File - /Users/moe/Downloads/cs61b/fa20-s1319/proj2ab/bearmaps/KDTree.java

```java
package bearmaps;
import java.util.List;
//@Josh Hug video support

public class KDTree implements PointSet {
    private static final boolean HORIZONTAL = false;
    private static final boolean VERTICAL = true;
    private Node root;

    private class Node {
        private Point p;
        private boolean orientation;
        private Node left; //down child
        private Node right; //up child

        Node(Point givenp, boolean po) {
            p = givenp;
            orientation = po;
        }
    }
    public KDTree(List<Point> points) {
        for (Point p : points) {
            root = add(p, root, HORIZONTAL);
        }
    }
    private Node add(Point p, Node n, boolean orientation) {
        if (n == null) {
            return new Node(p, orientation);
        }
        if (p.equals(n.p)) {
            return n;
        }
        int cmp = comparePoints(p, n.p, orientation);
        if (cmp < 0) {
            n.left = add(p, n.left, !orientation);
        } else if (cmp >= 0) {
            n.right = add(p, n.right, !orientation);
        }
        return n;
    }
    private int comparePoints(Point a, Point b, boolean orientation) {
        if (orientation == HORIZONTAL) {
            return Double.compare(a.getX(), b.getX());
        } else {
            return Double.compare(a.getY(), b.getY());
        }
    }
    @Override
    public Point nearest(double x, double y) {
        Point parameterPoint = new Point(x, y);
        Node closest = nearest(root, parameterPoint, root);
        return closest.p;
    }
```

```
54      private Node nearest(Node n, Point goal, Node best) {
55          Node goodSide;
56          Node badSide;
57          if (n == null) {
58              return best;
59          }
60          if (Point.distance(n.p, goal) < Point.distance(best.p, goal)) {
61              best = n;
62          }
63          int cmp = comparePoints(goal, n.p, n.orientation);
64          if (cmp < 0) {
65              goodSide = n.left;
66              badSide = n.right;
67          } else {
68              goodSide = n.right;
69              badSide = n.left;
70          }
71          best = nearest(goodSide, goal, best);
72          if (badSideBest(n, goal, best)) {
73              best = nearest(badSide, goal, best);
74          }
75          return best;
76      }
77      private boolean badSideBest(Node n, Point goal, Node best) {
78          Point bsPoint;
79          if (n.orientation == HORIZONTAL) {
80              bsPoint = new Point(n.p.getX(), goal.getY());
81          } else {
82              bsPoint = new Point(goal.getX(), n.p.getY());
83          }
84          return Point.distance(bsPoint, goal) < Point.distance(best.p, goal);
85      }
86  }
87
```