Assignment 2

This assignment has three parts. Part 1 is a practice of informal running time analysis. Part 2 is a practice of designing and implementing a small ***recursive*** method. Part 3 is an implementation of an application using the stack data structure.

Part 1 (20 points)

Consider the following five methods:

```
public static int example1(int[] arr) {
   int n = arr.length, total = 0;
   for (int j=0; j < n; j++)
     total += arr[j];
   return total;
 }

 public static int example2(int[] arr) {
   int n = arr.length, total = 0;
   for (int j=0; j < n; j += 2)
     total += arr[j];
   return total;
 }

 public static int example3(int[] arr) {
   int n = arr.length, total = 0;
   for (int j=0; j < n; j++)
     for (int k=0; k <= j; k++)
       total += arr[j];
   return total;
 }

 public static int example4(int[] arr) {
   int n = arr.length, prefix = 0, total = 0;
   for (int j=0; j < n; j++) {
     prefix += arr[j];
     total += prefix;
   }
   return total;
 }

 public static int example5(int[] first, int[] second) { // assume equal-length arrays
   int n = first.length, count = 0;
   for (int i=0; i < n; i++) {
     int total = 0;
     for (int j=0; j < n; j++)
       for (int k=0; k <= j; k++)
         total += first[k];
     if (second[i] == total) count++;
   }
   return count;
 }
```

Express the running time of each method using the *big-Oh* notation. You need to justify your answers. Your justification doesn't have to be formal (or mathematical) but must be logically

correct. If you show only answers without justification, no points will be given even though your answers are correct.

Part 2 (30 points)

Create a file named *Hw2_Part2.java* and design and implement a recursive algorithm that performs the following:

The algorithm, named *rearrange*, receives an array of integers and rearrange them in such a way all odd integers appear before all even integers. The order of integers is not important. The following is an example of an input array and the output array of the method:

Input: [10, 15, 20, 30, 25, 35, 40, 45]
Output: [15, 25, 35, 45, 10, 20, 30, 40]

Note that you may not use an additional array and the rearrangement of integers must occur within the given array *a*.

The algorithm *rearrange* must be implemented as a Java method and its signature must be:

```
public static void rearrange(int[] a)
```

If needed, you may write a separate method with additional parameters (refer to page 214 of the textbook).

An incomplete code of *Hw2_Part2.java* is posted on Blackboard. It contains a main method and you can use it to test your implementation of the algorithm.

Part 3 (50 points)

You are required to write a Java program named *Hw2_Part3.java* that evaluates and produces the value of a given arithmetic expression using a stack data structure. For example, if the given expression is (2 + (5 * 3)), your program must produce 17. As another example, consider the expression ((4 – 2) * ((9 – 3) / 2)). Given this expression, your program must produce 6.

We make the following assumptions about the input expression:

- All operands are positive integers.
- Only the following binary operators are allowed: +, –, *, and / (integer division)
- Expression is fully parenthesized. In other words, each operator has a pair of "(" and ")" surrounding its two operands.

Your program must read a number of arithmetic expressions from an input file, named *expressions.txt*, and produce an output. A sample input file is shown below:

((2 + 5) * 3)
(((3 + 2) * 3) – (6 / 2))
(((7 – 5) * 2) / (10 – 8))

Each line has one arithmetic expression. Operands, operators, and parentheses are separated by a space. The expected output for the above input is:

The value of ((2 + 5) * 3) is 21
The value of (((3 + 2) * 3) – (6 / 2)) is 12
The value of (((7 – 5) * 2) / (10 – 8)) is 2

Note that a given expression is repeated in the corresponding output. You must write the output on the screen.

A pseudocode is given below. You must implement this pseudocode.

For each expression in the input file, perform the following:

Create two stacks – one for operands and the other for operators

Scan the tokens in the expression from left to right and perform the following:

- If a token is an operand, push it to the operand stack.
- If a token is an operator, push it to the operator stack.
- If a token is a right parenthesis, pop two operands from the operand stack and pop an operator from the operator stack and apply it to the operands. The result is pushed back to the operand stack.
- If a token is a left parenthesis, ignore it.
- After all tokens are processed, what is left in the operand stack is the result.

(Repeat the same process for all expressions in the input file.)

We assume that all expressions in the input file are valid (i.e., you don't need to check for invalid expressions).

For the two stacks, you must use the ***LinkedStack.java*** class that is included in the textbook's source code collection. You may not use Java's stack class or a stack class defined by somebody else. Note that you must not modify the ***LinkedStack.java*** class.

Documentation

No separate documentation is needed. However, you must include sufficient inline comments within your program.

Deliverables

You need to submit the following files:

- *Hw2_Part1.pdf* (answers to part 1)
- *Hw2_Part2.java*
- *Hw2_Part3.java*
- All other necessary files, including:
    - *LinkedStack.java*
    - *Stack.java*
    - *SinglyLinkedList.java*
    - *Other files, if any*

Combine all files that are necessary to compile and run your program into a single archive file. Name the archive file *LastName_FirstName_hw2.EXT*, where *EXT* is an appropriate file extension, such as *zip* or *rar*. Then, upload it to Blackboard.

Grading

Part 1: Up to 5 points will be deducted for each code segment if your answer is wrong.

Part 2: Your facilitator will run your program with 3 different input arrays and up to 6 points will be deducted for each wrong output.

Part 3: Your program will be tested with an input file with 5 input expressions and up to 6 points will be deducted for each wrong output.

Points will be deducted up to 20 points if your program does not have sufficient inline comments.