CS 2336 – PROJECT 2 – Amazon Prime Delivery Tracker (part 2)

Pseudocode Due: before 10/1 at 11:59 PM

Core Implementation Due: before 10/8 at 11:59 PM

Final Submission Due: before 10/20 at 11:59 PM

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission and Grading:

- The pseudocode will be submitted in eLearning as a Word or PDF document and is not accepted late.
- All project source code will be submitted in Zybooks.
    - Projects submitted after the due date are subject to the late penalties described in the syllabus.
- **Type your name and netID in the comments at the top of all files submitted. (-5 points)**

Objectives:

- Implement a linked list using classes in Java
- Implement input validation
- Perform sorting and search algorithms on a linked list

Problem: Your Amazon internship is going well. The project manager is happy with your work on determining the area of the drivers' routes. He would like you to update the original program so it can be expanded for more than 20 drivers and additional delivery points as well as the ability to search and sort the data.

Pseudocode Information:

- Main.java
    - Detail the step-by-step logic of the main function
    - List other functions you plan to create
        - Determine the parameters
        - Determine the return type
        - Detail the step-by-step logic that the function will perform
- Linked List class
    - Sort function
        - Decide what sorting algorithm you are going to implement
        - Provide the basic algorithm for the sort
        - Update it to include any specific logic you need based on the problem

Zybooks Information:

- You will have multiple source files
    - `Main.java`
    - `LinkedList.java`
    - `Node.java`
    - `Driver.java`

- Core implementation has unlimited submissions
    - This will help you make sure the basic actions of your program are working properly in the environment
- Final submission is limited to 10 submissions
    - This will be manually checked by the grader
- White space will not be checked

Core Implementation:

- Read names and points from file
- Store names and area in linked list
- Search linked list for names and area
- Points are all positive integers
    - All integers are single digits in core implementation tests
    - Integers can be positive or negative
    - No floating point numbers in core implementation tests
- Core implementation tests will utilize sample input file for testing and grading
- Generics are not required for core implementation
- Sorting is not required for core implementation

Class Details:

- You are open to design the classes as you see fit with the minimum requirements listed below
- All classes must be of your own design and implementation.
    - **Do not use the pre-defined Java classes (-20 points if pre-defined classes used)**
- Driver class – `Driver.java`
    - Must be comparable
    - Attributes
        - Name (String)
        - Area (double)
        - Comparison variable
            - Static variable
            - Will be used to determine how to sort the list (name or area)
    - Methods
        - Default constructor
        - Overloaded constructor
            - Pass in driver's name
        - Accessors
        - Mutators
        - compareTo
        - toString
- Node class – `Node.java`
    - Must be comparable
    - Nodes are doubly linked
    - Attributes
        - Generic payload variable to hold object

        - Will be used to hold driver objects in the program
    - Next pointer
    - Previous pointer
  - Methods
    - Default constructor
    - Overloaded constructor
      - Pass in generic object for payload
    - Accessors for pointers
    - Mutators for pointers
    - toString
  - **EXTRA CREDIT (+10 points if implemented properly)**
    - Make the Node class generic
    - No reference to anything inside Driver class
    - Use generic variable to hold object
      - Will be used to hold driver objects in the program
    - Overloaded constructor will require a generic parameter
- Linked List – `LinkedList.java`
  - No reference to anything inside Driver class
  - Attributes
    - Head pointer (points to beginning of list)
    - Tail pointer (points to end of list)
  - Default constructor
  - Overloaded constructor
    - Takes node and assigns head and tail to point at the node passed in
  - Accessors
  - Mutators
  - toString (overridden)
    - create a string that lists each driver and their respective area separated on individual lines
      - `<name><tab><area><newline>`
  - Sort
    - Sort the linked list in ascending order
    - You may implement any sorting algorithm you prefer
    - The sort implementation must rearrange the nodes (-10 points if payloads swapped)

Details:

- The area of the shape can be calculated with the following formula:

$$\frac{1}{2}\left|\sum_{i=0}^{n-2}(x_{i+1} + x_i)(y_{i+1} - y_i)\right|$$

- Store each driver's information (name and area) in a node in the linked list
- Add each new node to the end of the linked list
- The number of drivers in each file is unknown
- The number of coordinates for each driver is unknown

- The list can be searched by name or area

User Interface:

- There will be 2 input files
    - Prompt the user for the name of the file containing the driver routes first
    - Prompt the user for the name of the file containing the search and sort commands
- Display all output to the console

File Structure:

- Driver route file
    - **Read this file first**
    - Each line in the file will contain the driver's first name followed by a list of coordinates
    - There will be a new line at the end of each line
        - The last line which may or may not have a newline at the end
    - The format for each line will be:
        - `<name><space><x0>,<y0><space><x1>,<y1><space>…<xn-1>,<yn-1><newline>`
        - The driver's name may be multiple words.
    - Each line in the file will represent a different driver.
    - The first and last set of coordinates will be the same.
        - If not, this should be treated as invalid input (see below)
- A second input file will provide search and sort information
    - **Read this file second**
    - Each line of the file will contain a command
    - Valid commands:
        - `sort <area/driver> <asc/des>`
            - sort the linked list on the given criteria – area or driver
            - `asc` – ascending order
            - `des` – descending order
        - `<driver name>`
            - Search the linked list for the given driver
        - `<number>`
            - Search the linked list for the given area
            - A match should be made if the number matches up to two decimal places

Input Validation

- Each file may contain invalid input
- If a line in the file contains invalid input, ignore that line
- Invalid input can contain the following:
    - In driver route file
        - Invalid characters (in name or coordinates)
            - Valid characters for names

        - Alphanumeric characters
        - Hyphens
        - Apostrophes
      - Coordinates can only be numeric (integer or floating point)
    - Coordinate missing x or y value
    - Last coordinate not same as first
  - In commands file
    - Command does not follow given format
      - See File Structure section
    - Invalid names
      - Same criteria as driver routes file
    - Invalid numbers

Output:

- All output will be written to the console
- The area will be rounded to 2 decimal places.
- After reading the entire driver route file, display all drivers and their respective areas
  - Format: `<driver name><tab><area><newline>`
- For each command in the second file, display the results:
  - Search
    - Output format if driver name or value found: `<driver name><space><area><newline>`
    - Output format if not found: `<value> not found<newline><newline>`
  - Sort – display sorted list
    - Format: `<driver name><tab><area><newline>`
  - Display a blank line between outputs