Consider the following code that contains two ADTs: a mutable 2-D Point class and a PointSet class which is essentially identical to the CharSet class used as an example in lecture. There is also a Main class with a main method that creates instances of these classes and calls some of their methods. Answer questions about this code on the following page.

/** 2-d (x,y) mutable point on the plane */
class Point {
    private int x, y;
    public Point(int x, int y) {
        this.x = x; this.y = y;
    }
    public void setX(int x) { this.x = x; }
    public void setY(int y) { this.y = y; }
}

/** Mutable set of 2-D Points */
class PointSet {
    // RI: elts has no nulls and no duplicates
    private List<Point> elts;

    public PointSet() { elts = new ArrayList<Point>(); }

    /** add a point to this */
    public void add(Point p) {
        // ... check for duplicates omitted
        elts.add(p);
    }

    // Three methods that return a list of the set elements
    public List<Point> getElts1() { return elts; }
    public List<Point> getElts2() { return new ArrayList<Point>(elts); }
    public List<Point> getElts3() {
        return Collections.unmodifiableList(elts);
    }
}

class Main {
    public static void main(String[] args) {
        PointSet ps = new PointSet();
        ps.add(new Point(1,2));
        ps.add(new Point(17,42));

        List<Point> e1 = ps.getElts1();
        List<Point> e2 = ps.getElts2();
        List<Point> e3 = ps.getElts3();
        // draw a diagram of memory when execution reaches here
    }
}
1. **(CSE 143 review)** Draw a diagram of memory showing all the variables and objects that exist at the end of method `main` when it is executed. Be sure you clearly show the distinction between local variables in `main` and Java objects referenced by those variables and by other objects. The results of the first two assignments, which create a `PointSet` and add a `Point` to it, are given below to help you get started. Add to this diagram to show the effect of the rest of the code.

```
ps          e1          e2          e3
         \       \       \       \       \       \       \       \       \       \       \       \       \       \\
          \       \       \       \       \       \       \       \       \       \       \       \       \       \\
         \       \       \       \       \       \       \       \       \       \       \       \       \       \\
          \       \       \       \       \       \       \       \       \       \       \       \       \       \\
         \       \       \       \       \       \       \       \       \       \       \       \       \       \\

main      elts
         \      \      \      \      \      \      \      \      \      \      \      \      \      \\
          \      \      \      \      \      \      \      \      \      \      \      \      \      \\
         \      \      \      \      \      \      \      \      \      \      \      \      \      \\
          \      \      \      \      \      \      \      \      \      \      \      \      \      \\

x 1 y 2   x 17 y 42

...       ...
```

2. Do any of the three implementations of method `getElts` (`getElts1`, `getElts2`, `getElts3`) have potential representation exposure problems? If so, explain which method(s) have the problem and why. (Briefly)

Yes, all three methods have representation exposure problems.

`getElts1` returns a reference to the private `PointSet` list `elts` to the caller, which means the caller can modify that list.

Since `Point` objects are mutable, all three methods return lists that refer to the original `Point` objects, which means the caller can modify any of the `Points`, changing the ones that are stored in (referenced by) the original `PointSet`. 