HOMEWORK 2 - Fall 2015

HOMEWORK 2 - due Tuesday, September 22nd no later than 6:00PM

REMINDERS:

- **Be sure your code follows the coding style for CSE214.**
- **Make sure you read the warnings about academic dishonesty. *Remember, all work you submit for homework or exams MUST be your own work.***
- **Login to your grading account and click "Submit Assignment" to upload and submit your assignment.**
- **You may not use any Java API Data Structures to implement this assignment.**
- **You may use Scanner, InputStreamReader, or any other class that you wish for keyboard input.**

In this assignment, you will write a train car manager for a commerical train. The train, modelled using a Double-Linked List data structure, consists of a chain of train cars, each of which contains a product load. A product load has a weight, a value, and can be dangerous or safe. The train car manager will be able to add and remove cars from the train, set the product load for each car, and determine useful properties of the train, such as it's length, weight, total value, and whether it contains any dangerous product loads.

1. Write a fully-documented class named `ProductLoad` which contains the product name (String), it's weight in tons (double), it's value in dollars (double), and whether the product is dangerous or not (boolean). You should provide accessor and mutator methods for each variable. The mutator methods for weight and value should throw exceptions for illegal arguments (i.e. negative values). The class should include a constructor. The full list of required methods is:

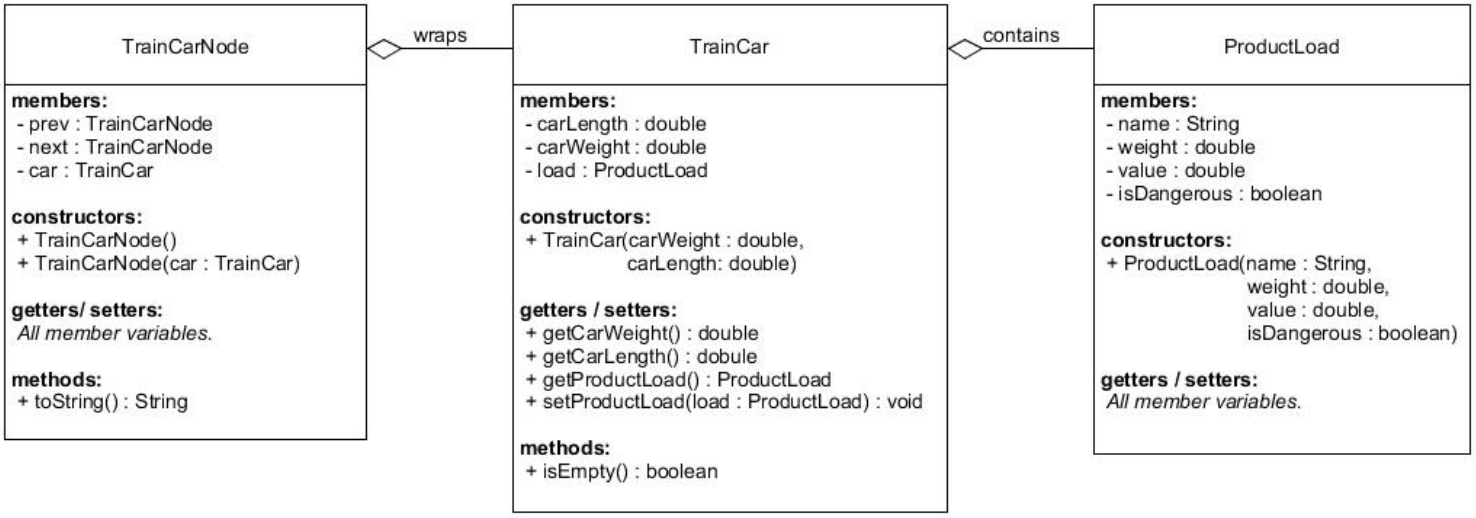
- **`public ProductLoad` - constructor (you may include a constructor with parameters)**
- **getter and setter methods for each variable**

2. Write a fully-documented class named `TrainCar` which contains a length in meters (double), a weight in tons (double), and a load reference (ProductLoad). The load variable of the train may be `null`, which indicates that the train car is empty and contains no product. The train car should have accessor methods for the length, weight, and load variables; however, you should only provide a mutator method for the load variable (the car weight and length should be constant once set). In addition, the class should specify a constructor method (with whatever parameters are necessary), and a method determining whether the car is empty or not. The full list of required methods is:

- **`public TrainCar` - constructor (you may include a constructor with parameters)**
- **getter methods for each variable**
- **setter method only for the load variable.**
- **isEmpty() method**

3. Write a fully-documented class named `TrainCarNode` which acts as a node wrapper around a `TrainCar` object. The class should contain two `TrainCarNode` references (one for the next node in the chain, one for the previous node in the chain), and one `TrainCar` reference variable containing the data. Include mutator/accessor methods for each member variable, and a constructor method. The full list of required methods is:

- **`public TrainCarNode` - constructor (you may include a constructor with parameters)**
- **getter and setter methods for each variable**



ProductLoad, TrainCar, and TrainCarNode UML specification.

4. Write a fully-documented class named `TrainlinkedList` which implements a Double-Linked List ADT. The class should contain references to the head and tail of the list, as well as a cursor variable (all `TrainCarNode`), and should provide methods to perform insertion, deletion, search, and various other operations. The class will be based on the following ADT specification:

`public class TrainLinkedList`
The `TrainlinkedList` class implements an abstract data type for a Double-Linked List of train cars supporting some common operations on such lists, as well as a few others.

- **`public TrainLinkedList()`**
  - ***Brief:***
    - **Constructs an instance of the `TrainLinkedList` with no `TrainCar` objects in it.**
  - ***Postconditions:***
    - **This `TrainLinkedList` has been initialized to an empty linked list.**
    - **head, tail, and cursor are all set to null.**

- **`public TrainCar getCursorData()`**
  - ***Brief:***
    - **Returns a reference to the `TrainCar` at the node currently referenced by the cursor.**
  - ***Preconditions:***
    - **The list is not empty (cursor is not null).**
  - ***Returns:***
    - **The reference to the `TrainCar` at the node currently referenced by the cursor.**

- **`public void setCursorData(TrainCar car)`**
  - ***Brief:***
    - **Places `car` in the node currently referenced by the cursor.**
  - ***Preconditions:***
    - **The list is not empty (cursor is not null).**
  - ***Postconditions:***
    - **The cursor node now contains a reference to `car` as its data.**

- **`public void cursorForward()`**
  - ***Brief:***
    - **Moves the cursor to point at the next `TrainCarNode`.**
  - ***Preconditions:***
    - **The list is not empty (cursor is not null) and cursor does not currently reference the tail of the list.**
  - ***Postconditions:***
    - **The cursor has been advanced to the next `TrainCarNode`, or has remained at the tail of the list.**

- **`public void cursorBackward()`**
  - ***Brief:***
    - **Moves the cursor to point at the previous `TrainCarNode`.**
  - ***Preconditions:***
    - **The list is not empty (cursor is not null) and the cursor does not currently reference the head of the list.**
  - ***Postconditions:***

        - The cursor has been moved back to the previous `TrainCarNode`, or has remained at the head of the list.

- `public void insertAfterCursor(TrainCar newCar)`
    - *Brief:*
        - Inserts a car into the train after the cursor position.
    - *Parameters:*
        - `newCar` - the new `TrainCar` to be inserted into the train.
    - *Preconditions:*
        - This `TrainCar` object has been instantiated
    - *Postconditions:*
        - The new `TrainCar` has been inserted into the train after the position of the cursor.
        - All `TrainCar` objects previously on the train are still on the train, and thier order has been preserved.
        - The cursor now points to the inserted car.
    - *Throws:*
        - `IllegalArgumentException` - Indicates that `newCar` is `null`.

- `public TrainCar removeCursor()`
    - *Brief:*
        - Removes the `TrainCarNode` referenced by the cursor and returns the `TrainCar` contained within the node.
    - *Preconditions:*
        - The cursor is not null.
    - *Postconditions:*
        - The `TrainCarNode` referenced by the cursor has been removed from the train.
        - The cursor now references the next node, or the previous node if no next node exists.

- `public int size()`
    - *Brief:*
        - Determines the number of `TrainCar` objects currently on the train.
    - *Returns:*
        - The number of `TrainCar` objects on this train.
    - *Notes:*
        - This function should complete in O(1) time.

- `public double getLength()`
    - *Brief:*
        - Returns the total length of the train in meters.
    - *Returns:*
        - The sum of the lengths of each `TrainCar` in the train.
    - *Notes:*
        - This function should complete in O(1) time.

- `public double getValue()`
    - *Brief:*
        - Returns the total value of product carried by the train.
    - *Returns:*
        - The sum of the values of each `TrainCar` in the train.
    - *Notes:*
        - This function should complete in O(1) time.

- `public double getWeight()`
    - *Brief:*
        - Returns the total weight in tons of the train. Note that the weight of the train is the sum of the weights of each empty `TrainCar`, plus the weight of the `ProductLoad` carried by that car.
    - *Returns:*
        - The sum of the weight of each `TrainCar` plus the sum of the `ProductLoad` carried by that car.
    - *Notes:*
        - This function should complete in O(1) time.

- `public boolean isDangerous()`
    - *Brief:*
        - Whether or not there is a dangerous product on one of the `TrainCar` objects on the train.
    - *Returns:*
        - Returns `true` if the train contains at least one `TrainCar` carrying a dangerous `ProductLoad`, `false` otherwise.
    - *Notes:*
        - This function should complete in O(1) time.

- `public void findProduct(String name)`
    - *Brief:*

        - **Searches the list for all `ProductLoad` objects with the indicated name and sums together their weight and value (Also keeps track of whether the product is dangerous or not), then prints a single `ProductLoad` record to the console.**
    - ***Parameters:***
        - **`name` - the name of the `ProductLoad` to find on the train.**
    - ***Notes:***
        - **This method should search the entire train for the indicated `ProductLoad`, and should not stop after the first match. For example, if there are three different `TrainCar` objects each carrying a `ProductLoad` with the name "corn", then this method should print a single record with the sum of the weight and value for the corn on each car. If nothing was found, indicate that there are no `ProductLoad` objects of the indicated name on board the train.**
        - **For the purposes of this assignment, you may assume that the dangerousness of loads with equal names are equal. Simply use the boolean value of isDangerous for the first match found.**

- **`public void printManifest()`**
    - ***Brief:***
        - **Prints a neatly formatted table of the car number, car length, car weight, load name, load weight, load value, and load dangerousness for all of the car on the train.**
    - ***Notes:***
        - **There should be a record for each `TrainCar` printed to the console, numbered from 1 to n. For each car, print the data of the car, followed by the `ProductLoad` data if the car is not empty. If the car is empty, print "Empty" for name, 0 for weight and value, and "NO" for dangerousness (see sample I/O for example).**

- **`public void removeDangerousCars()`**
    - ***Brief:***
        - **Removes all dangerous cars from the train, maintaining the order of the cars in the train.**
    - ***Postconditions:***
        - **All dangerous cars have been removed from this train.**
        - **The order of all non-dangerous cars must be maintained upon the completion of this method.**
    - ***Notes:***
        - **All the dangerous cars may be discarded after calling this method.**

- **`public String toString()`**
    - ***Brief:***
        - **Returns a neatly formatted String representation of the train.**
    - ***Returns:***
        - **A neatly formatted string containing information about the train, including it's size (number of cars), length in meters, weight in tons, value in dollars, and whether it is dangerous or not.**

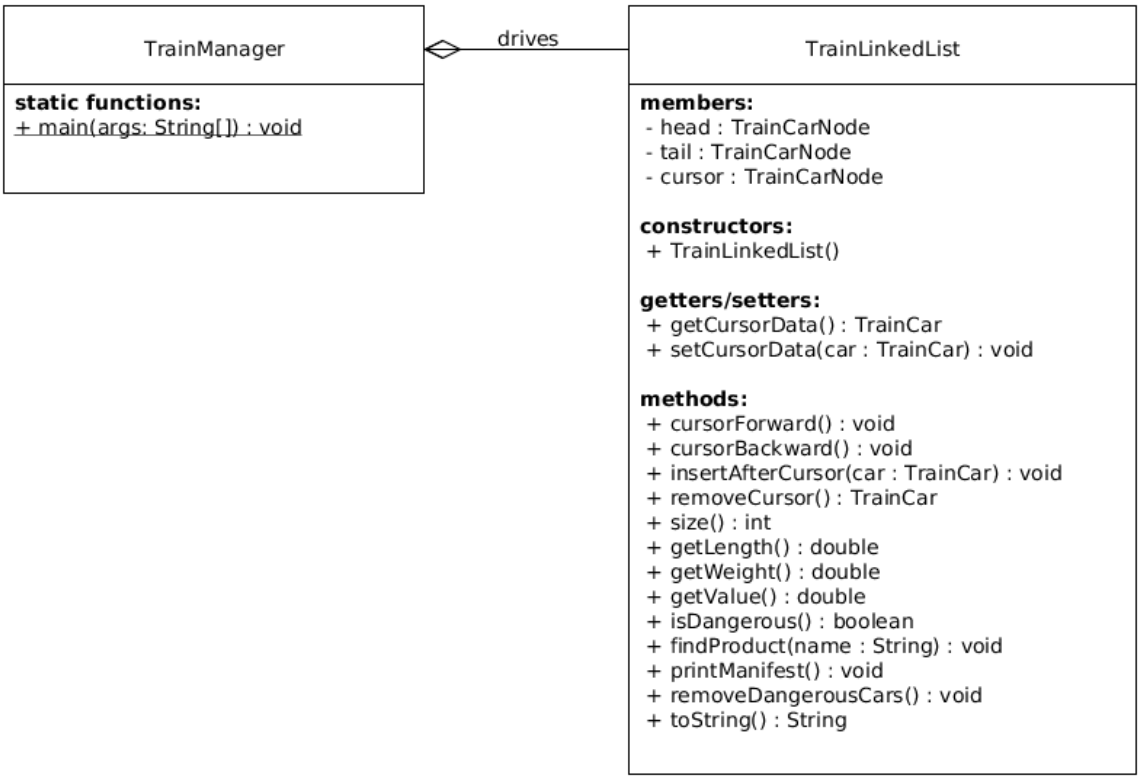
5. Write a fully-documented class named `TrainManager` that is based on the following specification:

`public class TrainManager`

`public static void main(String[] args)`
The main method runs a menu driven application which first creates an empty `TrainLinkedList` object. The program prompts the user for a command to execute an operation. Once a command has been chosen, the program may ask the user for additional information if necessary, and performs the operation. The operations should be defined as follows:

- F - Move Cursor Forward
  Moves the cursor forward one car (if a next car exists).
- B - Move Cursor Backward
  Moves the cursor backward one car (if a previous car exists).
- I - Insert Car After Cursor <Length> <Weight>
  Inserts a new empty car after the cursor. If the cursor is `null` (i.e. the train is empty), the car is set as the head of the train. After insertion, the cursor is set to the newly inserted car.
- R - Remove Car At Cursor
  Removes the car at current position of the cursor. After deletion, the cursor is set to the next car in the list if one exists, otherwise the previous car. If there is no previous car, the list is empty and the cursor is set to `null`.
- L - Set Load At Cursor <Name> <Weight> <Value> <Is Dangerous>
  Sets the product load at the current position in the list.
- S - Search For Product <name>
  Searches the train for all the loads with the indicated name and prints out the total weight and value, and whether the load is dangerous or not. If the product could not be found, indicate to the user that the train does not contain the indicated product.
- T - Print Train
  Prints the String value of the train to the console.
- M - Print Manifest
  Prints the train manifest - the loads carried by each car.
- D - Remove Dangerous Cars
  Removes all the dangerous cars from the train.
- Q - Quit

Terminates the program.



TrainLinkedList and TrainManager specification.

Note: You may include additional member variales or methods in any class as necessary or as you find convenient.

INPUT FORMAT:

- **Each menu operation is entered on its own line and should be case insensitive (i.e. 'q' and 'Q' are the same).**
- **Check to make sure that the position, if required, is valid. If not, print and error message and return to the main menu.**
- **For the Insert Course and Set Load commands, if the input information is valid, construct the object accordingly. Otherwise, print an error message and return to the main menu.**
- **You may assume Strings are at most 25 characters long.**

OUTPUT FORMAT:

- **Each command should output the result (as shown in sample IO below) after each operation is performed.**
- **All menu operations must be accompanied by a message indicating what operation was performed and whether or not it was successful.**
- **All lists must be printed in a nice and tabular form as shown in the sample output. You may use C style formatting as shown in the following example. The example below shows two different ways of displaying the name and address at pre-specified positions 21, 26, 19, and 6 spaces wide. If the '-' flag is given, then it will be left-justified (padding will be on the right), else the region is right-justified. The 's' identifier is for strings, the 'd' identifier is for integers. Giving the additional '0' flag pads an integer with additional zeroes in front.**

```
String name = "Doe Jane";
String address = "32 Bayview Dr.";
String city = "Fishers Island, NY";
int zip = 6390;

System.out.println(String.format("%-21s%-26s%19s%06d", name, address, city, zip));
System.out.printf("%-21s%-26s%19s%06d", name, address, city, zip);

Doe Jane              32 Bayview Dr.             Fishers Island, NY 06390
Doe Jane              32 Bayview Dr.             Fishers Island, NY 06390
```

HINTS:

- **Remember that the position parameter to all of the methods in the `TrainLinkedList` class refers to the position of a `TrainCar` within the list**

(starting at position 1).

SAMPLE INPUT/OUTPUT:

```
// Comment in green, input in red, output in black

(F) Cursor Forward
(B) Cursor Backward
(I) Insert Car After Cursor
(R) Remove Car At Cursor
(L) Set Product Load
(S) Search For Product
(T) Display Train
(M) Display Manifest
(D) Remove Dangerous Cars
(Q) Quit

Enter a selection: I

Enter car length in meters: 15.0
Enter car weight in tons: 10.0

New train car 15.0 meters 10.0 tons inserted into train.

// Menu not shown in sample i/o

Enter a selection: M

    CAR:                            LOAD:
      Num   Length (m)    Weight (t)  |    Name      Weight (t)     Value ($)   Dangerous
    ================================+=============================================================
 ->    1          15.0          10.0  |     Empty           0.0          0.00          NO

// Menu not shown in sample i/o

Enter a selection: L

Enter produce name: Corn
Enter product weight in tons: 100.0
Enter prduct value in dollars: 15440
Enter is product dangerous? (y/n): n

100.0 tons of Corn added to the current car.

// Menu not shown in sample i/o

Enter a selection: M

    CAR:                            LOAD:
      Num   Length (m)    Weight (t)  |    Name      Weight (t)     Value ($)   Dangerous
    ================================+=============================================================
 ->    1          15.0          10.0  |      Corn         100.0     15,440.00          NO

// Menu not shown in sample i/o

Enter a selection: I

Enter car length in meters: 18.5
Enter car weight in tons: 8.3

New train car 18.5 meters 8.3 tons inserted into train.

// Menu not shown in sample i/o

Enter a selection: M

    CAR:                            LOAD:
      Num   Length (m)    Weight (t)  |    Name      Weight (t)     Value ($)   Dangerous
    ================================+=============================================================
       1          15.0          10.0  |      Corn         100.0     15,440.00          NO
 ->    2          18.5           8.3  |     Empty           0.0          0.00          NO

// Menu not shown in sample i/o
```

```
Enter a selection: L

Enter product name: Corn
Enter product weight in tons: 85.0
Enter prduct value in dollars: 13120
Enter is product dangerous? (y/n): n

85.0 tons of Corn added to the current car.

// Menu not shown in sample i/o

Enter a selection: M

    CAR:                              LOAD:
      Num   Length (m)    Weight (t)  |    Name      Weight (t)     Value ($)   Dangerous
    ================================+=============================================================
        1          15.0          10.0  |      Corn         100.0     15,440.00          NO
 ->     2          18.5           8.3  |      Corn          85.0     13,120.00          NO

// Menu not shown in sample i/o

Enter a selection: T

Train: 2 cars, 33.5 meters, 203.3 tons, $28,560.00 value, not dangerous.

// Menu not shown in sample i/o

Enter a selection: F

No next car, cannot move cursor forward.

// Menu not shown in sample i/o

Enter a selection: B

Cursor moved backward

// Menu not shown in sample i/o

Enter a selection: M

    CAR:                              LOAD:
      Num   Length (m)    Weight (t)  |    Name      Weight (t)     Value ($)   Dangerous
    ================================+=============================================================
 ->     1          15.0          10.0  |      Corn         100.0     15,440.00          NO
        2          18.5           8.3  |      Corn          85.0     13,120.00          NO

// Menu not shown in sample i/o

Enter a selection: I

Enter car length in meters: 32.1
Enter car weight in tons: 17.4

New train car 32.1 meters 17.4 tons inserted into train.

// Menu not shown in sample i/o

Enter a selection: M

    CAR:                              LOAD:
      Num   Length (m)    Weight (t)  |    Name      Weight (t)     Value ($)   Dangerous
    ================================+=============================================================
        1          15.0          10.0  |      Corn         100.0     15,440.00          NO
 ->     2          32.1          17.4  |     Empty           0.0          0.00          NO
        3          18.5           8.3  |      Corn          85.0     13,120.00          NO

// Menu not shown in sample i/o

Enter a selection: L

Enter produce name: TNT
Enter product weight in tons: 25.0
Enter prduct value in dollars: 151500
Enter is product dangerous? (y/n): y
```

```
25.3 tons of TNT added to the current car.

// Menu not shown in sample i/o

Enter a selection: M

    CAR:                              LOAD:
      Num   Length (m)    Weight (t)  |    Name      Weight (t)     Value ($)   Dangerous
    ================================+=============================================================
        1          15.0          10.0  |      Corn         100.0     15,440.00          NO
 ->     2          32.1          17.4  |       TNT          25.3    151,500.00         YES
        3          18.5           8.3  |      Corn          85.0     13,120.00          NO

// Menu not shown in sample i/o

Enter a selection: T

Train: 3 cars, 65.6 meters, 246.0 tons, $180,060.00 value, DANGEROUS.

// Menu not shown in sample i/o

Enter a selection: F

Enter product name: Corn

The following products were found on 2 cars:

        Name      Weight (t)     Value ($)   Dangerous
    ================================================
         Corn         185.0     28,560.00          NO

// Menu not shown in sample i/o

Enter a selection: F

Enter product name: Milk

No record of Milk on board train.

// Menu not shown in sample i/o

Enter a selection: D

Dangerous cars successfully removed from the train.

// Menu not shown in sample i/o

Enter a selection: M

    CAR:                              LOAD:
      Num   Length (m)    Weight (t)  |    Name      Weight (t)     Value ($)   Dangerous
    ================================+=============================================================
        1          15.0          10.0  |      Corn         100.0     15,440.00          NO
 ->     2          18.5           8.3  |      Corn          85.0     13,120.00          NO

// Menu not shown in sample i/o

Enter a selection: R

Car successully unlinked. The following load has been removed from the train:

        Name      Weight (t)     Value ($)   Dangerous
    ================================================
         Corn          85.0     13,120.00          NO

Enter a selection: M

    CAR:                              LOAD:
      Num   Length (m)    Weight (t)  |    Name      Weight (t)     Value ($)   Dangerous
    ================================+=============================================================
 ->     1          15.0          10.0  |      Corn         100.0     15,440.00          NO

// Menu not shown in sample i/o

Enter a selection: Q
```

```
Program terminating successfully...
```
