COMP 1020
Lab 9

MATERIAL COVERED

- Linked Lists

Notes:

- The three exercises in this lab are cumulative – each builds on the previous one, and they must be done in order.
- Only one of the three exercises is required, but try to do as many as you can.

A list of (x,y) points

Create two classes **`Node`** and **`PointList`** which will implement a list of (x,y) points using a linked list.

1. A **`Node`** should have three private instance variables: two **`double`** values **`x`** and **`y`**, and a link to the next **`Node`** in the list. Provide a constructor which will initialize all three of these instance variables. Provide a **`Node getLink()`** method which will allow access to the link. You do not need any other get/set methods for the Bronze exercise.

2. Provide a **`String toString()`** method in the **`Node`** class which will return a **`String`** showing the **`x`** and **`y`** values in the format **`"(0.343,0.982)"`** . Use the built-in method **`String.format("%5.3f",dataValue)`** to ensure that both the **`x`** and **`y`** values display exactly 3 decimal places.

3. A **`PointList`** should have a single private instance variable (usually named "top") containing a reference to the first **`Node`** in the list (or **`null`** if there are none). Provide a constructor which will create an empty list.

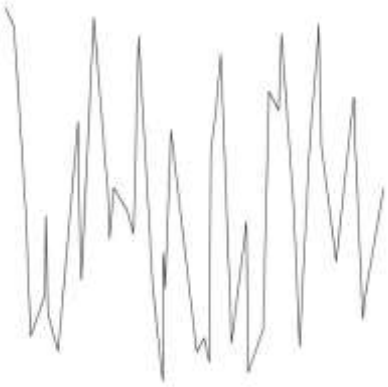
4. Provide a **`void add(double x, double y)`** method to the **`PointList`** class which will add the point (x,y) to this list of points, ***at the beginning***. Provide the usual **`String toString()`** method which will return a **`String`** showing all of the points in the list, in the following format: **`"[ (0.435,0.234)  (0.123,0.876)  (0.321,0.789) ]"`**. Note that, to keep it simple, you can simply surround each point with some blank space.

5. Run the supplied **`TestLab9Bronze.java`** program to test your classes. This program will generate and print a list of 5 random points.

A sorted list of points.

Modify your classes so that the list of points can be kept sorted, with the x co-ordinates in ascending order. (The y coordinates will not be in any particular order.) You will also add a method that will display the list of points in the **`StdDraw`** window by "connecting the dots" – joining the sequence of points by lines.

1. Modify the **`Node`** class by adding the usual get/set methods **`getX`**, **`getY`**, and **`setLink`** which you will need. (You already have a **`getLink`** method. You still don't need **`setX`** or **`setY`**.)
2. Add a **`void insert(double x, double y)`** method to the **`PointList`** class, which will add the point (**`x`**,**`y`**) to the list of points, *in the correct location so that the list will remain sorted*, with the x coordinates in ascending order (assuming that the existing points in the list are already sorted, of course).
3. Add a **`void connectTheDots()`** method to the **`PointList`** class which will draw a series of lines in the **`StdDraw`** window that connect the points in the list. A line should connect the 1st point to the 2nd, another line should connect the 2nd point to the 3rd, etc. Do not connect the last point to the first point. If there are fewer than 2 points in the list, it shouldn't draw anything at all – but it shouldn't crash, either. This will be tested. The **`StdDraw`** method to draw a line is simply **`StdDraw.line(x1,y1,x2,y2)`**.
4. Run the supplied **`TestLab9Silver.java`** program to test your new classes. If the sorting is working properly, then you should see a (random) image something like the one below. The lines should go strictly left-to-right, and should never "double back".



Multiply-linked lists

There is no rule that says that a **`Node`** has to be on just *one* list. Modify your **`Node`** and **`PointList`** classes so that the points are kept sorted by *both* the **`x`** coordinate, *and* the **`y`** coordinate, at the same time. You will not have to write any completely new code at all, but you will have to duplicate and modify a lot of existing code to provide two different versions – one for **`x`**, and one for **`y`**.

1. Modify your **`Node`** class so that every node now has 2 links: one to the next point in a list sorted by **`x`** coordinate, and one to the next point in a list sorted by **`y`** coordinate. Add a 4th parameter to your constructor to match. You will now need **`getXLink`**, **`getYLink`**, **`setXLink`**, and **`setYLink`** methods. (The **`getX`**, **`getY`**, and **`toString`** methods are not affected.)

2. Modify your **`PointList`** class so that it has *two* top/first pointers and not just one – one to the first **`Node`** in the list of nodes sorted by **`x`**, and another to the first **`Node`** in the list of nodes sorted by **`y`**. Change the constructor to match. You can delete the **`add`** method entirely, since it should no longer be used. The **`toString`** method should continue to list the nodes in ascending order by **`x`** values.

3. Modify the **`void insert(double x, double y)`** method so that it still creates a single new **`Node`**, but now inserts it into *both* lists (the one sorted by **`x`**, and the one sorted by **`y`**).

4. Provide *two* versions of the connect-the-dots method: **`connectTheXDots`** and **`connectTheYDots`**, one of which will connect the points in the order given by the **`x`** coordinates, and the other by the **`y`** coordinates.

5. Run the supplied **`TestLab9Gold.java`** program to test your new classes. If the sorting in both directions is working properly, then you should see a (random) image something like the one below. The black lines should go strictly left-to-right, and the red lines should go strictly bottom-to-top, but both should join the same set of points. Modern art!!