BBM 104 – Introduction to Programming Lab.
Lab 1 – Introduction to Java on Eclipse Platform
Math with Java[1]

Developed for the Spring 2017 term by Res. Assist. Dr. Ahmet Selman Bozkır

Submission:

- **The HelloJava.java**
- **Lab report (report.pdf)**

Table of Contents

1 Introduction 1
1.1 Learning Objectives 1
1.2 Marking Scheme 1
2 An Overview of the Application 2
2.1 Theory 2
2.1.1 Riemann Sum 2
2.1.2 Maclaurin Series for approximation of Arcsinh(x) 3
2.1.3 Armstrong Numbers 3
2.2 Software usage 3
2.3 Details of the commands 3
2.4 Functions 4
3 File read with scanner 4
4 Sample Input & Output 5
5 Special notes 5

1 INTRODUCTION

1.1 LEARNING OBJECTIVES

The learning objective of this lab is to let student warm up for Java programing language on Eclipse platform. Furthermore, you will have a chance to implement some mathematical operations at Java using recursive and iterative programming approaches. Additionally, you will learn how to read a file and parse it in java 1.8. Throughout the experiment, we will use Eclipse Neon IDE (Integrated Development Environment) for coding activities. Other environments are not being accepted for this assignment.

1.2 MARKING SCHEME

The marking scheme is shown below. Evaluation of code section will involve different aspects. Moreover, the report for your assignment must be submitted along with your code file.

[1] Required environment: Eclipse Neon2 + JDK 8

Component	Mark %
Code file evaluation • Short main function implementation • Comments for code sections • Concordance to variable name convention • Correct results	80%
Lab report evaluation • Aim • Software usage • Sample results • Personal comments including: - difficulties you have faced - what you have learned from this assignment	20%
Total	100%

Note: In your solutions, your algorithm and code should be as simple as possible. Algorithms, which are more complicated than what should be, will get reduced marks.

2 An Overview of the Application

In this assignment, we will develop a very simple application which do some mathematical calculations. Briefly, you will write a Java application which covers the following operations:

- Integral computation via middle Riemann sum with in a range
- Numerical approximation of arcsinh(x) function by Maclaurin series
- Finding the Armstrong (narsist) numbers within a range

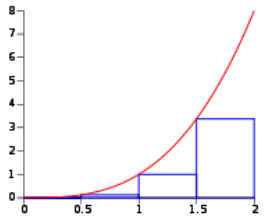
In order to make these operations, you will read your input.txt (commands) file. Remember that, this file will be provided as a command argument and the name of the file may vary. You can use Scanner object to read your file. Information about using command arguments and file I/O via Scanner object will be given in next sections.

2.1 Theory

In this section, details of the commands along with their theory have been provided.

2.1.1 Riemann Sum

Another numerical method for approximating an integral is Riemann sum which has been named after German mathematician Bernhard Riemann. The sum is calculated by dividing the region up into shapes (rectangles, trapezoids, parabolas, or cubics) that together form a region that is similar to the region being measured, then calculating the area for each of these shapes, and finally adding all of these small areas together. This approach can be used to find a numerical approximation for a definite integral even if the fundamental theorem of calculus does not make it easy to find a closed-form solution [2].



Left

Let f : $D \rightarrow \mathbf{R}$ be a function defined on a subset, D, of the real line, R. Let $I = [a, b]$ be a closed interval contained in D, and let

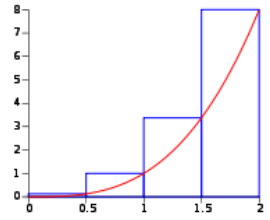
$$P = \{[x_0, x_1], [x_1, x_2], \ldots, [x_{n-1}, x_n]\},$$

be a partition of I, where

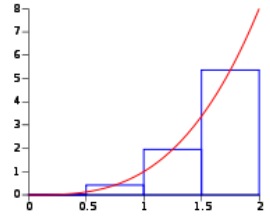
$$a = x_0 < x_1 < x_2 < \cdots < x_n = b.$$

A Riemann sum of f over I with partition P is defined as

$$S = \sum_{i=1}^{n} f(x_i^*)(x_i - x_{i-1}), \quad x_{i-1} \leq x_i^* \leq x_i$$



Right



Middle

Notice the use of "a" instead of "the" in the previous sentence. This is due to the fact that the choice of x_i^* in the interval $[x_{i-1}, x_i]$ is arbitrary, so for any given function f defined on an interval *I* and a fixed partition P, one might produce different Riemann sums depending on which x_i^* is chosen, as long as $x_{i-1} < x_i^* < x_i$ holds true.

- If $x_i^* = x_{i-1}$ for all *i*, then *S* is called **left** Riemann sum
- If $x_i^* = x_i$ for all *i*, then *S* is called **right** Riemann sum
- If $x_i^* = \frac{1}{2}(x_i + x_{i-1})$ for all *i*, then *S* is called **middle** Riemann sum

In this experiment, you are supposed to employ middle Riemann sum rule.

2.1.2 Maclaurin Series for approximation of Arcsinh(x)

Maclaurin Series are special cases of Taylor Series where the series are centered at zero. Each of these series are sum of infinite number of terms and used for calculating a special function.

For this operation, you will implement arcsinh(x) function as Maclaurin Series. For function arcsinh(x) value of "x" will be given by user. After calculating result using formula above, you will print it to the console. Although the formula contains infinite number of terms, you need to add only first 30 terms. This will give you a precise enough result. Keep in mind that the formula is defined in $|x| < 1$

$$arcsinh(x) = \sum_{n=0}^{\infty} \frac{(-1)^n (2n)!}{4^n (n!)^2 (2n+1)} x^{2n+1} \quad \text{for } |x| < 1$$

2.1.3 Armstrong Numbers

In recreational number theory, a narcissistic number (also known as an Armstrong number, after Michael F. Armstrong) is a number that is the sum of its own digits each raised to the power of the number of digits [3]. For example 407 and 153 are both Armstrong numbers since,

$407 = 4^3 + 0^3 + 7^3$

$153 = 1^3 + 5^3 + 3^3$

In this assignment one of your task is to find out and print the Armstrong numbers whose digit number is below or equal to the given digit parameter (e.g 3, 5)

2.2 SOFTWARE USAGE

Your program will be a console application and read the input file. For each line there will be command to be executed. Upon execution, result of each command must be printed to console by using `System.out.println` function. Keep in mind that, your input file will be located at the same directory where .classpath and .project files are being located.

Upon clicking the "Run" button, it must immediately execute/output the result of the listed commands and exit without any prompt or user interaction. Note that there is no limit for the number of commands and they will be syntax-error free. Moreover, each command is delimited by the character of whitespace.

2.3 DETAILS OF THE COMMANDS

In this experiment, your input file will have 4 kinds of commands with varying number of parameters. For each command, output must involve both command (with parameters) and its result. Please refer to the table below for the details.

1. IntegrateRiemann

Command syntax:
```
IntegrateRiemann name_of_function a b number_of_partitions
String               String           int int int
```

Sample Input:
```
IntegrateRiemann Func2 2 7 100
IntegrateRiemann Func3 0.2 0.5 10
```

Sample Output:
```
IntegrateRiemann Func2 2 7 100 Result: 126.84545266999328
IntegrateRiemann Func3 0.2 0.5 10 Result: 0.10264104641331061
```

2. Arcsinh

Command syntax:
```
Arcsinh value
String  double
```

Sample Input:
```
Arcsinh 0.8
```

Sample Output:
```
Arcsinh 0.8 Result: 0.73298321768061
```

3. Armstrong

Command syntax:
```
Armstrong value
String    int
```

Sample Input:
```
Armstrong 3
```

Sample Output:
```
Armstrong 3 Result: 0 1 2 3 4 5 6 7 8 9 153 370 371 407
```

2.4 FUNCTIONS

For the numerical method based integrations you will use 3 different functions (Func1, Func2 and Func3). Their formulas are given below. Keep in mind that you must implement these functions as hardcoded. For the Func3, you have to use your own arcsinh(x) function.

Func1(x) = $x^2 - x + 3$	Func2(x) = $(3\sin(x) - 4)^2$	Func3(x) = arcsinh(x) $\lvert x\rvert < 1$

3 FILE READ WITH SCANNER

There exist various ways to read a text file in Java. However, in this experiment you are supposed to use the most basic one. `Scanner` is a simple text scanner which can parse primitive types and strings using regular expressions. A `Scanner` breaks its input into tokens using a delimiter pattern, which by default matches whitespace. The resulting tokens may then be converted into values of different types using the various next methods. Since you are a beginner in Java, the example code has been given below.

```
public static void main(String[] args) {
    try {
         Scanner scanner = new Scanner(new File(args[0]));
         while(scanner.hasNextLine()){
               String line = scanner.nextLine();
               // do something with your line
          }
          scanner.close();
        }
        catch (FileNotFoundException ex) {
            System.out.println("No File Found!");
            return;
        }
}
```

4 SAMPLE INPUT & OUTPUT

Input.txt

```
IntegrateReimann Func1 2 7 10
IntegrateReimann Func1 2 7 100
IntegrateReimann Func2 2 7 10
IntegrateReimann Func2 2 7 100
IntegrateReimann Func3 0.2 0.5 10
Arcsinh 0.8
Armstrong 3
```

Output window

```
IntegrateReimann Func1 2 7 10 Result: 104.0625
IntegrateReimann Func1 2 7 100 Result: 106.43190624999957
IntegrateReimann Func2 2 7 10 Result: 126.77543832897483
IntegrateReimann Func2 2 7 100 Result: 126.84545266999328
IntegrateReimann Func3 0.2 0.5 10 Result: 0.10264104641331061
Arcsinh 0.8 Result: 0.73298321768061
Armstrong 3 Result: 0 1 2 3 4 5 6 7 8 9 153 370 371 407
```

5 SPECIAL NOTES

- Send your submission via a zip file. The zip file must contain your code "HelloJava.java" and "report.pdf". On the other hand, name your zip file with your student number.
- Ensure that your code is running without any problem. Complete crashing will be marked with 0 point. Partial crashing will be marked by checking the runnable parts.
- Cheating/plagiarism is completely prohibited. Your codes will be checked via a special and scientific software against plagiarism. If your code or report exceeds a certain threshold, you and the person you have copied the code will be punished according to the rules of Hacettepe University Student & Cheating Policy
- Due date is 8.3.2016 23:59 GMT+2