Spatial Trees

There are lots of real world applications that require efficient storage and queries of spatial data (points, lines, rectangles, etc). In this assignment, you'll be implementing a data structure that stores 2D points in a way that supports efficient "what's nearby?" queries. For example, if the points represent restaurants, this data structure could quickly answer the query "which restaurants are within walking distance?"

This requires drawing output.

2-d Trees

We will be storing our 2D points in a binary tree that is similar to the binary search tree we discussed in class. It is not quite the same, however, because we have a question to consider: How do we order 2D points? Should we order them by *x*-coordinate? By *y*-coordinate? By some mixture of the two?

2-d trees use the last approach: ordering the points by a mix of the two coordinates. They have two types of nodes -- *x*-nodes and *y*-nodes -- and maintain the following two properties:

- The children of an *x*-node are *y*-nodes
- The children of a *y*-node are *x*-nodes

Each type of node uses a different comparison to order points. This causes different levels of the tree to compare points differently, using the following rules:

- For every *x*-node:
    - All descendants in the left subtree have a smaller *x*-coordinate than the point stored at the node. Visually, all descendant points are left of this point.
    - All descendants in the right subtree have a larger *x*-coordinate than the points stored at the node. Visually, all descendant points are right of this point.
- For every *y*-node:
    - All descendants in the left subtree have a smaller *y*-coordinate than the point stored at the node. Visually, all descendant points are below this point.
    - All descendants in the right subtree have a larger *y*-coordinate than the point stored at the node. Visually, all descendant points are above this point.

As it turns out, this property allows us to efficiently perform queries like "which points are within a given radius of a target?"

What to Do

You will need to implement the following classes:

- *SpatialTreeNode*
- *SpatialTree*
- *Driver*

The details of each class are given below:

Point2D

We will be representing points using *java.awt.geom.Point2D* (链接到外部网站。). This is an abstract class that has a bit of an odd constructor:

```
Point2D point = new Point2D.Double(0.0, 0.0);
```

The upside is that you get good helper methods like **distance()**, which will be useful later when we make queries.

SpatialTreeNode

This class will be very similar to our *TreeNode* from class, it stores a *left*, *right*, and *parent* reference. Additionally, it should store a *Point2D* object, and a *boolean* representing whether it is an *x*-node or a *y*-node.

SpatialTree

This class will represent the actual 2-d tree, and should store the *root* as a *SpatialTreeNode*. You will need to implement the following methods:

- **SpatialTree()**
  - A constructor that creates an empty tree
- **void add(Point2D point)**
  - Adds a point to the tree
- **void draw()**
  - Draws the spatial tree (you may use StdDraw or Swing; your choice). Each node should show the point, and the regions bounding each point (see below). (*Hint:* use recursion.)
- **ArrayList<Point2D> query(Point2D center, double radius)**
  - Returns a list of all points contained inside the query circle (the distance to *center* is less than or equal to *radius*)
  - This method is the what this data structure was designed for! **Take care to implement this efficiently by exploring nodes only when necessary.** (*Hint:* consider checking an individual node. What do you have to check at that node? Is it always necessary to search both child subtrees?)

Driver

In the driver, simply create a spatial tree and fill it with random points (about 100 works well).

Then in an infinite loop, allow the user to draw query circles and display the points that fall within the query circle (see below for an example run).

Example Functionality

Here is an example video 链接到外部网站。 of how your application might behave on a random set of points.

The first phase shows points being added by the user (you don't have to implement this, but it is useful for debugging). Note how x-nodes (represented with a red line) divide regions into left and right halves, while y-nodes (represented with a blue line) divide regions into top and bottom halves.

The second phase shows the user making queries. You should implement an interface that allows the user to do this. Each time the user draws a disk, you should display the points that fall within that query region.

What to Submit

When you are finished, submit a zip of your project on Canvas. In the least, this should include the source files:

- SpatialTreeNode.java
- SpatialTree.java
- Driver.java

Hints

1. At first, this seems like a complicated task, but it's not too different from a standard binary tree. Before you start coding, try to think of what exactly is different here. Try to reuse as much of the code from class as you can. What pieces (from a high level) need to be changed?
2. It can be tough to test this code, so I suggest doing this in parts. First, code up the **add()** method. Add a **toString()** method to print out the actual structure of the tree, and work a few examples by hand to make sure your trees match the structure exactly. Second, code up the **draw()** method, and test it the same way (run a few examples by hand, ensuring they draw how you expect). Lastly, code the **query()** method and its display method.
3. Take time to test your methods as you go! It's easy to make a mistake in the **add()** method, which only becomes harder to find as you write more code!
4. In order to draw the line that correspond to point in the node, we will pass the rectangle bound in which the line is draw. See the picture below: