

```

package quadtree;

/**
 * Class for quadtrees to store points within a given rectangular area. Allows
 * inserting points and getting all points for a given rectangle. Quadtrees can
 * be drawn using <code>Window</code>.
 */
public class QuadTree {

    private QuadNode root;

    /**
     * Constructor for creating a quadtree for a given rectangular area.
     *
     * @param x
     *         the x coordinate of the rectangular area
     * @param y
     *         the y coordinate of the rectangular area
     * @param w
     *         the width of the rectangular area
     * @param h
     *         the height of the rectangular area
     */
    public QuadTree(int x, int y, int w, int h) {
        super();
        root = new QuadLeaf(x, y, w, h);
    }

    /**
     * Inserts a point into this quadtree
     *
     * @param p
     *         the point to be inserted
     * @throws QuadTreeInsertionException
     *         throws when the point is not within the rectangular area.
     */
    public void insert(Point p) throws QuadTreeInsertionException {
        root = root.insert(p);
    }

    /**
     * Returns all the points stored in this nodes or its subnodes which are
     * within the given rectangle rect
     *
     * @param rect
     *         the rectangle the points should lie in.
     * @return the array of points contained in r.
     */
    public Point[] findPoints(Rect rect) {
        return root.findPoints(rect);
    }

    /**
     * Draws this node on <code>Window</code>
     */
    public void draw() {
        root.draw();
    }
}

```

```

package quadtree;
/**
 * Abstract class for quadtree nodes.
 */
public abstract class QuadNode {
    /**
     * The rectangle for this node
     */
    protected Rect rect;

    /**
     * Constructor for a Quadtree node with coordinates and size for the
     * rectangle
     *
     * @param x
     *         the x coordinate for the rectangle
     * @param y
     *         the y coordinate for the rectangle
     * @param w
     *         the width for the rectangle
     * @param h
     *         the height for the rectangle
     */
    public QuadNode(int x, int y, int w, int h) {
        super();
        rect = new Rect(x, y, w, h);
    }

    /**
     * Inserts the point into this node. Insertion may result in splitting a
     * leaf into a branch. Thus the node is returned from this operation which
     * may be different from this node. It allows the parent node to reset this
     * node.
     *
     * @param point
     *         the point to be inserted
     * @return the node, which may be this node or a new branch
     * @throws QuadTreeInsertionException
     *         thrown when point is not within the rect of this node
     */
    public abstract QuadNode insert(Point point)
        throws QuadTreeInsertionException;

    /**
     * Returns all the points stored in this nodes or its subnodes which are
     * within the given rectable r
     *
     * @param r the rectangle the points should lie in.
     * @return the array of points contained in r.
     */
    public abstract Point[] findPoints(Rect r);

    /**
     * Draws this node on <code>Window</code>
     */
    public void draw() {
        rect.draw();
    }

    /**
     * Tests if the point p is within the rectangle of this node
     *
     * @param p
     *         the point
     * @return true, if the point p is contained in the rectangle of this node,
     *         false otherwise
     */
    protected boolean contains(Point p) {
        return rect.contains(p);
    }
}

```

```

package quadtree;

/**
 * Class for leaves in a quadtree. Can store a set of points in an array.
 */
public class QuadLeaf extends QuadNode {

    /**
     * maximal number of points which can be stored in this leaf node.
     */
    static final int CAP = 2;

    /**
     * Array for storing the points.
     */

    private Point[] points;
    /**
     * Number of points which are stored in this leaf node.
     */
    private int nPoints;

    /**
     * Constructor for a Quadtree leaf node with coordinates and size for the
     * rectangle
     *
     * @param x
     *         the x coordinate for the rectangle
     * @param y
     *         the y coordinate for the rectangle
     * @param w
     *         the width for the rectangle
     * @param h
     *         the height for the rectangle
     */
    public QuadLeaf(int x, int y, int w, int h) {
        super(x, y, w, h);
        points = new Point[CAP];
        nPoints = 0;
    }

    /**
     * (non-Javadoc)
     *
     * @see quadtree.QuadNode#insert(quadtree.Point)
     */
    @Override
    public QuadNode insert(Point point) throws QuadTreeInsertionException {
        if (!rect.contains(point)) {
            throw new QuadTreeInsertionException("Point not within node ",
                this, point);
        }
        if (nPoints == CAP) {
            QuadBranch branch = new QuadBranch(rect.x, rect.y, rect.w, rect.h);
            for (Point p : points) {
                branch.insert(p);
            }
            branch.insert(point);
            return branch;
        } else {
            points[nPoints] = point;
            nPoints++;
            return this;
        }
    }

    /**
     * (non-Javadoc)
     *

```

```

    * @see quadtree.QuadNode#pointsWithin(quadtree.Rect)
    */
    @Override
    public Point[] findPoints(Rect rect) {
        Point[] ps = new Point[nPoints];
        int n = 0;
        for (int i = 0; i < nPoints; i++) {
            if (rect.contains(points[i])) {
                ps[n] = points[i];
                n++;
            }
        }
        Point[] result = new Point[n];
        for (int i = 0; i < n; i++) {
            result[i] = ps[i];
        }
        return result;
    }

    /*
     * (non-Javadoc)
     *
     * @see quadtree.QuadNode#draw()
     */
    @Override
    public void draw() {
        super.draw();
        for (int i = 0; i < nPoints; i++) {
            points[i].draw();
        }
    }
}
}

```

```

package quadtree;

/**
 * Class for inner branch nodes in a quadtree. Branch nodes have four sub nodes
 * for the four rectangular regions representing the northwest, northeast,
 * southeast, and southwest subareas of equal size.
 */
public class QuadBranch extends QuadNode {

    private static final int NW = 0;
    private static final int NE = 1;
    private static final int SE = 2;
    private static final int SW = 3;

    private QuadNode[] subNodes = new QuadNode[4];

    public QuadBranch(int x, int y, int w, int h) {
        super(x, y, w, h);
        int wh = w / 2;
        int hh = h / 2;
        subNodes[NW] = new QuadLeaf(rect.x, rect.y, wh, hh);
        subNodes[NE] = new QuadLeaf(rect.x + wh, rect.y, rect.w - wh, hh);
        subNodes[SE] = new QuadLeaf(rect.x + wh, rect.y + hh, rect.w - wh,
            rect.h - hh);
        subNodes[SW] = new QuadLeaf(rect.x, rect.y + hh, wh, rect.h - hh);
    }

    /*
     * (non-Javadoc)
     *
     * @see quadtree.QuadNode#insert(quadtree.Point)
     */
    @Override
    public QuadNode insert(Point point) throws QuadTreeInsertionException {
        if (!rect.contains(point)) {
            throw new QuadTreeInsertionException("Point not within node ",
                this, point);
        }
        for (int i = 0; i < 4; i++) {
            if (subNodes[i].contains(point)) {
                QuadNode n = subNodes[i].insert(point);
                // n may have been split into a branch with subnodes
                // so set subNodes[i] to returned node of insert operation
                subNodes[i] = n;

                break;
            }
        }
        return this;
    }

    /*
     * (non-Javadoc)
     *
     * @see quadtree.QuadNode#findPoints(quadtree.Rect)
     */
    @Override
    public Point[] findPoints(Rect r) {
        Point[][] results = new Point[4][];
        int n = 0;
        for (int i = 0; i < 4; i++) {
            if (subNodes[i].rect.overlaps(r)) {
                results[i] = subNodes[i].findPoints(r);
            } else {
                results[i] = new Point[0];
            }
            n = n + results[i].length;
        }
        Point[] result = new Point[n];

        int k = 0;

```

```
        for (int i = 0; i < 4; i++) {
            for (int j = 0; j < results[i].length; j++) {
                result[k] = results[i][j];
                k++;
            }
        }
        return result;
    }

    @Override
    public void draw() {
        super.draw();
        for (int i = 0; i < 4; i++) {
            subNodes[i].draw();
        }
    }
}
```

```

package quadtree;

import inout.Window;
import java.awt.Color;

/**
 * A Point with x and y coordinate
 */
public class Point {
    /**
     * The x coordinate
     */
    public int x;
    /**
     * The y coordinate
     */
    public int y;

    /**
     * Constructor with x and y coordinates.
     *
     * @param x
     *         the x coordinate
     * @param y
     *         the x coordinate
     */
    public Point(int x, int y) {
        this.x = x;
        this.y = y;
    }

    /**
     * Draws the point in black on <code>Window</code>.
     */
    public void draw() {
        Window.fillRectangle(x - 1, y - 1, 2, 2, Color.black);
    }

    /**
     * Draws the point in the given color on <code>Window</code>.
     *
     * @param color the drawing color
     */
    public void draw(Color color) {
        Window.fillRectangle(x - 1, y - 1, 2, 2, color);
    }

    /*
     * (non-Javadoc)
     * @see java.lang.Object#equals(java.lang.Object)
     */
    @Override
    public boolean equals(Object obj) {
        if (!(obj instanceof Point)) {
            return false;
        }
        Point p = (Point) obj;
        return x == p.x && y == p.y;
    }

    /*
     * (non-Javadoc)
     *
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString() {
        return String.format("(%1$3d/%2$3d)", x, y);
    }
}

```

```

}

```

```

package quadtree;

import inout.Window;

import java.awt.Color;

/**
 * A rectangle with x, y coordinates and width and height
 */
public class Rect {

    /**
     * The x coordinate of the origin of the rectangle
     */
    public int x;

    /**
     * The y coordinate of the origin of the rectangle
     */
    public int y;

    /**
     * The width of the rectangle
     */
    public int w;

    /**
     * The height of the rectangle
     */
    public int h;

    /**
     * Constructor with x, y, width and height
     *
     * @param x
     *         the x coordinate
     * @param y
     *         the y coordinate
     * @param w
     *         the width of the rectangle
     * @param h
     *         the height of the rectangle
     */
    public Rect(int x, int y, int w, int h) {
        super();
        this.x = x;
        this.y = y;
        this.w = w;
        this.h = h;
    }

    /**
     * Tests if the given point p is within this rectangle
     *
     * @param p
     *         the point
     * @return true if the point p is within this rectangle, false otherwise
     */
    public boolean contains(Point p) {
        return (x <= p.x && p.x < x + w && y <= p.y && p.y < y + h);
    }

    /**
     * Tests if this rectangle overlaps with the rectangle <code>other</code>
     *
     * @param other
     *         the other rectangle
     * @return true, if the rectangle overlap, false otherwise
     */
    public boolean overlaps(Rect other) {

```

```

        return !(other.x + other.w <= x || other.x > x + w
                || other.y + other.h <= y || other.y > y + h);
    }

    /**
     * Draws this rectangle on <code>Window</code> in black
     */
    public void draw() {
        Window.drawRectangle(x, y, w, h);
    }

    /**
     * Draws this rectangle on <code>Window</code> in the given color
     *
     * @param color
     *         the drawing color
     */
    public void draw(Color color) {
        Window.drawRectangle(x, y, w, h, color);
    }

    /**
     * Draws this rectangle on <code>Window</code> in the given color
     *
     * @param color
     *         the drawing color
     */
    public void fill(Color color) {
        Window.fillRectangle(x, y, w, h, color);
    }

    /**
     * (non-Javadoc)
     *
     * @see java.lang.Object#toString()
     */
    @Override
    public String toString() {

        return String.format("(%1$3d/%2$3d) w %3$3d h %4$3d", x, y, w, h);
    }
}

```

```

package quadtree.test;

import inout.In;
import inout.Out;
import inout.Window;

import java.awt.Color;

import quadtree.Point;
import quadtree.QuadTree;
import quadtree.QuadTreeInsertionException;
import quadtree.Rect;

/**
 * Class to show and test the quadtree implementation.
 */
public class QuadTreeTest {

    /**
     * The main test method. Creates a quadtree and randomly inserts a set of
     * points. Reads a search rectangle from the user and puts out all points
     * within the search rectangle. Draws the quadtree with points as well as
     * the search rectangle and the points found on <code>Window</code>.
     *
     * @param args
     */
    public static void main(String[] args) {

        Window.open();

        QuadTree tree = new QuadTree(8, 8, 510, 510);
        try {
            for (int i = 0; i < 20; i++) {
                tree.insert(new Point((int) (Math.random() * 50) + 11,
                    (int) (Math.random() * 50) + 11));
            }
            for (int i = 0; i < 10; i++) {
                tree.insert(new Point((int) (Math.random() * 100) + 11,
                    (int) (Math.random() * 100) + 11));
            }
            for (int i = 0; i < 10; i++) {
                tree.insert(new Point((int) (Math.random() * 300) + 11,
                    (int) (Math.random() * 300) + 11));
            }
            for (int i = 0; i < 4; i++) {
                tree.insert(new Point((int) (Math.random() * 400) + 11,
                    (int) (Math.random() * 400) + 11));
            }
            for (int i = 0; i < 2; i++) {
                tree.insert(new Point((int) (Math.random() * 500) + 11,
                    (int) (Math.random() * 500) + 11));
            }
        } catch (QuadTreeInsertionException e) {
        }

        tree.draw();
        searchIn(tree);
    }

    /**
     * Does the search for points within the rectangle. Reads rectangles from
     * the user, finds the points within the rectangle and puts out the result.
     *
     * @param tree
     *         the quadtree
     */
    private static void searchIn(QuadTree tree) {
        Rect r = readRect();
        while (r != null) {
            Point[] points = tree.findPoints(r);
            Window.clear();
        }
    }
}

```

```

        r.fill(Color.LIGHT_GRAY);
        tree.draw();
        outputPoints(points);
        r = readRect();
    }
}

/**
 * Puts out the points found. It draws the points in red on
 * <code>Window</code> and prints out the points found with
 * <code>Out</code>.
 *
 * @param points
 *         the points found
 */
private static void outputPoints(Point[] points) {
    Out.println("Punkte gefunden: ");
    for (Point p : points) {
        Out.println(p.toString());
        p.draw(Color.RED);
    }
    Out.println();
}

/**
 * Reads the coordinates for the rectangle using <code>In</code>.
 *
 * @return the rectangle created from the coordinates read in.
 */
private static Rect readRect() {
    Out.print("Bitte Suchgebiet eingeben (x y w h): ");
    int x = In.readInt();
    if (x < 0) {
        return null;
    }
    int y = In.readInt();
    int w = In.readInt();
    int h = In.readInt();
    if (x < 0 || y < 0 || w < 0 || h < 0) {
        return null;
    }
    Rect r = new Rect(x, y, w, h);
    return r;
}
}

```