```java
/**
 *
 * @author Andrew Walters
 */
import java.io.File;

public class KdTreeST<Value> {
    //constants for deciding which key to search
    private static boolean cmpX = true;

    //root of tree
    private Node root;

    //node of tree
    private class Node<Value> {
        private Point2D key;       // sorted by key
        private Value val;         // associated data
        private Node left, right;  // left and right subtrees
        private int N;             // number of nodes in subtree

        //node constructor
        public Node(Point2D key, Value val, int N) {
            this.key = key;
            this.val = val;
            this.N = N;
        }//end node constructor
    }//end node definition

    // is the tree empty?
    public boolean isEmpty() {
        return size() == 0;
    }//end isEmpty

    // return number of key-value pairs in BST
    public int size() {
        return size(root);
    }//end size()

    // return number of key-value pairs in BST rooted at x
    private int size(Node x) {
        if (x == null) return 0;
        else return x.N;
    }//end size(Node)

    // add the point p to the tree or if it already exists, update
    public void insert(Point2D p, Value v) {
        if (v == null) {
            delete(p);
            return;
        }
        root = put(root, p, v, cmpX);
    }

    // helper method for insert
    private Node put(Node x, Point2D p, Value v, boolean dim) {
        //if the bottom of the tree has been reached, create a new node
        if (x == null) {
            return new Node(p, v, 1);
        }

        //compareTo by dimension
        int cmp = compareDim(p,x.key,dim);

        //go left
        if      (cmp <= 0) {
            x.left  = put(x.left,  p, v, !dim);
        }//end left

        //go right
        else if (cmp > 0) {
            x.right = put(x.right, p, v, !dim);
        }//end right

        //found node
        else if (compareDim(p,x.key,!dim) == 0) {
            x.val   = v;
        }//end found node

        //if this dim is equal but other dim is not, go left
        else {
            x.left  = put(x.left,  p, v, !dim);
        }

        x.N = 1 + size(x.left) + size(x.right);
        return x;
    }//end insert

    /*****************************************************************
      * Helper method for put that replaces compareTo
      * returns -1 for a < b
      * returns 1 for a > b
      * returns 0 for a == b
      * recursive calls to this method should contain !dim
      * **************************************************************/
    private int compareDim(Point2D a, Point2D b, boolean dim) {
        //compare x coordinates
        if(dim==cmpX) {
            return Double.compare(a.x(),b.x());
        }//end if
        //compare y coordinates
        else {
            return Double.compare(a.y(),b.y());
        }//end else
    }//end compareDim

    //returns value mapped to by p
    public Value get(Point2D p) {
        return get(root,p,cmpX);
```

```java
    }//end get

    private Value get(Node x, Point2D p, boolean dim) {

        //if bottom of tree has been reached
        if(x == null) {
            return null;
        }

        //if key has been found
        if(p.compareTo(x.key) == 0) {
            return (Value)x.val;
        }

        //compare by dim
        int cmp = compareDim(p,x.key,dim);

        if(cmp<=0) {
            return get(x.left,p,!dim);
        }
        else {
            return get(x.right,p,!dim);
        }
    }

    // does the ST contain the point p?
    public boolean contains(Point2D p) {
        return get(p) != null;
    }//end contains

    // draw points to standard draw
    public void draw() {
        draw(root,new RectHV(0,0,1,1),cmpX);
    }//end draw

    //helper method for draw
    public void draw(Node x, RectHV myRect, boolean dim) {

        //draw node
        //draw line
        StdDraw.setPenRadius(.002);
        if(dim == cmpX) {
            StdDraw.setPenColor(StdDraw.RED);
            StdDraw.line(x.key.x(),myRect.ymin(),x.key.x(),myRect.ymax());
        }
        else {
            StdDraw.setPenColor(StdDraw.BLUE);
            StdDraw.line(myRect.xmin(),x.key.y(),myRect.xmax(),x.key.y());
        }
        //draw point
        StdDraw.setPenRadius(.01);
        StdDraw.setPenColor(StdDraw.BLACK);
        x.key.draw();
        //create new rectangles for subtrees
        RectHV leftRect,rightRect;
        if(dim == cmpX) {
            leftRect = new RectHV(myRect.xmin(),myRect.ymin(),x.key.x(),myRect.ymax());
            rightRect = new RectHV(x.key.x(),myRect.ymin(),myRect.xmax(),myRect.ymax());
        }//end x
        else {
            leftRect = new RectHV(myRect.xmin(),myRect.ymin(),myRect.xmax(),x.key.y());
            rightRect = new RectHV(myRect.xmin(),x.key.y(),myRect.xmax(),myRect.ymax());
        }//end y

        //draw left subtree if it exists
        if(x.left != null) {
            draw(x.left,leftRect,!dim);
        }
        //draw right subtree if it exists
        if(x.right != null) {
            draw(x.right,rightRect,!dim);
        }
    }

    //all points in the ST that are inside the rectangle
    public Iterable<Point2D> range(RectHV rect) {
        return range(rect, new RectHV(0,0,1,1), root, cmpX);
    }//end iterable

    //helper method for range
    private Iterable<Point2D> range(RectHV rect, RectHV myRect, Node x, boolean dim) {

        //create queue to hold points
        Queue<Point2D> queue = new Queue<Point2D>();

        //if bottom of the tree was reached
        if(x == null) {return queue;}

        //determine if the current node is contained in rectangle
        if(rect.contains(x.key)) {
            queue.enqueue(x.key);
        }//end check for this point

        //create right and left rectangles
        RectHV leftRect,rightRect;
        if(dim == cmpX) {
            leftRect = new RectHV(myRect.xmin(),myRect.ymin(),x.key.x(),myRect.ymax());
            rightRect = new RectHV(x.key.x(),myRect.ymin(),myRect.xmax(),myRect.ymax());
        }//end x
        else {
            leftRect = new RectHV(myRect.xmin(),myRect.ymin(),myRect.xmax(),x.key.y());
            rightRect = new RectHV(myRect.xmin(),x.key.y(),myRect.xmax(),myRect.ymax());
        }//end y

        //search left subtree if it intersects the rectangle
        if(rect.intersects(leftRect)) {
            Iterable<Point2D> list = range(rect,leftRect,x.left,!dim);
            for(Point2D key: list) {
                queue.enqueue(key);
            }//end loop
        }//end search left subtree

        //search right subtree if it intersects the rectangle
        if(rect.intersects(rightRect)) {
            Iterable<Point2D> list = range(rect,rightRect,x.right,!dim);
            for(Point2D key: list) {
                queue.enqueue(key);
            }//end loop
        }//end search right subtree

        return queue;

    }//end range

    public Point2D nearest(Point2D p) {
        RectHV rect = new RectHV(0,0,1,1);
        return nearest(root, p, rect, cmpX);
    }

    // a nearest neighbor in the ST to p; null if set is empty
    private Point2D nearest(Node x, Point2D p, RectHV rect, boolean dim) {

        //local variables
        Point2D incumbent = x.key;
        Point2D challenger;
        double dist = incumbent.distanceTo(p);

        //create right and left rectangles
        RectHV leftRect,rightRect;
        if(dim == cmpX) {
            leftRect = new RectHV(rect.xmin(),rect.ymin(),x.key.x(),rect.ymax());
            rightRect = new RectHV(x.key.x(),rect.ymin(),rect.xmax(),rect.ymax());
        }//end x
        else {
            leftRect = new RectHV(rect.xmin(),rect.ymin(),rect.xmax(),x.key.y());
            rightRect = new RectHV(rect.xmin(),x.key.y(),rect.xmax(),rect.ymax());
        }//end y

        //compare this node to p
        int cmp = compareDim(p,x.key,dim);

        //algorithm for nearest:
        //first search the subtree that contains the point if it is a non-null subtree
        //the search should return the point in the subtree nearest to the point
        //after the first subtree has been searched, check if the distance to the other subtree
        is greater than the point returned by the first subtree
        //if the distance is greater, that subtree may be pruned and the value of the first
        subtree returned
        //otherwise the second subtree should be searched

        //if the point lies in the left subtree
        if(cmp <= 0) {
            //and the left subtree is not empty
            if (x.left != null) {
                //then set challenger to the nearest point in the left subtree
                //if that point is furher away than the current node, change near to the
                current node
                if ((challenger = nearest(x.left, p, leftRect, !dim)).distanceTo(p) < dist) {
                    incumbent = challenger;
                    dist = challenger.distanceTo(p);
                }
            }//end searching left subtree

            //if the right subtree is not empty
            if(x.right != null) {
                //and the right subtree could contain the nearest point
                if (rightRect.distanceTo(p) < dist) {
                    if ((challenger = nearest(x.right, p, rightRect, !dim)).distanceTo(p) < dist
                    ) {
                        incumbent = challenger;
                        dist = challenger.distanceTo(p);
                    }
                }
            }//end searching right subtree
        }//end point contained in left subtree

        //if the point lies in the right subtree
        else{
            //and the left subtree is not empty
            if (x.right != null) {
                //then set challenger to the nearest point in the right subtree
                //if that point is furher away than the current node, change near to the
                current node
                if ((challenger = nearest(x.right, p, rightRect, !dim)).distanceTo(p) < dist) {
                    incumbent = challenger;
                    dist = challenger.distanceTo(p);
                }
            }//end searching left subtree

            //if the left subtree is not empty
            if(x.left != null) {
                //and the left subtree could contain the nearest point
                if (leftRect.distanceTo(p) < dist) {
                    if ((challenger = nearest(x.left, p, leftRect, !dim)).distanceTo(p) < dist) {
                        incumbent = challenger;
                        dist = challenger.distanceTo(p);
                    }
                }
            }//end searching left subtree
        }//end point contained in the right subtree
        //default case (should never be reached)
        return incumbent;
    }//end nearest

    public void delete(Point2D p) {
        root = delete(root, p, cmpX);
    }

    private Node delete(Node x, Point2D p, boolean dim) {

        //if node not found
        if (x == null) return null;

        //if node found
        if (p.compareTo(x.key) == 0) {
            if (x.right == null) return x.left;
            if (x.left  == null) return x.right;
            Node t = x;
            //x = min(t.right);
            //x.right = deleteMin(t.right);
            x.left = t.left;
            x.N = size(x.left) + size(x.right) + 1;
            return x;
        }
        return x;
    }//end delete

    // unit testing of the methods (not graded)
    public static void main(String[] args) {

        KdTreeST<Integer> points = new KdTreeST<Integer>();

        double[] x = new double[800];
        double[] y = new double[800];

        //test constructor, insert, and draw
        for (int i = 0; i < 800; i++) {
            x[i] = StdRandom.uniform(0.0,1.0);
            y[i] = StdRandom.uniform(0.0,1.0);
        }

        Stopwatch watch = new Stopwatch();

        for (int i = 0; i < 800; i++) {
            points.insert(new Point2D(x[i], y[i]),1);
        }

        System.out.println("elapsed time: " + watch.elapsedTime());

        StdDraw.setCanvasSize(600, 600);
        StdDraw.setXscale(0, 1);
        StdDraw.setYscale(0, 1);
        StdDraw.setPenRadius(.01);
        points.draw();

        //test range
        RectHV rect = new RectHV(.2,.2,.4,.4);
        StdDraw.setPenColor(StdDraw.GRAY);
        StdDraw.setPenRadius(.002);
        rect.draw();
        Iterable<Point2D> list = points.range(rect);
        StdDraw.setPenColor(StdDraw.PINK);
        StdDraw.setPenRadius(.02);
        for(Point2D key: list) {
            key.draw();
        }//end loop

        //test nearest
        Point2D center = new Point2D(.5, .5);
        StdDraw.setPenColor(StdDraw.CYAN);
        StdDraw.setPenRadius(.02);
        center.draw();
        StdDraw.setPenRadius(.005);
        center.drawTo(points.nearest(center));

    }//end main

}
```