QuadTrees

A quadtree is a datastructure that is used to break up 2D space into regions. Quadtrees use adaptive subdivision to place smaller regions where there are a greater number of objects. Below is an example of a quadtree:

Quadtrees are created by starting with one large rectangle that contains the entire world. If the number of objects in that rectangle is too large it then subdivides itself into 4 smaller rectangles. The objects are then placed in the correct smaller rectangle node and the process continues for each of the smaller rectangles. Below is another picture of a quadtree (used for a different purpose than this assignment) which shows the tree representation alongside the 2D space it is working on:
For this assignment, you will make a QuadTree class as well as a Planet class. A Planet is simply a small circle located somewhere in space. You will create a large number of these Planets (more later on how to do that) and then have the QuadTree break the list of Planets into a tree.

Similar to the LinkedList or BST/AVL trees, your QuadTree will use a linked implementation. This means you will need a nested class called QTNode inside your QuadTree class to hold the node information. The QTNode class should contain something to represent its 2D spatial range (a Rectangle makes a good choice), a list (potentially empty if an interior node) of planets, and links to its children (0 or 4 of them).
As for the top-level QuadTree class you will need a constructor which takes in the size of the 2D space (i.e. 10k x 10k) as well as the maximum number of planets per leaf. The constructor should create an empty tree. Just as with the LinkedList and BST/AVL structures, it will also need an instance variable to reference the root of the tree. Additionally, you should include 3 public methods:

public void draw(Graphics g, int scale);
This method will draw the QuadTree gridlines on the Graphics object given. Scale is included since you probably don’t want a 10,000x10,000 pixel window to draw your 10,000x10,000 area. I use a scale of 20 and draw my 10,000x10,000 area in a 500x500 window.

public void addPlanet(Planet p);
We are going to setup our structure to build the quadtree by adding one planet at a time. So this should take in a single planet, figure out where in the tree that planet belongs, and add the planet to that leaf node’s list of planets. It of course will then need to check if there are too many planets in this node now and if so split that node into 4 more nodes. Note that this splitting process may end up being recursive in nature. You also may want to have the QTNode class have some methods in it to help you with the splitting process.

public List<Planet> findLocalPlanets(Rectangle r);
Whenever the user moves their ship (explained later) you will need to use your tree to determine the planets nearby the ship. This method takes in a rectangle that describes the area the ship can ‘see’. It should traverse through the nodes and gather up all the planets from leaf nodes that intersect this rectangle. Note that you will again probably want to have QTNode contain some methods to help you in this process. Also, I would not implement this method until you are sure your QuadTree is built correct and you can verify this visually.

Lastly, you will probably want a class to create a GUI to show your QuadTree in action. Create a top-level class (with a main) called SpaceSimulation. This class should first create an empty QuadTree. Then it should create the planets and add them to the QuadTree one at a time. You might also want to store them in a large List as well since later we will want both versions to demonstrate speedup. I suggest you create your Planets in clusters. I use 10 clusters each which contain 5000 Planets. The extent of space is 10k x 10k. After creating the Planets and QuadTree, create a GUI with a JFrame and add a MapPanel to it.

A MapPanel is simply an extension of JPanel (like BreakoutPanel or DrawingPanel) that allows you store and repaint all the Planets and Gridlines associated with the QuadTree.

This means you will need a method in your Planet class which can draw the Planet. Actually you will eventually need two such methods, one to draw the Planet on the POV panel (more later) and one to draw it on the Map panel. When it is drawn on the Map panel the locations will need to be scaled so that the entire 10k x 10k region of space fits onto a 500x500 Map panel. Your Planets will also have different sizes for their radii, however when downscaled by a factor of 20 this will create many radii < 1 which will not show up on the screen. So simply draw them with a radius of 1.
Note that in general when you draw a circle with a radius, you use fillOval. FillOval takes the upper left corner of a rectangle which contains the circle to be drawn. fillOval also takes the width and height of this rectangle. Note that the width and height of the rectangle is twice the radii of the circle you want drawn.

**Space Ship Simulation**

To complete the project you should add another JFrame and panel (POVPanel) to the SpaceSimulation class. This will pop up a second window which shows a space ship centered in the window. You should also create a SpaceShip class to hold the information about the location of the space ship.

In the basic version of the assignment, your SpaceShip needs to know its location in space as well as its size. Since you can draw the space ship as a simple circle, radius works as a size. The space ship should also know the size of the area it can see. For me this is a 500x500 window which matches the 500x500 window being displayed on the screen. The space ship should be drawn in the middle of the area it can see.

The POVPanel simply holds onto the ship and the planets that need rendered. When it draws these note that scaling will not be necessary as in the Map case, but shifting will be.

You should also add a KeyListener to this panel so that you can move the ship via keyboard commands. There are several methods in the KeyListener interface but the one you want is the keyPressed method. You can simply use getKeyChar() from the KeyEvent object to decide which key is being pressed. Use w/s/a/d for up/down/left/right. Additionally, you will eventually want to add q and e for quadtree rendering vs everything rendering which allows you to switch between rendering/colliding with just the local planets from the quadtree vs rendering/colliding with all the planets in the system.

Once you have both GUIs working at the same time and movement implemented you should add collision detection. Each time the SpaceShip moves it might collide with any of the Planets. You need to check this. Since Planets and the SpaceShip are both circles this check is easy and implement it as described in class.

**Additional Nodes**

The above describes the classes you must use as well as some methods you must use. However, many other pieces of the design in terms of methods and where they should go has been left unsaid. This is on purpose. It is your goal to try to put things in the best possible place. I would first just try and get things working. Once things are working you can focus on moving methods around so things are better designed.

If you have any questions on the details of what I want or design issues please ask me.
Submission

Submission will be similar to the last assignment. Jar file in the W: drive, printout of your source code, cover sheet, and screen shots of your GUIs. You may work with a partner.