COP3538
Project 5 – Hash Tables
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.txt (A set of state data to load into the program)
- States.Query.txt (A list of state names to load into the program)

Output File Requirements

- States.Output.txt

Miscellaneous Files

- Hashable.java (Provides the method definitions required to implement a hash table)
- Linkable.java (Provides the method definitions required to implement a node)
- States.Description.txt (Provides a description of the layout of the required input files)

Design Specification Requirements

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

1. Create 5 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 does performs the following
         a. Initialize the program
            1. Create an instance of the HashTable class
               a. The hash table **MUST** store Node objects
            2. Create an String array to hold 10 values
         b. Read data into the program using the first input data
            1. Read state records from the input data file
               a. State records are stored in both comma-separated and fixed-length formats
               b. Determine which format is used
            2. Create an object instance of the State class for each state record
               a. Use the appropriate State class constructor to create the State object

3. Insert the State object into the hash table
    a. Calculate the hash value of the state name using a hash function
    b. Store the State object in a Node object
    c. Insert the Node object into the hash table using the hash value as an index
    d. If a collision occurs (a Node object is already stored at that index in the array)
        1. Link the Node object to the existing Node object(s) using the Next pointers
            a. Node objects should be ordered by the **state name**
4. Continue until all the state records are read from the input data file

c. Display the state data stored in the hash table
1. Use the same format as project 1 to display the state data
2. Indicate the array index each state is stored in
    a. 0 or more states can exist at the same array index
    b. If no states exist at an array index, display "None"

d. Read data into the program using the second data file
1. Read each state name into the String array created in step 1.B.1.a.2

e. Search the hash table using the state names in the String array
1. Loop through the String array
2. For each state name, call the HashTable.findState method
    a. Pass the state name as a parameter
3. If the state is found, display the state name, its hash value and position in the linked list
    a. Refer to the sample output in the **Example Output** section below.
4. If the state is not found, display the state name and "was not found in the hash table."
    a. Refer to the sample output in the **Example Output** section below.

f. Write the state data in the hash table to a text data file
1. Use the filename listed in the **Output File Requirements** section
2. Use the same format as project 1 to write the state data to the file

C. The **HashTable** class
1. **MUST implement the Hashable interface (Hashable.java)**
    a. **Only methods implemented for the interface can be used (a constructor is necessary)**
2. Defines the properties and methods of a hash table
3. Select the best data type for each property
4. Use a constructor to accept an integer value to specify the size of the array in the hash table

D. 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 node
3. Select the best data type for each property
4. Use a constructor to accept a State object to store in the node

E. 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
4. Use 2 constructors to accept the State data values to store in the state
    a. A single string value (used for fixed-length records)
    b. An array of string values (used for comma-separated values records)

Additional Notes

- Refer to chapter 11 of the textbook, as well as the class lecture notes, for assistance with the 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

Example Output

```
Ima Java Programmer
Project 5

Hash Table List:
State            Capital          Abbr  Population  Region           Region #
Index 0:
Arkansas         Little_Rock      AR     2,538,303  South            3
// 2 Additional States
Michigan         Lansing          MI     9,817,242  Midwest          4

Index 1:
Alabama          Montgomery       AL     4,351,999  South            3
// 2 Additional States
Nevada           Carson_City      NV     1,746,898  West             6

Index 2:
Georgia          Atlanta          GA     7,642,207  South            3
// 6 Additional States
Utah             Salt_Lake_City   UT     2,099,758  West             6

Index 3:
Louisiana        Baton_Rouge      LA     6,170,884  South            3

Index 4:
Arizona          Phoenix          AZ     5,884,015  Southwest        5
// 3 Additional States
North_Dakota     Bismark          ND       638,244  Midwest          4

Index 5:
Delaware         Dover            DE       743,603  Middle_Atlantic  2
// 5 Additional States
Virginia         Richmond         VA     6,791,345  Middle_Atlantic  2

Index 6:
California       Sacramento       CA     4,630,270  West             6
// 3 Additional States
Wyoming          Cheyenne         WY       480,907  West             6

Index 7:
Nebraska         Lincoln          NE     9,987,615  Midwest          4
// 3 Additional States
Wisconsin        Madison          WI     5,223,500  Midwest          4

Index 8:
Montana          Helena           MT       880,453  West             6
// 2 Additional States
Washington       Olympia          WA     6,413,197  West             6

Index 9:
Alaska           Juno             AK       614,010  West             6
// 5 Additional States
West_Virginia    Charleston       WV     4,278,916  Middle_Atlantic  2

State Locations:
Kentucky location is Hash: ? Position: ?
Washington location is Hash: ? Position: ?
Nevada location is Hash: ? Position: ?
North_Dakota location is Hash: ? Position: ?
Delaware location is Hash: ? Position: ?
Louisiana location is Hash: ? Position: ?
HI was not found in the hash table
Texas location is Hash: ? Position: ?
Montgomery was not found in the hash table
```

Michigan location is Hash: ? Position: ?