PR Quadtree

This assignment involves implementing a region quadtree (specifically the PR quadtree as described in section 3.2 of Samet's paper) as a Java generic. Because this assignment will be auto-graded using a test harness we will provide, your implementation must conform to the public interface below, and include at least all of the public and private members that are shown:

```
// The test harness will belong to the following package; the quadtree
// implementation must belong to it as well.  In addition, the quadtree
// implementation must specify package access for the root pointer in order
// that the test harness may have access to it.
//
package Minor.P3.DS;
import java.io.FileWriter;
import java.io.IOException;

public class prQuadtree< T extends TwoDComparable<? super T> > {

    // You must use a hierarchy of node types with an abstract base
    // class.  You may use different names for the node types if
    // you like (change displayHelper() accordingly).
    public class prQuadNode {
        // abstract class
    }
    public class prQuadLeaf extends prQuadNode {
        // members and public interface are up to you
    }
    public class prQuadInternal extends prQuadNode {
        // members and public interface are up to you
    }

    prQuadNode root;                // pointer to root node, if any
    long xMin, xMax, yMin, yMax;    // world boundaries

    public prQuadtree(long xMin, long xMax, long yMin, long yMax) {
        // details are up to you
    }

    public boolean insert(T elem) {
        // details are up to you
    }

    public boolean delete(T Elem) {
        // details are up to you
    }

    public T find(T Elem) {
        // details are up to you
        return null;              // if no matching element is found
    }

    public Vector<T> find(long xLeft, long xRight, long yLow, long yHigh) {
        // details are up to you
        return null;              // if no matching elements are found
    }

    public void display(FileWriter Out) {
        // see the comments and code for display() below
    }
}
```

You may safely add features to the given interface, but if you omit or modify members of the given interface you will almost certainly face compilation errors when you submit your implementation for testing.

You must implement tree traversals recursively, not iteratively. You will certainly need to add a number of private recursive helper functions that are not shown above. Since those will never be called directly by the test code, the interfaces are up to you.

For this assignment, the bucket size is 1; that is, each leaf node will store exactly one data object.

Note that the test harness for this assignment is shallower than the one for the BST project. That is, aside from checking the root pointer, it does not attempt to examine the internal structure of your quadtree at all. Therefore, the restrictions on the node types are extremely flexible.

On the other hand, your implementation must be designed to work with data objects that implement the following interface:

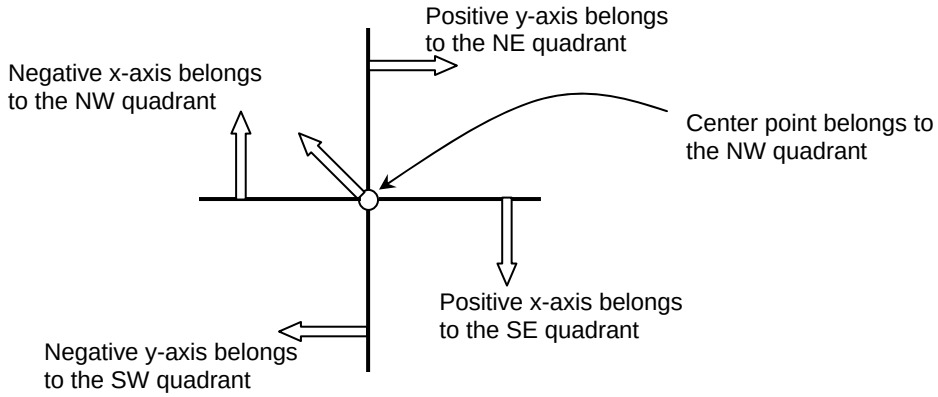
```
package Minor.P3.DS;

public interface TwoDComparable<T> {

   public Direction compare2D(Object other);
   public Direction compare2D(Long x, Long y);
   public Direction compare2D(Double x, Double y);
   public long getX();
   public long getY();

   public boolean equals(Object other);
}
```

The interface is intended to allow your implementation to either take coordinates from the data object (via `getX()` and `getY()`) and use those values to make directional decisions, or to use the `compare2D()` method supplied by the data object for the same purpose. The only restriction is that if you take the first approach you should make sure that your logic follows the following pattern when partitioning the world space:



The actual test data will be created so that (ideally) no data points will ever lie on a region boundary. However, you should conform to the guideline above just to be safe.

The `TwoDComparable` interface depends upon the following enumerated type:

```
package Minor.P3.DS;

public enum Direction {NW, SW, SE, NE, NONE};
```

During our testing of your implementation, we will use the following data object type; a more complete version is posted on the course website.

```
package Minor.P3.DS;

public class Point implements TwoDComparable<Point> {

   private long xcoord;
   private long ycoord;

   public Point() { . . . }
   public Point(long x, long y) { . . . }

   public long getX() { . . . }
   public long getY() { . . . }

   public boolean equals(Object other) { . . . }

   public Direction compare2D(Object other) { . . . }
   public Direction compare2D(Long x, Long y( { . . . }
   public Direction compare2D(Double x, Double y) { . . . }
   public String toString() { . . . }
}
```

Note that if you use calls to `compare2D()` in your quadtree you should implement that method to be consistent with the directional guidelines given on the previous page (if you want to match the posted test results).

Displaying the tree

It is necessary that your quadtree implementation display the tree in a specific form, so that the testing protocols can evaluate the structure correctly. The posted shell for the PR quadtree class includes a complete implementation of display methods that will produce satisfactory results. The given code is essentially the same as that in the course notes; the only significant change is that hyphens are used to make the indentation of each node clearer.

You may modify the given display code if you need to change names of the node classes or node members, but do not alter the manner in which the tree nodes are laid out when the tree is printed.

Design and implementation requirements

There are some explicit requirements, in addition to those on the *Programming Standards* page of the course website:

- You must implement the PR quadtree to conform to the specification.
- The insertion logic must not allow duplicate (according to `equals()`) records to be inserted.
- The insertion logic must not allow the insertion of records that lie outside the world boundaries.
- Deletion must be handled exactly as described in the course notes. In particular, pay attention to the possibility that a branch may contract more than a single level when a leaf node is deleted.
- Under no circumstances should any of the PR quadtree member functions write output, except for the `display()` method and its helper.

Testing:

We will be testing your implementation with our own test driver. We may (or may not) release information about that driver before the assignment is due. In any case, it is your responsibility to design and carry out a sensible test of your implementation before submitting it. For that purpose, you may share test code (**but absolutely no tree code!!**) via the class Forum.

Be sure you test all of the interface elements thoroughly, both in isolation and in interleaved fashion.

Evaluation:

You should document your implementation in accordance with the *Programming Standards* page on the course website. It is possible that your implementation will be evaluated for documentation and design, as well as for correctness of results. If so, your submission that achieved the highest score will be evaluated by one of the TAs, who will assess a deduction (ideally zero) against your score from the Curator.

Note that the evaluation of your project may depend substantially on the quality of your code and documentation.

What to turn in and how:

This assignment will be auto-graded using a test harness on the Curator system. The testing will be done under Windows (which should not matter at all) using Java version 1.6.21 or later.

Submit a single `.java` file (not zipped) containing your PR quadtree generic to the Curator System. Submit nothing else. Your solution should not write anything to standard output.

Your submitted source file will be placed in the appropriate subdirectory with the packaged test code, and will then be compiled with the test harness using the following command, executed in the root of the source directory tree:

```
javac testDriver.java
```

Instructions, and the appropriate link, for submitting to the Curator are given in the *Student Guide* at the Curator website:

http://www.cs.vt.edu/curator/.

You will be allowed to submit your solution multiple times; the highest score will be counted.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement provided on the Programming Standards page in one of your submitted files.