COP3538
Project 4 – Binary Tree
100 Points

Submission Requirements

- Submit your project folder via the FileUploader tool provided on the website
  - Follow the project submission guidelines for the class

Design Documentation Requirements

- No design documents are required for this project.

Input File Requirements

- States.Input.A.txt (A set of state data to load into the program in CSV format)
- States.Input.B.txt (A set of state data to load into the program in CSV format)
- States.Delete.txt (A list of state populations to load into the program)

Output File Requirements

- No output files are required for this project.

Miscellaneous Files

- Linkable.java (Provides the method definitions required to implement a node)
- Stackable.java (Provides the method definitions required to implement a stack)
- States.Description.txt (Provides a description of the layout of the required States.Input files)
- States.Delete.Description.txt (Provides a description of the layout of the States.Delete input file)
- Treeable.java (Provides the method definitions required to implement a Binary Tree)

Design Specification Requirements

Note: Refer to the sample output in the **Example Output** section below.

1. Create 6 classes
   A. The project class
      1. This is the class that contains the **public static void main(String[] args)** method
      2. This class should only do three things
         Note: Use command-line arguments to pass the input files to the program.
         a. Display programmer name(s) and project title on separate lines, followed by a blank line
         b. Create an instance of the Driver class
         c. Call the Driver.execute method
            1. Pass the arguments variable (**args**) to the execute method
   B. The **Driver** class
      1. Must contain an execute method that performs the following
         a. Initialize the program
            1. Create an instance of the Stack class
               a. The stack **MUST** store State objects in a singly-linked list
            2. Create an instance of the BinaryTree class
            3. Create an integer array to hold 17 values
         b. Read data into the program using the two data files
            1. Read state records from **both** input data files simultaneously

        2. Create an object instance of the State class for each state record
        3. Determine which State object to push to the stack first
        4. Continue until all the state records are read from both files
    c. Display the state data stored in the stack
        1. Use the same format as project 1 to display the state data
    d. Transfer the state objects from the stack to the binary tree
        1. Pop each state object from the stack
        2. Insert the state objects into the binary tree
            a. Order the state objects by their **population**
    e. Display the state data stored in the binary tree in LNR order (ascending)
        1. Use the same format as 1.B.1.c to display the state data
            a. Identify the binary tree (see example output)
    f. Display the state data stored in the binary tree in RNL order (descending)
        1. Use the same format as 1.B.1.c to display the state data
            a. Identify the binary tree (see example output)
    g. Read the deletion data into the program using the third input data file
        1. Read each integer value from the States.Delete.txt data file
        2. Store each integer value in the array created in 1.B.1.a.3
    h. Delete state objects from the binary tree using the integer values stored in the array
        1. Traverse the array created in 1.B.1.a.3
        2. Locate each state object with a population value stored in the array
        3. Remove the selected state objects from the binary tree
    i. Display the updated state data stored in the binary tree
        1. Follow the instructions in 1.B.1.e and 1.B.1.f

C. The **Node** class
    1. **MUST implement the Linkable interface (Linkable.java)**
        a. **Only methods implemented for the interface can be used (a constructor is necessary)**
    2. Defines the properties and methods of a linked list or binary tree node
    3. Select the best data type for each property
    4. Use a constructor to accept a State object to store in the node

D. The **Stack** class
    1. **MUST implement the Stackable interface (Stackable.java)**
        a. **Only methods implemented for the interface can be used (a constructor is not needed)**
    2. Defines the properties and methods of a stack
    3. The stack **MUST** store State objects in singly-linked lists (**no arrays allowed**)
    4. Select the best data type for each property

E. The **BinaryTree** class

    Note: See the **BinaryTree Method Descriptions** and **UML Class Diagram** sections below for assistance.

    1. **MUST implement the Treeable interface (Treeable.java)**
        a. **Only public methods implemented for the interface can be used**
            1. **All other methods MUST be private**
            2. **A constructor is not needed**
    2. Defines the properties and methods of a binary tree
    3. Select the best data type for each property

F. The **State** class
    1. Defines the properties and methods of a state
    2. Select the best data type for each property
    3. Select the appropriate parameter types and return type for each method

Additional Notes

- Refer to chapters 4, 6 and 8 of the textbook, as well as the class lecture notes, for assistance with the algorithms required for this project
- Remove as much duplicate code as possible
- Ensure the source code conforms to the coding standards for the course
- Format the output using the **String.format** method

BinaryTree Method Descriptions

Note: Public methods are listed in **Black** and private methods are listed in **Red**.

1. **DeleteNoChildren**
   A. Deletes a node from the binary` tree where both children are null
   B. Set the appropriate child of the parent to null

2. **DeleteSingleChild**
   A. Deletes a node from the binary tree where only one child is null
   B. Set the appropriate child of the parent to the existing child of the node to delete

3. **DeleteWithChildren**
   A. Deletes a node from the binary tree where both children exist
   B. Find the lowest child node below the node to delete
      1. If on left child path
         a. Start with the right child of the node to delete
         b. Then follow the left child path to the lowest child node
      2. If on right child path
         a. Start with the left child of the node to delete
         b. Then follow the right child path to the lowest child node
   C. Set the lowest child node's child to the appropriate child on the node to delete
      1. If on left child path
         a. Set the lowest child node's left child to the left child of the node to delete
      2. If on right child path
         a. Set the lowest child node's right child to the right child of the node to delete
   D. Set the appropriate child of the parent to the appropriate child of the node to delete
      1. If on left child path
         a. Set the parent's left child to the right child of the node to delete
      2. If on right child path
         a. Set the parent's right child to the left child of the node to delete

4. **Display**
   A. Displays the state objects in the binary tree in LNR or RNL order depending on the parameter
      1. If true, call the DisplayTreeLNR method
         a. Pass the root node
      2. If false, call the DisplayTreeRNL method
         a. Pass the root node

5. **DisplayTreeLNR**
   Note: This method **MUST** be implemented as a **RECURSIVE** method.
   A. For each node
      1. Display the state record contained in each node along current node's left child path
         a. Requires a recursive method call until the left child is null
      2. Display the state record contained in the current node
      3. Display the state record contained in each node along current node's right child path
         a. Requires a recursive method call until the right child is null

6. **DisplayTreeRNL**
   Note: This method **MUST** be implemented as a **RECURSIVE** method.
   A. For each node
      1. Display the state record contained in each node along current node's right child path
         a. Requires a recursive method call until the right child is null
      2. Display the state record contained in the current node
      3. Display the state record contained in each node along current node's left child path
         a. Requires a recursive method call until the left child is null

7. **Insert(Node item)**
   A. Inserts the provided state object into the binary tree
      1. If the binary tree is empty
         a. Set the state object to the root node
      2. Otherwise, call the recursive **insert** method to insert the state object

8. **Insert(Node current, Node node)**
   Note 1: This method **MUST** be implemented as a **RECURSIVE** method.
   Note 2: This method must only be called from the insert(Node item) method.
   A. Inserts the provided state object into the binary tree
      1. Traverse the binary tree searching for a parent node to attach the state object to
         a. Recursively call this Insert method to traverse the tree
            1. Pass the current node and the node to the insert method as parameters
      2. Insert the state object at the appropriate parent node
         a. Update the parent node's leftChild or rightChild pointer

9. **IsEmpty**
   A. Returns whether the binary tree is empty

10. **Remove**
    A. Locates the appropriate binary tree node to delete using the provide state population
    B. Determine how many children the node to delete has
       1. Keep track of the parent node
       2. Keep track of the current path (left child or right child)
       3. Call the appropriate method to delete the node
          a. DeleteNoChildren
          b. DeleteSingleChild
          c. DeleteWithChildren

UML Class Diagram

A UML class diagram of the BinaryTree class is provided below. The BinaryTree class in your project should match the diagram exactly.

BinaryTree
- root : Node
- deleteNoChildren(parent : Node, leftChild : boolean) : void - deleteSingleChild(parent : Node, leftChild : boolean, temp : Node) : void - deleteWithChildren(parent : Node, leftChild : boolean, temp : Node) : void + display(ascending : boolean) : void - displayTreeLNR(node : Node) : void - displayTreeRNL(node : Node) : void + insert(item : Node) : void - insert(current : Node, node : Node) : void + isEmpty() : boolean + remove(population : int) : Node

Optional Extra Credit (10 points)

Note 1: Ensure the program satisfies each of the required design specifications above prior to attempting the extra credit.
Note 2: Several additional public methods will need to be added to the BinaryTree class to successfully complete the optional extra credit. These additional methods should be **placed at the bottom** of the BinaryTree class under the section comment "Extra Credit Methods".

Add appropriate methods to answer the following questions about the data stored in the binary tree:

1. What is the minimum state population?
2. What is the maximum state population?
3. What is the average state population?
4. What is the median state population? (This is the most difficult question to answer)
5. How many nodes exist to the left of the root node?
6. How many nodes exist to the right of the root node?
7. How many nodes have 0 children?
8. How many nodes have 1 child?
9. How many nodes have 2 children?
10. What is the maximum level reached (0 to n)?

Additional Note about Extra Credit

- Create a new class (**ExtraCredit**) to call each method and store the answers.
- Most of the code to determine the answers should exist in the **BinaryTree** class.
- It is recommended to use recursive methods to answer questions 3 through 10.
- See proper placement and examples of the answers in the **Example Output** below.

Example Output

```
Ima Java Programmer
Project 4

State              Capital            Abbr  Population  Region            Region #
Wyoming            Cheyenne           WY     6,201,427  West              6
// 48 Additional States
Alabama            Montgomery         AL     9,127,604  South             3

State Data (Ascending by Population):
State              Capital            Abbr  Population  Region            Region #
Texas              Austin             TX     1,598,167  Southwest         5
// 48 Additional States
Ohio               Columbus           OH    10,867,339  Midwest           4

State Data (Descending by Population):
State              Capital            Abbr  Population  Region            Region #
Ohio               Columbus           OH    10,867,339  Midwest           4
// 48 Additional States
Texas              Austin             TX     1,598,167  Southwest         5

Data About the Binary Tree:
Minimum State Population: 1,598,167
Maximum State Population: 10,867,339
Average State Population: 5,982,303.72
Median State Population: 6,194,749.00
Total Nodes to the Left of Root: 25
Total Nodes to the Right of Root: 24
Total Number of Nodes w/ 0 Children: 17
Total Number of Nodes w/ 1 Child: 17
Total Number of Nodes w/ 2 Children: 16
Maximum Level Reached in the Binary Tree: 12
```

Optional Extra Credit

```
Deleted States:
State              Capital            Abbr  Population  Region            Region #
Texas              Austin             TX     1,598,167  Southwest         5
// 15 Additional States
Oregon             Salem              OR    10,161,966  West              6

State Data (Ascending by Population):
State              Capital            Abbr  Population  Region            Region #
Michigan           Lansing            MI     1,863,922  Midwest           4
// 31 Additional States
Ohio               Columbus           OH    10,867,339  Midwest           4

State Data (Descending by Population):
State              Capital            Abbr  Population  Region            Region #
Ohio               Columbus           OH    10,867,339  Midwest           4
// 31 Additional States
Michigan           Lansing            MI     1,863,922  Midwest           4
```