Assignment 2
Coin Toss and Recursion

CSIS 3475

Assignment

- Download **Assignment 2.zip** and import it into an Eclipse workspace using the standard instructions.
- Finish the Coin Toss project
- Finish the Recursion project.
- Submit the completed projects using the standard submission instructions
    - Export the projects to a zip archive named **Assignment 2 YourName.zip** where YourName must be your first initial and last name.
    - You MUST use the submission instructions exactly or you will lose marks.
    - You MUST name the archive correctly or you will lose marks.
    - For example, for Michael Hrybyk
        - Assignment 2 MHrybyk.zip

Coin Toss

- Create a list of coins, which are tossed and are either randomly heads or tails.
- Show the coin list, then move all of the coins which are heads to a new list, leaving the rest (tails) in the original list. Then display each list, and the summary statistics including number of coins in each list, and their total value (in cents).
- Coins are the basic Canadian coins.
  - Coin class is provided with complete implementation. Do not touch this.
- To finish the project
  - Implement the ListInterface ADT using
    - AList
    - LListUsingLibraryLinkedList
    - LList
    - LListWithTail
  - Complete the methods in CoinToss
    - The template code relies on ListInterface throughout.
    - Test each ListInterface implementation by commenting out appropriate lines
  - Test each implementation against Output.txt.
    - The output of your programs should match the output **exactly**.

Implement ListInterface classes

- ListInterface is provided for you. It is exactly the same as in the textbook and lectures.
  - Every class must implement all methods.
- Classes are provided but incomplete
  - Classes to be implemented
    - AList
    - LListUsingLibraryLinkedList
    - LList
    - LListWithTail
  - You must implement all methods for each.
  - You may need to create private methods as needed.

ListInterface implementations

- Constructor and helper methods are already provided for all implementations
- AList
    - MUST use an array, NOT use links and nodes
    - MUST increase size of array used dynamically as list grows.
- LListUsingLibraryLinkedList
    - MUST use the Java library ArrayList and LinkedList, NOT use internal links and nodes.
    - Comment out the object not being used (ArrayList or LinkedList) for testing.
- LList
    - Must use links, NOT an array
    - Must use a single nextNode link for each node, and keep track of ONLY the head of the list (firstNode).
    - You must use the external Node class provided.
- LListUsingTail
    - Similar to above, but you must also now use a tail reference (lastNode).

CoinToss program

- Complete the methods in CoinToss and test your code against Output.txt.
  - Comment out the ListInterface classes not being used
  - Output must match exactly.
  - You must code initializeCoins(), moveCoins(), showCoins(), and getTotalValue();

Recursion

- You must implement versions of a factorial and Fibonacci sequence. ~ show recursively.
  - Implement an iterative, recursive, and tail recursive version of each.
- The RecursionDemo program is already done
  - For each implementation, the program displays the result of the sequence and the number of recursive calls made.
  - Do not touch this code. It is a simple test harness.
  - Note the DisplaySequence() method simply uses the methods from the SequenceInterface.
  - The output of the program must match Output.txt exactly.

Math background

- Assume that everyone knows the definition of factorial.
  - Research this if you do not.
  - You must use iteration and recursion to calculate.
- The definition of a Fibonacci sequence is given in the comments of each file.
  - See also
    - https://en.wikipedia.org/wiki/Fibonacci_number
    - https://www.mathsisfun.com/numbers/fibonacci-sequence.html
  - Do NOT use the golden ratio which is O(1).
  - You must use iteration and recursion to calculate.

SequenceInterface

- Each class you create must implement the SequenceInterface.
- The classes already have the constructor and helper methods created.
- The compute() method returns the result of the sequence evaluation of the argument. The method must also count the number of calls (recursive) that are made.
  - Note that for iterative implementations, the number of calls is always 1.
- Other methods get or reset the number of calls.
- You must implement the compute method for each type of sequence required.

Sequences

- You must complete the code for the following classes, which calculate factorial or Fibonacci number.
  - You need to complete the compute() method.
  - All other methods and the constructor are provided.
- Factorial
  - IterativeFactorial – must use only an iterative approach
  - RecursiveFactorial – must use only simple recursion
  - TailRecursiveFactorial – must use tail recursion
- Fibonacci – similar to Factorial above
  - IterativeFibonacci – must use only an iterative approach
  - RecursiveFibonacci – must use only simple recursion
  - TailRecursiveFibonacci – must use tail recursion
- Note that the number of calls for iterative is always 1.
- Note that the number of calls for tail recursion is significantly less than normal simple recursion.

Grading

Item	Marks
Project properly named and submitted	0.2
All code properly formatted and commented	0.2
List ADT Implementations	1.0
Coin Toss main program	1.2
Factorial implementation	1.2
Fibonacci implementation	1.2
Total	5.0