

ENSF 409 – Principles of Software Development

Winter 2020

Lab Assignment #1: Introduction to Java

Due Dates	
In-lab	Submit electronically on D2L **before end of lab period (4:00 PM) on Friday January 24**
Post-Lab	Submit electronically on D2L **before 2:00 PM on Friday January 31**

The objectives of this lab are to:

1. familiarize students with the process of writing, compiling and running a simple Java program.
2. implement and use Java classes and objects
3. get familiar with UML basic notations to represent a class
4. practice using array and ArrayList in Java
5. implement a simple linked-list class in Java


The following rules apply to this lab and all other lab assignments in future:

1. Before submitting your lab reports, take a moment to make sure that you are handing in all the material that is required. If you forget to hand something in, that is your fault; you can't use `I forgot' as an excuse to hand in parts of the assignment late.
2. **20% marks** will be deducted from the assignments handed in up to **24 hours** after each due date. It means if your mark is X out of Y, you will only gain 0.8 times X. There will be no credit for assignments turned in later than 24 hours after the due dates; they will be returned unmarked.

How to Compile and Run a Java Program

From Command-line: Once you edited your code using a text editor and saved it in your working directory (e.g `HelloWorld.java`), you can compile your Java program, using the following command on the command line in a terminal window:

```
javac HelloWorld.java
```

This command compiles your source code, links it with program libraries, and creates a byte-code file called: `HelloWorld.class`

To run your program, you should type at the command line: java HelloWorld

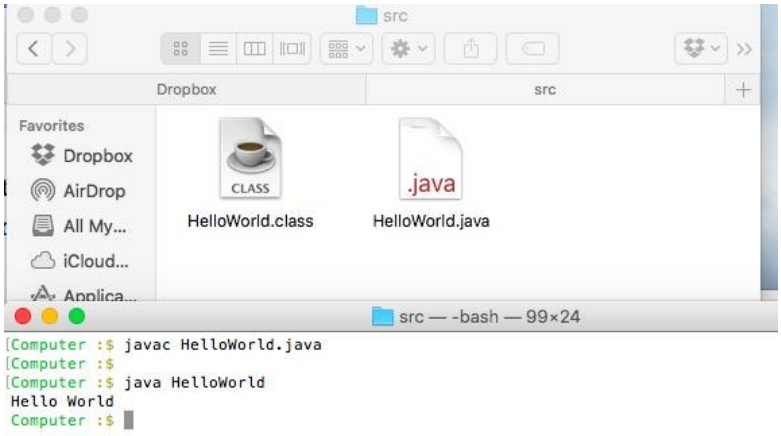

Figure 1-Running a program from command line

Eclipse IDE: In the Eclipse environment, you can run any Java class with a main method using the run command.

Run → Run As → Java Application

Figure 2-Running a program in Eclipse


In Lab (15 marks)

In-Lab Exercise - 1: Formatting Numbers (5 Marks)

There are different ways in Java to format the program numeric output. An easy way is the C-style output formatting, using the `System.out.printf` method:

```
int x = 20;

double y = 2.3;

System.out.printf("%5d%10.2f", x, y);
```

Prints:

```
   20      2.30
```

What to Do: You are expected to write a simple Java program called `AverageCalculator`, with only a main method that receives several numbers from command line. Then the program displays the numbers and the average of the numbers on the screen. Once you wrote and compiled the program you should be able to run the program as follows:

```
java AverageCalculator 2.4 2.6 1.8 1.2
```

This sample run of the program should display:

```
The 4 numbers are: 2.400 2.600 1.800 1.200

And their average is: 2.000
```

The command line arguments in Java are almost similar to C/C++. For the above given example, the data on the command line can be accessed, as illustrated in Figure 3. It means that you can access each argument by using `String` array `args`. For example `args[0]` refers to "2.4". Once you have access to each string of digits you should be able to convert it to numeric values by using statements such as:

```
double x = Double.parseDouble(args[0]);
```

Now `x` holds the numeric value of 2.4.

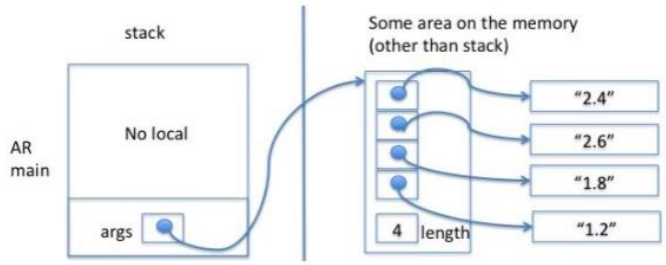

Figure 3-Accessing command line data

Notes: Your program must print the value of the average on the screen, formatted with three digits after the decimal point.

What to Hand in: Submit your .java file.

In-Lab Exercise - 2: One-dimensional Array (10 Marks)

A group of friends decide to run a Marathon. Their names and times (in minutes) are:

Name	**Time(minutes)**
Elena	341
Thomas	273
Hamilton	278
Suzie	329
Phil	445
Matt	402
Alex	388
Emma	275
John	243
James	334
Jane	412
Emily	393
Daniel	299
Neda	343
Aaron	317
Kate	265

What to Do:

Write a method that takes an array of integers as input, and returns the index corresponding to the lowest number (i.e. fastest time). Then, print out the name and time corresponding to the returned index (i.e. the name and time of the fastest runner). Here is a program skeleton to get started:


```
class Marathon {

	//Implement your method

	public static void main (String[] args) {

		String[] names = { "Elena", "Thomas", "Hamilton", "Suzie",
		"Phil", "Matt", "Alex", "Emma", "John", "James", "Jane",
		"Emily", "Daniel", "Neda", "Aaron", "Kate" };

		int[] times = { 341, 273, 278, 329, 445, 402, 388, 275, 243,
		334, 412, 393, 299, 343, 317, 265

		};

	//Call your method here and, then print the name and time of the
	//fastest runner.

	}

}
```

What to Hand in: Submit your Marathon.java file which includes the code for task 1 and task 2.

How to submit: Include all your files for the in-lab section in one folder, zip your folder and upload it in D2L before the deadline.


Post Lab (60 marks)

Post-Lab Exercise- 3: Validate Social Insurance Number (10 Marks)

The Social Insurance Number (SIN) is a nine-digit number that you need to work in Canada or to have access to government programs and benefits. There is an algorithm that allows verifying if a given SIN is valid or not:

1. Add first, third, fifth, and seventh digits. (Note: the first number is the most significant digit)
2. Multiply second digit by two and add the digits of the resultant product together.
3. Repeat step 2 for fourth, sixth, and eighth digits.
4. Add the four terms found in steps 2 and 3.
5. Add the totals from steps 1 and 4 together.
6. 10 minus the least significant digit of step 5 total should be the ninth digit of the SIN.

Example: SIN 366497626

Step 1: 3 + 6 + 9 + 6 =24,
Step 2: 6 * 2 = 12 → 1 + 2 = 3
Step 3: 4 * 2 = 8, → 0 + 8 = 8
7 * 2 = 14 → 1+4=5
2 * 2 = 4 → 0+4=4
Step 4: 3 + 8 + 5 + 4 = 20
Step 5: 24 + 20 = 44
Step 6: 10 – 4 = 6

The result of step 6 matches the last digit in the SIN, therefore, it's a valid SIN. Otherwise, it would be an invalid SIN.

What to Do:

Task 1 - Download the `SinValidator.java` file from D2L. Then, compile and run the program. The program gives incorrect output. Your job is to complete the definition of a method, called `validateSin` -- using the above-mentioned algorithm.

What to Hand in: Submit your modified `SinValidator.java` file.



Post-Lab Exercise- 4: Multi-dimensional Array (10 marks)

What to Do: Write a method that reads three different sentences from the console application. Each sentence should not exceed 60 characters. Then, copy each character of each input sentence in a [3 x 60] character array.

1. The first sentence should be loaded into the first row in the reverse order of characters – for example, "I like programming!" should be loaded into the array as "!gnimmargorp ekil I".
2. The second sentence should be loaded into the second row in the reverse order of words – for example, "I like programming!" should be loaded into the array as "programming! like I".
3. The third sentence should be loaded into the third row where if the index of the array is divisible by 5, and that index corresponds to a letter, then the character is replaced by its corresponding upper case letter, "I like programming!" should be loaded into the array as "I likE proGrammIng!" – that is, characters in index positions 0, 5, 10, 15, and 20 were replaced by upper case letters.

Note that all white space and **special** characters (e.g. ! ? *) are also characters. Also, the index starts from position 0 and the array does not need to be null terminated. Your program must print the contents of the character array on the console.

What to Hand in: Submit your .java file.



Post-Lab Exercise - 5: Marathon code with ArrayList (10 marks)

Modify your code in "Exercise 2" and make your Marathon class to work with `ArrayLists`:

1. Your program must work with two `ArrayLists` instead of the two arrays.
2. The program must receive the names and running times from console and it must work for any number of participants. The number of participants will not be known in the beginning.
3. The program will keep reading the name and running time of participants and when the user enters "quit", it will stop reading inputs and will pass the array list of the running times to the function called `findFastestRunner` that calculates the fastest (minimum) running time.
4. Then, you will output the name and the running time of the fastest runner.

Skeleton of the main method is given below. You need to complete the main method and implement the `findFastestRunner` method.

```
public static void main(String[] args) {
  // Define two array lists here to store the names and the running times

// Read user input
  String sin;
  Scanner scan = new Scanner(System.in);
  while (true)
  {
       System.out.println("Please enter the name of the participant");
       sin = scan.nextLine();
       if(sin.toUpperCase().equals("QUIT"))
             break;
       // Add the name to your ArrayList
       System.out.println("Please enter the running time of the participant");
       sin = scan.nextLine();

       // Add the running time to your array list

  }
  // Call the function findFastestRunner and pass the running times array list to it

  // Print the name of the fastestrunner to the console
}
```


Post-Lab Exercise- 6: Developing and using class and objects (15 marks)

The objective of this exercise is to help you to understand using classes and objects of classes in Java.

What to Do:

Task 1 - Create a class called `Clock.java`. In this file write the definition of a class called `Clock`. This class represents a digital clock that can be used to keep track of elapsed time in terms of number of days, hours, minutes, and seconds. Therefore, class Clock must have four private data fields: `day, hour, minute`, and `second`, and must support the following access methods:

- A set of methods such as `getDay, getHour, getMinute, getSecond`, that allow data fields days, hours, minutes, and seconds to be retrieved.
- A set of setter methods such as `setDay, setHour, setMinute, setSecond`, that allow setting an arbitrary new value for day (any value), hour (0 to 23), minute (0 to 59), and seconds (0 to 59). Errors must be detected in case that user tries to enter an out-of-range value and the data field should remain unchanged.
- In addition to the above access methods, class `Clock` must support two constructors:
    - A constructor that sets the class data fields to the values supplied by the constructor's arguments.
    - A default constructor that sets all the data fields to zero by calling the other constructor.

Class `Clock` **should** also have two other methods as follows:

- A method called `increment` that increments the value of time by multiple seconds with a single call.
- A method called `calculateTotalSeconds` that returns the total time in seconds.

Note: The digital clock is a 24-hour clock (for example 1:3:5 PM is 13:3:5).

A UML representation for class `Clock` is shown in Figure 4.



Clock
Attributes
- int day - int hour - int minute - int second
Operations
+ Clock() + Clock(int day, int hour, int minute, int second) + int getDay() + int getHour() + int getMinute() + int getSecond() + void setDay(int day) + void setHour(int hour) + void setMinute(int minute) + void setSecond(int second) + void increment(int second) + int calculateTotalSeconds()

Figure 4. UML representation of class Clock (Note that "-" means private and "+" means public)

Task 2 - Your class should also have a main method to call the expected functionalities. Here is an example of the required main function that shows most of the functionalities of this class. Write similar test codes for `t2`.

```
public static void main(String[] args) {

  // Create elapsed time with the default values of zeros for day, hour,
  // minute and second.
  Clock t1 = new Clock(); // Default constructor

  // sets hour to 23
  t1.setHour(23);
  // sets day to 1
  t1.setDay(1);
  // sets minute to 59
  t1.setMinute(59);
  // sets day to 16
  t1.setSecond(16);

  // prints: 1:23:59:16
  System.out.println(t1.getDay() + ":" + t1.getHour() +":" + t1.getMinute() + ":" + t1.getSecond());

  // increments time t1 by 44 seconds:
  t1.increment(44);
  // prints: 2:0:0:0
  System.out.println(t1.getDay() + ":" + t1.getHour() +":" +t1.getMinute() + ":" + t1.getSecond());

  // prints the total elapsed time in seconds: 172,800
  System.out.printf("The elapsed time in seconds is: %d", t1.calculateTotalSeconds());

  // REPEAT SIMILAR TESTS FOR t2
  //Elapsed time is 3 days, 1 hour, 4 mins and 5 secs
  Clock t2 = new Clock(3, 1, 4, 5);
}
```



What to Hand in: Submit your Clock class along with the completed main method which tests the functionalities of the class with the `t2` object.

Post-Lab Exercise- 7: Single Linked list (15 marks)

Task 1 - In this exercise, you are going to convert a simple linkedlist program written in C++ to a java program. Using the code, logic and algorithms in three given C++ files [1] (`useSimpleList.cpp, SimpleList.cpp, and SimpleList.h`), create a java program with a single file called `SimpleList.java`. Then, compile and run your program. If the translation of your C++ program to Java is properly done and your program runs with no error, it must produce the following console output:

```
List just after creation -- is empty.
After calling push_front. list must have: 50
50
After calling push_back and set function. list must have: 770 440
770 440
After three more calls to push_back. list must have: 770 440 330 220 110
770 440 330 220 110
After removing two nodes. list must have: 440 330 110
440 330 110
After removing two nodes. list must have: 10 440 33 330 110 40
10 440 33 330 110 40
After 6 removes. list must have: 440 330 40
440 330 40
After call to clear, list must be empty:

After three calls to push_back. list must have: 331 221 111
331 221 111
```

Task 2 - For this exercise, you need to add class, data field, and method comments in your code, following the "Java Documentation Guidelines for ENSF 409" which is also posted on the D2L.

What to Hand in: Submit your completed java code (`SimpleList.java`) along with a zipped file called javaDoc.zip that contains the generated files by the javadoc tool.

How to submit: Include all your files for the post-lab section in one folder, zip your folder and upload it in D2L before the deadline.

[1] The given C++ version of the `SimpleList` is designed for a simple lab exercise for ENSF 409 and doesn't follow the law of Big-3. It may cause problems if you try to make copies of `SimpleList` objects.