DUE DATE: APRIL 3RD, 2020 AT 11:59PM

Instructions:

- Read the **`SubmissionProcedures.PDF`** and the **`Standards.java`** carefully BEFORE you start.
- Let me know about any typos, clarifications or questions you may have right away. Don't wait.
- Read the whole assignment before you start.
- Start right away. The sooner you get stuck, the sooner you can get help to get un-stuck.

Assignment overview

Learning outcomes:

- Learning to extend your previous assignments
- An understanding of object hierarchies and use subclasses to extend other classes.
- An understanding of how use LinkedLists.
- Familiarity with StdDraw to create more complex animation sequences
- Familiarity using recursion

Additional Notes:

Read **Standards.java** and **SubmissionGuidelines.PDF** Carefully & before you start.

Several Classes will be provided as a starting point including some that you developed or worked with in A3. You can either continue from your own solution to A3 (I will give a couple of bonus marks for this if you can adapt your current code to work) or start from the handout.

As before, you should be tracking your project in git and using all the best practices we have been discussing so far.

Please provide detailed commenting and commit your code often!!!

Tips:

Start early. Read the assignment fully before beginning. We will go through it in the first class after it has been handed out. You can also email questions or join in on the Zoom video chat channel to ask questions. I will be checking email often and exploring new platforms for handling this distance ed situation but if you have any suggestions I am open to hearing them.

Notes on Academic Misconduct:

Your code should be done entirely by you. Any suspicion of Academic Misconduct or plagiarism will be fully investigated and can lead to serious academic penalties including 0 on the assignment for a first office, or an F in the course. Third offences could lead to expulsion from ICM. If you have any questions as to what qualifies as Academic Misconduct, please discuss it with me.

Phase 0: Introduction

Before you start.

I am providing a framework of the methods here with the understanding that you will create a **`GameController`** class that contains your **`main`** method and a **`TestClass`** for testing all of your methods as you progress. It can be useful in your **`TestClass`** to create several static methods that each test one phase, or part of a phase. This way you can easily focus your attention on the most relevant information while maintaining an ability to run tests on earlier classes easily (in case you break them along the way).

Phase 1: Node Object

The Start of Linked Lists

The **Node** class will store a **Vector2** object as well as the next **Node** in the sequence and is a part of your **LinkedList**.

It will contain the following:

Variables:

private Vector2 data stores a single **Vector2** object.
private Node next stores the next **Node** in the **LinkedList**

Methods

You should create a constructor that accepts both a **Vector2** data value and a **Node** (the next **Node**) (in that order).

It should also contain basic accessor and mutator methods

```
public void setData(Vector2 data)
public void setLink(Node next)
public Vector2 getData()
public Node getLink()
```

Next, you will create a **toString()** method which should return the **Vector2** with the following format:

```
N(0.001, 0.123)
```

Where **N** represents a **Node**, and **0.001** is the **x** values of the **Vector2, 0.123** is the **y** value of the **Vector2**. They should include the brackets and comma and be rounded to 3 decimal places.

You can use the **Vector2 toString()** method to help with this.

Phase 2: Basic Linked List operations

Fundamentals of LinkedLists

You will be creating a **`LinkedList class`**. It will hold a list of **`Nodes`** and allow for several operations on them. This particular **`LinkedList`** will hold **`Nodes`** which contain **`Vector2`** values, allowing for some specific methods related to geometry.

Your linked list only needs a single variable, **`Node top`**. This points to the first **`Node`** in the list or **`null`** if the list is empty.

Create a **`Constructor`** that accepts no parameters.

`boolean isEmpty()` will return true if the list is empty and false otherwise.

`public Node getTop()` will return the first **`Node`** in the list.

`public int size()` will return the total number of **`Nodes`** stored in the list. This should be calculated each time **`size()`** is called, rather then being stored as it was in our partially full array.

`void add(Vector2 data)` will add the given **`Vector2`** into the **`LinkedList`** as the first **`Node`**. It will need to create a new **`Node`** to hold it.

`void add(Node newNode)` will add the given **`Node`** to the front of the **`LinkedList`**

`void addLast(Vector2 data)` will add a new **`Node`** to the **`LinkedList`** with the data value given.

`Vector2 remove()` will remove the first **`Node`** in the list and return the value. If the list is empty it should return **`null`** (however you should avoid calling this method if the list is empty so check first).

`String toString()` will return a **`String`** that includes ALL the nodes in the list. It should create this **`String`** by stepping through the **`LinkedList`** an calling **`toString`** on each individual **`Node`**.

Phase 3: Recursion – Weights and Values

A Thing to hold other Things, but how much is in that Thing?

We can currently store a list of **`Things`** in a **`ContainerThing`**. Since **`ContainerThing`** is itself a **`Thing`**, this should mean that you should be able to store **`ContainerThings`** in other **`ContainerThings`**. This is the essence of recursion.

Complete the modifications necessary to enable **`ContainerThing`** to calculate the total weight of all **`Things`** inside itself, including other **`ContainerThings`**. This must be done recursively to return a final sum of ALL **`Things`**, even **`Things`** within other **`Things`**. You should edit the **`totalWeight()`** method in the **`ContainerThing`** and other classes (**`Thing`**, **`QuantityThing`**) to allow the **`Container`** to recursively search for the weight of all things it holds. It should also support

You should also update your **`Thing`** (and subclasses) **`toString`**() methods to work recursively in this same way.

Up until now, we have been just storing the **`value`** of a **`Thing`** as a primitive value in the **`Thing`** super class but we should be able to calculate the total **`value`** of all **`Things`** in a **`ContainerThing`**, including other **`ContainerThings`** which in turn may contain even more **`Things`**.

Add a method **`double totalValue()`** to all **`Thing`** objects (including subclasses) which will return the value of the specified **`Thing`**. If it is a **`ContainerThing`**, it should also include the value of any **`Thing`** objects stored inside it recursively. Again, this should work recursively even if you can find an iterative solution.

Example:

> **`ContainerThing item1`** *has a* **`weight`** *of* **`1`**
> **`item1`** *contains* **`ContainerThing item2`** *which also has a* **`weight`** *of* **`1`**
> **`item2`** *contains* **`ContainerThing item3`** *which has a* **`weight`** *of* **`3`**
> **`item2`** *contains a* **`Thing`** *object* **`item4`** *which has a* **`weight`** *of* **`5`**
>
> *This gives us* **`item1`** *(1) +* **`item2`** *(1) +* **`item3`** *(3) +* **`item4`** *(5) so* **`item1.totalWeight()`** *returns 9 while* **`item2.totalWeight`** *would return 8.*

Phase 4: More Complex LinkedLists

`void drawLine()` Will iterate through the **`LinkedList`** and draw a line that connects all points. This method assumes that the points represent positions on the GUI (rather than Vectors).

`public double totalLength()` will return the total distance between all the **`Nodes`**. This should return **`0`** if there are less than 2 **`Nodes.`** To do this, you will find the distance between **`Node0`** and **`Node1`**, then add that to the distance between **`Node1`** and **`Node2`**, then add that to the distance between **`Node`** 2 and **`Node`** 3, until all the **`Nodes`** have been included.

`void insert(int index, Vector2 data)` will insert a new **`Node`** containing in the nth position. if **`index`** is greater than the last **`Node`**, add it to the end but print a warning to the console.

`int compareTo(LinkedList other)` will compare the measured lengths of two **`LinkedLists`**. It should return **`0`** if they are equal, a negative value if this **`LinkedList`** is shorter than the other **`LinkedList`**, and a positive value if this **`LinkedList`** is longer then the other **`LinkedList`**. You can use the **`totalLength()`** method for this calculation.

> *Side note: Keep in mind we could have alternatively chosen to write the compareTo method to compare some other factors (size() for example) so this really depends on the needs of the project.*

`public LinkedList deepCopy()` returns a deep copy of this **`LinkedList`**. You should also create a deep copy of the **`Nodes`** themselves and the **`Vector2`** objects within them to avoid issues in the future with multiple references.

`public void addList(ArrayList source)` will accept an **`ArrayList`** containing **`Vector2`** values. These should be added to the **`LinkedList`** so that they are in the same order as in the **`ArrayList`** (that is, index 0 of the **`ArrayList`** will be in the first **`Node`** of the **`LinkedList`**).

For full marks, you should avoid using the **`addLast`** method as it will cause unnecessary processing. You should also avoid iterating through the **`LinkedList`** more than once. There is a simple solution if you think hard about it.

Phase 5: Advanced Movement

Create an **`AnimationSequence class`** that extends **`AnimatedThing`**

`AnimationSequence` will store a **`LinkedList`** of **`Vector2`** coordinates. It will set its own animation goal position (using **`AnimatedThing`** methods) to be the first position in the list, then move towards that goal until it is within range (some small threshold value). Next, it will remove the first position (the current goal) and set the next **`Vector2`** waypoint value as its new goal. Once all the positions have been reached it should stop moving and print out a success message. You should be able to use several of the methods provided in **`AnimatedThing`** to help with this.

`public AnimatedSequence()` is just a basic constructor.

`public void addWaypoint(Vector2 point)` will add a new waypoint to the front of the list. This should also set the current destination to the new position. That is, the **`AnimationSequence`** should immediately start moving towards the new point provided.

`public void drawPath()` will draw a path from the current position, through all the remaining points on the current path. This might be a good one to do first as it is useful in testing.

`public void setSequence(LinkedList sequence)` will accept a **`LinkedList`** as input. It should immediately reset any current animations and use the new sequence as the new path to follow.

`public void insertWaypoint(int index, Vector2 point)` will add a new waypoint to the current LinkedList of waypoints, at the specified index.

`public void addLastWaypoint()` will add a new waypoint to the end of the **`LinkedList`**. It should not affect the current animation sequence (unless the list was previously empty).

`public void reachedGoal()` will override the same method in the **`AnimationThing`**. Since this method will get called each time a waypoint is reached (via polymorphism), you should create a new version that can detect whether a waypoint is reached and a new waypoint needs to be set or the end of the path is reached. If the end of the path is reached, you should call the **`reachedGoal()`** method in the **`AnimatedThing`** class.

*Note, the current position will not be included if you just use the **`LinkedList drawLine`**() method directly without any additional steps taken.*

Note: You should be detecting invalid input and handling it when possible.

Hand in instructions:

Place all your **.java** files, the **.git** repo and the **.gitignore** files in a single .zip file with the name LASTNAME_FIRSTNAME_A2.zip (NOT RAR or other archive formats) in the hand-in folder on Moodle with your **readme.txt, honesty declaration and test_output.txt**

Files to hand in:

Thing.java and all subclasses
LinkedList.java
Node.java
Vector2 (if you made modifications, which should not be necessary)
GameController.java (with the **main** method)
TestClass.java

DO NOT:
- DO NOT hand in **.class** files
- DO NOT hand in Standards.java or other unused or unneeded files.
- DO NOT hand in **bluej** files or other various editor files **(.vs, .proj, etc).**
- DO NOT ignore the hand in instructions.

Congratulations, you are done. Be proud of yourself and relax.

Hand In Additional Notes:

- It is especially important that your classes and methods are very accurate. Make sure your class names are **Uppercase** and your **methodNames** are lowercase and don't contain typos. **If your program doesn't compile for the marker you will lose significant marks**, even if it works on your machine.

- Also be sure you remove any package headers added by IntelliJ or Eclipse etc as these will break compilation.

- See the **SubmissionGuidelines.pdf** document for full instructions. See the **Standards.java** document for formatting guidelines. Don't hand in the **Standards.java** file.

- Include a **README.txt** file when you hand in the assignment to tell the marker which phases were completed, so that only the appropriate test can be run. This can also include messages about errors you are encountering.

- For example, if you say that you completed Phases 1-3, then the marker will compile your files to check for completion of those phases. If it fails to compile and run, you will lose all of the marks for the test runs (at least half of the assignment). The marker will not try to run anything else, and will not edit your files in any way.

- Submit a completed Honesty Declaration with EACH assignment. Your assignment will not be marked without it. I will provide an easy text based version you can use so you don't have to mess around signing PDFs.

Good luck! Don't Panic!