created in our programs but all of them have the same structure (even if each object is a different string). For example, all string objects created have the method *length()* that returns the length of the string. The *String* class has been used as an example but is one of many pre-defined classes. Note that we can also define our own classes to encapsulate primitive variables, other objects, and methods to operate on them.

Please see the following example of a class in Java (note that we do not include *public static void main(String[] args)* as a method since that only should exist in the main class, and not all others we create):

```
1 public class Person {
    String name;
3     int age;

5     Person() {
        name = "";
7         age = 0;
    }

9     void printInfo() {
11        System.out.println("Name: "+name+"\nAge: "+age);
    }
13 }
```

This simple class defines attributes of a person and provides a member method to print the details formatted. We call the class variables *member variables* and the encapsulated methods *member methods*. See the code example below on how to use this class in the *main()* function.

```
1 public class Main {
    public static void main(String [] args) {
3        Person brother = new Person();

5        brother.name = "George";
        brother.age = 42;
7        brother.printInfo();
    }
9 }
```

Console Output

Name: George
Age: 42

In the output above, member variables of the object "brother" were accessed using the dot operator.

2.4 Constructor

When instantiating an object from a class template, there is a special method used to instantiate an object called a *constructor*. This method constructs an object with initial values and always returns an object of the class type. In the *Person* class above, the constructor method is on line 5. The constructor can always be identified because it has two unique properties:

1. It has no return type (it is not void, it is omitted). This is always omitted for a constructor since the method always returns an object instance of that class.
2. It is always named identically to the class name. The constructor for the class *Person* must be named *Person* as well with case sensitivity.

The constructor may take any number of parameters and the constructor method can also be overloaded, meaning a different constructor can be called depending on the arguments passed when instantiating an object. The following example of the *Person* class has two overloaded constructors:

```
1 public class Person {  
    String name;  
3    int age;  
  
5    Person() {  
        name = "";  
7        age = 0;  
        System.out.println("Constructor 1");  
9    }  
  
11   Person(String n, int a) {  
        name = n;  
13        age = a;  
        System.out.println("Constructor 2");  
15    }  
  
17   void printInfo() {  
        System.out.println("Name: "+name+"\nAge: "+age);  
19    }  
}
```

A different constructor will be called for each of the object instantiations below:

```
Person a = new Person();  
2 Person b = new Person("John", 34);
```

Console Output

```
Constructor 1  
Constructor 2
```

If we do not explicitly define a constructor method for our classes, a default one will be called implicitly that will be identical to the following:

```
ClassName() {}
```

This means that we can create new classes without explicitly defined constructors, but we cannot add any member variable initialization or other behaviour we choose to execute when an object is instantiated.

2.5 Creating a new class

In Eclipse, we can add a new class file to our project by right clicking the project package and selecting new class file from the menu.

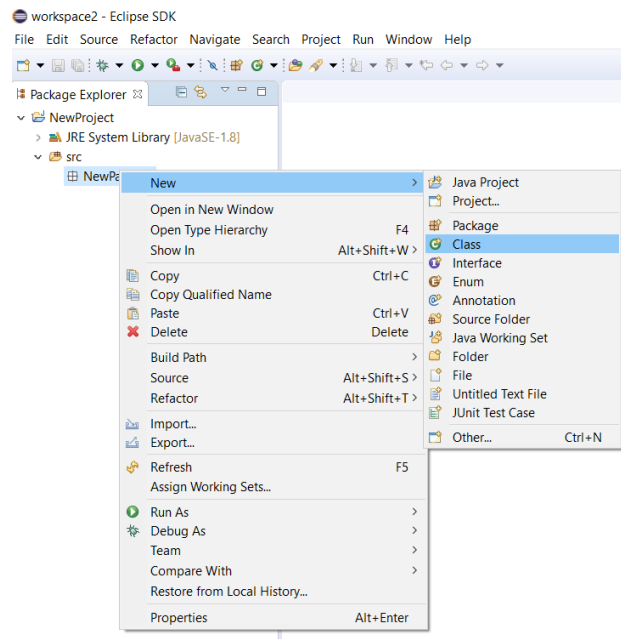


Figure 1: Creating a new Class file

The following screen will allow us to name the class. Note that for this class we do not want to check the box indicating we would like to include a main method.

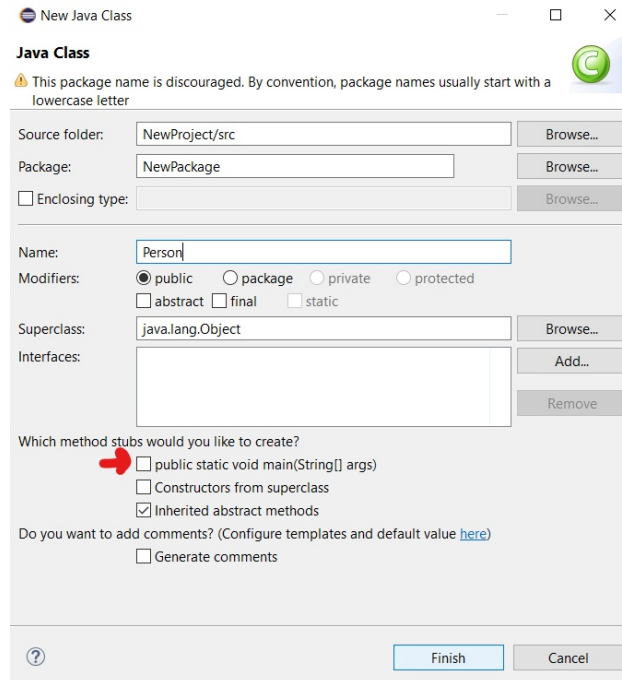


Figure 2: Choosing the class name

The *public* identifier describes the visibility of certain classes or methods from other classes or methods. For now, we have only covered method and classes with public visibility. In Java, only one public class can be declared per class file. It may make sense to try and circumvent this by nesting classes inside each other, but this **is not** a desirable structure for the programs we are trying to create in this course.

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 - Sorting array of strings by length

4.0.1 Requirement

Write a program that will allow a user to input a set of strings and sort them by length.

4.0.2 Specifications

- Begin your program by asking the user how many inputs they wish to sort and create a String array to hold all the inputs.
- From the main, iterate over each array index using a loop and ask the user to input a string on each iteration to assign to that location.
- Once the array has been populated, you must pass the array to a method named "sortLength" that has the following header:

```
1 public static void sortLength(String[] args)
```

- This method will implement a bubble sort algorithm that will sort the array from shortest strings to longest. Once the method has finished sorting the array, you must print the sorted results from the main method. The *sortLength* method may not contain any print statements.
- The bubble sort algorithm is presented in unit lecture 6, slides 94 and 95.

4.0.3 Sample Output

Console Output

```
Welcome! How many strings would you like to sort?
6

Please enter string 1: abcde
please enter string 2: ab
please enter string 3: d
please enter string 4: fedcba
please enter string 5: abcd
please enter string 6: abc

Your sorted array is:
d
ab
abc
abcd
abcde
fedcba

Goodbye!
```

Question 2 - Matrix Operations

4.0.4 Requirement

Write a program (according to the specifications outlined below) that perform matrix operations on fixed 3x3 matrices.

4.0.5 Specifications

- You must create methods in the main class from the headers listed below:

```
1 public static void populateMatrix(double [][] a)
   public static void printMatrix(double [][] a)
3 public static void addMatrices(double [][] a,
   double [][] b, double [][] result)
5 public static void subtractMatrices(double [][] a,
   double [][] b, double [][] result)
7 public static void multiplyMatrices(double [][] a,
   double [][] b, double [][] result)
```

- *printMatrix* will take a previously declared 3x3 array as a parameter and will print its contents on the console.
- *populateMatrix* will take a previously declared 3x3 array as a parameter and will ask the user to populate elementwise with user inputs. You can assume the user will enter real numbers when prompted (see sample output for formatting).
- *addMatrices* will take three previously declared 3x3 arrays and populate the result array with the matrix addition of a and b. Note

$$result[i][j] = a[i][j] + b[i][j]$$

- *subtractMatrices* will take three previously declared 3x3 arrays and populate the result array with the matrix subtraction of a and b. Note

$$result[i][j] = a[i][j] - b[i][j]$$

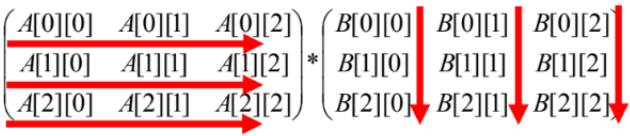
- *multiplyMatrices* will take three previously declared 3x3 arrays and populate the result array with the matrix multiplication of a and b. See the figure (Figure 3) below for matrix multiplication rules.
- In the main method, you must declare three 3x3 arrays to hold inputs and computation results. You must call the *populateMatrix* method twice to fill both computation matrices with user-input values.
- You must then run your program in a loop structure where the following menu is displayed and the user is allowed to choose an appropriate choice:

- a. Add
- b. Subtract
- c. Multiply
- d. Exit

- When a user chooses an option, perform the appropriate operation by calling one of the created methods described above. You **may not** use any print statements inside the *addMatrices*, *subtractMatrices*, or *multiplyMatrices* methods. You must print the result matrix from the main after each operation.
- Operations should not compound, as in, once matrices a and b are populated, they will not be altered during program execution.

Matrix Multiplication – Multiplying a matrix by a matrix:

The result of this multiplication will also be a 3x3 matrix. Consider the following example:

$$C = A * B = \begin{pmatrix} A[0][0] & A[0][1] & A[0][2] \\ A[1][0] & A[1][1] & A[1][2] \\ A[2][0] & A[2][1] & A[2][2] \end{pmatrix} * \begin{pmatrix} B[0][0] & B[0][1] & B[0][2] \\ B[1][0] & B[1][1] & B[1][2] \\ B[2][0] & B[2][1] & B[2][2] \end{pmatrix}$$


To multiply A by B, each row of A is multiplied by each column of B and added together. For example, the **first element** of C, or C[0][0] would be as follows:

$$C[0][0] = A[0][0]*B[0][0] + A[0][1]*B[1][0] + A[0][2]*B[2][0]$$

The following summation is given to help you write a loop to calculate all the elements of $C = A*B$

$$C[i][j] = \sum_{k=0}^2 A[i][k] * B[k][j]$$

Figure 3: Matrix multiplication rules

4.0.6 Sample Output

Console Output

Welcome! Please populate the first 3x3 matrix:

Enter a real number for element (1,1): 1

Enter a real number for element (1,2): 2

Enter a real number for element (1,3): 3

Enter a real number for element (2,1): 4

Enter a real number for element (2,2): 5

Enter a real number for element (2,3): 6

Enter a real number for element (3,1): 7

Enter a real number for element (3,2): 8

Enter a real number for element (3,3): 9

1.0 2.0 3.0

4.0 5.0 6.0

7.0 8.0 9.0

Please populate the second 3x3 matrix:

Enter a real number for element (1,1): 2

Enter a real number for element (1,2): 4

Enter a real number for element (1,3): 6

Enter a real number for element (2,1): 8

Enter a real number for element (2,2): 10

Enter a real number for element (2,3): 12

Enter a real number for element (3,1): 14

Enter a real number for element (3,2): 16

Enter a real number for element (3,3): 18

2.0 4.0 6.0

8.0 10.0 12.0

14.0 16.0 18.0

** Please choose an option: **

a. Add

b. Subtract

c. Multiply

d. Exit

Option: a

Console Output

```
3.0      6.0      9.0
12.0     15.0     18.0
21.0     24.0     27.0
```

```
*****
** Please choose an option: **
*****
a. Add
b. Subtract
c. Multiply
d. Exit
```

Option: b

```
-1.0     -2.0     -3.0
-4.0     -5.0     -6.0
-7.0     -8.0     -9.0
```

```
*****
** Please choose an option: **
*****
a. Add
b. Subtract
c. Multiply
d. Exit
```

Option: c

```
60.0     72.0     84.0
123.0    162.0    192.0
204.0    252.0    300.0
```

```
*****
** Please choose an option: **
*****
a. Add
b. Subtract
c. Multiply
d. Exit
```

Option: d

Goodbye!

Question 3 - Employee Database

4.0.7 Requirement

Write a program according to the specifications below that allows a user to create a simple employee database, populate it with data, and print a report to the console using a custom class.

4.0.8 Specifications

- Create a class called *Employee* with four public member variables, and one public member method, and one constructor.
- The member variables will have the following names and types:

String m_firstName

String m_lastName

int m_ID

int m_salary

The member methods must have the following headers:

```
public Employee(String fName, String lName, int ID, int salary)
```

```
public void printInfo()
```

- The *Employee* constructor will take values for all member variables and assign the local arguments to the member variables accordingly.
- The *printInfo* method of the *Employee* class will simply print all employee details on a comma separated line (See sample output for details).
- In the main method of the main class, ask the user to enter the database size (number of employees) and create an array of Employee objects.
- From the main method, a new method named *enterData* will be called to populate the employee array. Define this method with the method header:

```
public static void enterData(Employee[] database)
```

- Inside this method, assign a first name, last name, ID, and salary to every employee using the member access operator (.). First you must ask the user to input a value for each member variable and assign them appropriately.
- You must validate that the salary is a positive number. We can assume that the user will only enter an integer value when prompted.

- You must create another method in the main class named *printReport* that will be responsible for iterating over all employees in the database (employee array) and calling the `printInfo()` method on each object using the dot operator. This method will have the following header:

```
1 public static void printReport(Employee[] database)
```

Hints

- To iterate over the database, you will need to know its length. The length of an array in Java can be found as follows: `myArray.length`. Note that there are no parentheses because the length is a property of arrays, unlike the *length()* method of the String class.

4.0.9 Sample Output

Console Output

Please enter the number of employees in the database:

3

Employee 1

Please enter the first name:

John

Please enter the last name:

Smith

Please enter the ID:

435

Please enter the salary:

25000

Employee 2

Please enter the first name:

Jim

Please enter the last name:

Taylor

Please enter the ID:

937

Please enter the salary:

24000

Employee 3

Please enter the first name:

Jane

Please enter the last name:

Read

Please enter the ID:

295

Please enter the salary:

25000

Report for all employees:

John Smith, ID: 435, Salary: 25000

Jim Taylor, ID: 937, Salary: 24000

Jane Read, ID: 295, Salary: 25000