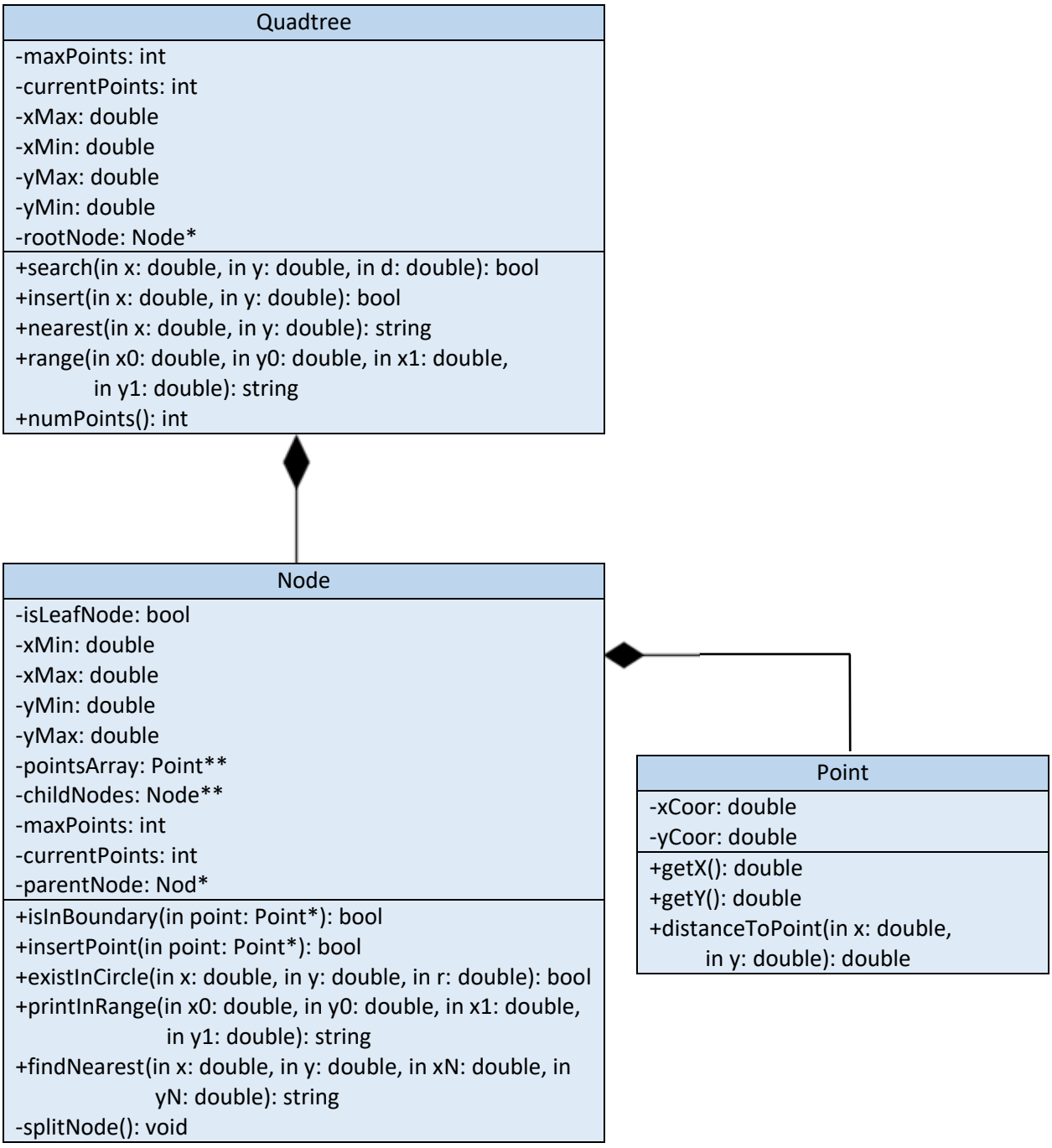
Quadtree (Project 3)

20992999 Kaixuan Chen

UML Diagram



Class Functions

Point Class

Each object represents a point (coordinate pair) stored in the Quadtree. It is stored in the Node class – specifically, in the Node objects with isLeafNode attribute equalling “true”. It has an helper function distanceToPoint(double x, double y) that calculates the distance from itself to (x, y).

Node Class

Each Node object is either a leaf node, or a node with four children nodes.

Function	Behavior
isInBoundary(Point* point) returns: bool	Checks if the point falls within the boundary of the Node.
insertPoint(Point* point) returns: bool	Insert a point into the Node. If this node is not a leaf node, find the correct children node containing the point and call insertPoint() again. If the node is full after insertion, call splitNode().
existInCircle(double x, double y, double r) returns: bool	Return whether or not a point with coordinates within this circle is stored in the quadtree. Using traversal identical to printInRange(), go through points in the rectange bounded by (x-r, y-r), (x+t, y+r). Return true if we encounter a point less than distance r away from (x, y), return false if we run out of points.
printInRange(double x0, double y0, doublex1, double y1) returns: string	If the rectangle bounded by (x0, y0), (x1, y1) falls outside the boundary of the node, cut the rectangle at the edge of the node. If the rectangle falls entirely outside the boundary, return “”. If called on a leaf node, return a string of all points within the rectangle. If no a leaf node, call this function on all child nodes, and return the concatenation of the results.
findNearest(double x, double y, double xN, double yN) returns: string	Go to the leaf node containing (x, y), iterate through the pointArray to find the nearest point. If one of x/y coordinate of the target is closer to the node boundary than to the point we found, or if we did not find a point, iterate through all neighbouring nodes to find the nearest point. Otherwise, return the point we found.
splitNode() returns: void	Split this node into four children nodes, and copy all points into the correct children node, deleting pointsArray at the end.

Quadtree Class

One Quadtree object is created in the main program, representing the entire Quadtree data structure.

Function	Behavior
search(double x, double y, double d) returns: bool	Call existInCircle(x, y, d) on rootNode and return the result.
insert(double x, double y) returns: bool	Dynamically create a new Point object and pass it into the insertPoint() function on the rootNode. If the insert is successful, increment currentPoints and return ture.
nearest(double x, double y) returns: string	If currentPoints is zero, return “no point exists”. Otherwise return the result of findNearest(x, y) on rootNode.
range(double x0, double y0, double x1, double y1) returns: string	Call printInRange(x0, y0, x1, y1) on rootNode. If the result is non-empty, return it, otherwise return “no points within range”.

Runtime Requirements

INSERT function in the Quadtree is essentially the insertPoint() function of Node class. There are two scenarios: (1) leaf node, in which case insertion is constant time since insertion to an array is constant time, and splitting a node and copying a constant number of points is also constant time. (2) non-leaf node, in which case insert function is called again on one of the children nodes. We are guaranteed to move down one level of the tree with every call of insertPoint(), thus the time for insertion into the Quadtree is O(D) where D is the depth of the tree.

RANGE functionality is essentially the printInRange() function of the Node class. If the range falls entirely within a single leaf node, we are guaranteed that the range falls entirely within a single children node at every level in the tree. This means that when a Node recursively calls printInRange() on all of its children, three of them will return nothing immediately. In order to reach the particular leaf node, we need to call printInRange() D times, where D is the depth of the tree. Since concatenating a finite number of points require only constant time, the range functionality has runtime O(D) whenever the range is within a single leaf node.