EECS 1021 – Lab C: The Util Library in Java (2 week lab)

Dr. James Andrew Smith, PEng and Richard Robinson

This lab takes place in the fourth week of school.

Summary: In this lab, you will be using various java.util classes in addition to basic Java constructs such as arrays and classes.

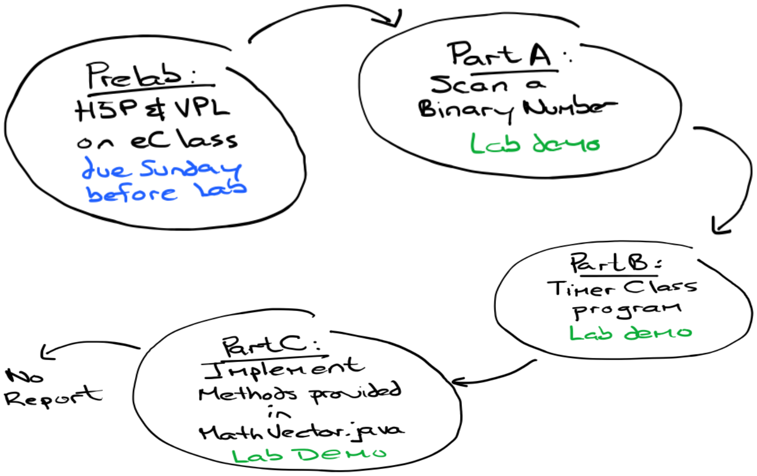


Figure Lab C has a pre-lab, three demos and no report.

Intro

Scanning and printing from users is commonplace in programming applications. So is determining the current time and using pre-built classes. We're going to do all three in this lab.

Due dates.

- *Pre-Lab*: All of the interactive pre-lab activities are due on the Sunday night before the first of the two lab sessions.
- *Lab Demo*:
  - *Option 1*: live lab demo to the TA (Zoom or, starting Feb 7, in-person)
  - *Option 2*: record a screen capture and submit a video to eClass.

Marking Guide:

- All interactive Pre-lab activities are graded out of 1 and count towards your "interactive" activity grade. Any other pre-lab activity is not graded.
- Part A: 0.4 marks. 0.2 if partially successful. 0 if not attempted.
- Part B: 0.2 marks. 0.1 if partially successful. 0 if not attempted.
- Part C: 0.4 marks. 0.2 if partially successful. 0 if not attempted.

Pre-lab

Check for pre-lab activities in Module 3. These could be either H5P activities or VPL activities or both. All pre-lab activities are due on the Sunday before the first of the two labs (week 1 and 2) at 11:55pm.

Part A: Binary Sum and java.util

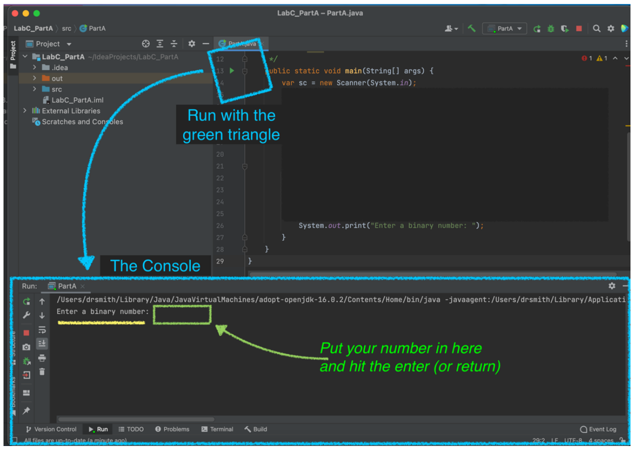
In this lab, you will be using various java.util classes in addition to basic Java constructs such as arrays and classes.

Java.util is a package in the Java library.[1]

For this part of the lab, your goal is to implement a program which continuously accepts a binary number from the input, and outputs the sum thus far (also in binary).

An example usage of this program is as follows:

1. User input	**Enter a binary number:** 101
• print	Running sum: 101
2. User input	**Enter a binary number:** 11
• print	Running sum: 1000
3. User input	**Enter a binary number:** 1010
• print	Running sum: 10010
4. User input	**Enter a binary number:** 11
• print	Running sum: 10101
5. …	**Enter a binary number:**
Action sequence	Text in console



```
 */
public static void main(String[] args) {
    var sc = new Scanner(System.in);

        System.out.print("Enter a binary number: ");
    }
  }
}
```

```
/Users/drsmith/Library/Java/JavaVirtualMachines/adopt-openjdk-16.0.2/Contents/Home/bin/java -javaagent:/Users/drsmith/Library/Applicati
Enter a binary number:
```

Figure 1 Run your program in the console found in IntelliJ. This can also be done in jShell.

Specification

1. The program should print: "Enter a binary number: "
2. The program should read a binary number from the standard input
3. The sum of all numbers entered so far should be printed to the standard output, in binary, in the format `"Running sum: "` followed by the value.
4. Steps 1 through 3, inclusive, should be repeated indefinitely until the program is stopped.

```
package eecs1021;

import java.util.Scanner;

public class PartA {
  /**
   * Continuously reads binary numbers from the user and prints the running sum in binary.
   *
   * An input of 101, 11, 1010, 11 will yield outputs of 101, 1000, 10010, and 10101
   *
   * @param args
   */
  public static void main(String[] args) {

            /* fill it in here */
  }
}
```

PartA.java

[1] Reference on packages in the Java Library: https://www.w3schools.com/java/java_packages.asp

Procedure

1. Create a Scanner object from the System.in stream.
    - Hint: var scanObject = new Scanner(Syst…. (go back to Lab A)
2. Set the radix of the scanner to 2.
    - Hint: take your scanner object and add .useRadix(Put_A_Value_In_Here) method to it
3. Print "Enter a binary number: " to the console.
    - Hint: remember sout in IntelliJ?
4. Create a while loop which repeats as long as the scanner has a next line.
    - Hint: inside the brackets of while(), put your scanner object and attach the .hasNext() method to it. This method will return 1 to the while as long as there is something to scan.
5. Retrieve the read int from the scanner, and add it to the running sum.
    - Hint: Do this inside the while loop body (inside the curly braces)
6. Print the running sum to the console, in binary, in the format "Running sum: " followed by the value.
    - Hint: Use Integer.toBinaryString(variable_to_be_converted) here inside a println().
7. Print "Enter a binary number: " to the console again before repeating the loop.
    - Hint: just a println().

```
start of Class {
    start main method {
        create scanner object
        tell scanner object to use radix 2
        tell user to enter #
        create counter w̄ ∅ value
        while (check .hasNext() on scanner object) {
            make variable which captures a scanned
              integer using .nextInt() method
            increment counter with that value
            println w̄ a concatenated String that includes
              Integer.toBinaryString(your counter)
            println "Enter a binary number: "
        }
    }
```

Regular #'s (Radix 10)	Binary #'s (Radix 2)
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
13	1101
14	1110
15	1111
16	10000
17	10001
18	10010
19	10011
20	10100
⋮	⋮

Part B [0.2 marks]

For this part, your goal is to become familiar with the Timer class.

Specification

1. Every 1000 milliseconds, the program should print the current time (in ms) to the console, and the value of a counter which increments with every iteration.

Procedure

1. Create a Timer object.

2. Use the schedule method of the Timer object to schedule a task to be executed every 1000 milliseconds.

   1. The task should contain a variable to keep track of the number of iterations.

   2. Within the task's run method, print the current time (in ms) to the console, and the value of the counter to the console.

Procedure

```
class {
  method {
    create Timer() object
    schedule a timer task {
      init a counter
      @Override
      create a run() method {
        print System.currentTimeMillis
        print counter & increment
      }
    }, delay , period);
  }                ↑ parenthesis for schedule
}
```

Structure of the class & method

```
package eecs1021;

import java.util.Timer;
import java.util.TimerTask;

public class PartB {
  public static void main(String[] args) {


            /* fill it in here */


  }
}
```

PartB.java

Part C [0.4 marks]

In this part, you will be implementing several methods of the MathVector class provided to you. You'll get a good feel for these by doing the pre-lab exercises.

This exercise is free-form. Be creative. Use all the methods within MathVector as listed below.

Specification

1. Each method's specification is provided in its corresponding Javadoc comments.
2. Implement
   a. toString, [0.05 marks]
   b. add, [0.05 marks]
   c. magnitude [0.05 marks]
   d. at [0.05 marks]
   e. parse [0.05 marks]
   f. random, and [0.05 marks]
   g. filled [0.05 marks]

If you implement all of them you get a bonus 0.05: 0.4 out of 0.4 marks.

Procedure

1. It's up to you to figure out the implementation of the methods!

```
package eecs1021;

import java.util.Random;
import java.util.StringJoiner;

/**
 * A class representing a mathematical vector.
 */
class MathVector {
  private final int[] array;

  /**
   * Private constructor. {@code array} is set to an empty array of the given size.
   *
   * @param size the size of the array
   */
  private MathVector(int size) {
    this.array = new int[size];
  }

  /**
   * Creates a MathVector instance backed by the given array.
   *
   * @param source the array to use
   */
  public MathVector(int[] source) {
    this.array = source;
  }

  /**
   * Static method to create a new MathVector instance with the specified {@code size}, with each element set to the specified {@code value}.
   *
   * @param size  the number of elements in the new vector
   * @param value the value to set each element to
   * @return a new MathVector instance
   */
  public static MathVector filled(int size, int value) {
    var result = new MathVector(size);
    for (int i = 0; i < size; i++) {
      result.array[i] = value;
    }
    return result;
  }

  /**
   * Static method to create a new MathVector instance with the specified {@code size}, with each element set to a random value in the range [min, max).
   *
   * @param size the number of elements in the new vector
   * @return a new MathVector instance
   */
  public static MathVector random(int size, int min, int max) {
    var result = new MathVector(size);
    var rng = new Random();

    for (int i = 0; i < size; i++) {
      result.array[i] = rng.nextInt(max - min + 1) + min;
    }
    return result;
  }

  /**
   * Static method to create a new MathVector instance from the String {@code s}, whose format should be "x1,x2,x3,...", where xi is the ith element.
   *
   * @param s the string to parse
   * @return a new MathVector instance
   */
  public static MathVector parse(String s) {
    var split = s.split(",");

    var result = new MathVector(split.length);

    for (int i = 0; i < split.length; i++) {
      result.array[i] = Integer.parseInt(split[i]);
    }

    return result;
  }

  /**
   * Returns the element at the specified index.
   *
   * @param index the index of the element to return
   * @return the element at the specified index
   */
  public int at(int index) {
    return array[index];
  }

  /**
   * Returns the euclidean distance of this vector.
   *
```

```
     * @return the euclidean distance of this vector
     */
    public double magnitude() {
        double sum = 0;
        for (var e : array) {
            sum += Math.pow(e, 2);
        }
        return Math.sqrt(sum);
    }

    /**
     * Returns a new MathVector instance that is the sum of this vector and the specified vector. (ie, each element is added together)
     *
     * @param other the other vector
     * @return a new MathVector instance that is the sum of this vector and the specified vector
     */
    public MathVector add(MathVector other) {
        var result = new MathVector(array.length);
        for (int i = 0; i < array.length; i++) {
            result.array[i] = array[i] + other.array[i];
        }
        return result;
    }

    /**
     * Returns a String representation of this vector. The String should be in the format "[1, 2, 3]"
     *
     * @return a String representation of this vector
     * @apiNote **DO NOT** use the built-in {@code Arrays.toString()} method.
     */
    @Override
    public String toString() {
        var sj = new StringJoiner(", ", "[", "]");
        for (var e : array) {
            sj.add(String.valueOf(e));
        }

        return sj.toString();
    }
}
```

MathVector.java