```
Rubin Peci
N15878268
September 11th, 2019
Homework #1
CS 102 - Data Structures

1a.
//NonDestRotate returns an entirely new instance of Rectangle
//With swapped x and y spans, which creates a new Rectangle that is rotated 90 degrees
public Rectangle NonDestRotate() {
        return new Rectangle(getYSpan(), getXSpan());
}

//DestRotate returns the exact same Rectangle
//It modifies the spans of this rectangle, and it doesn't create a copy
public Rectangle DestRotate() {
        this.setSpans(getYSpan(), getXSpan());
        return this;
}

public LocatedRect NonDestRotate() {
                return new LocatedRect(right(), right() + ySpan, yL, yL + xSpan);
        }

public LocatedRect DestRotate() {
        setCorner(right(), yL);
        double tmp = xSpan;
        xSpan = ySpan;
        ySpan = tmp;
        return this;
}

1b.
public class TestRotate {

        public static void main (String args[]) {
                //Create a 2 by 5 Rectangle
                Rectangle rect1 = new Rectangle(2.0, 5.0);
                System.out.println("Rectangle 1: " + rect1);
                //Testing out NonDestRotate()
                System.out.println("Testing NonDestRotate()");
                System.out.println("Rotated rect1: " + rect1.NonDestRotate());
                System.out.println("Rectangle 1: " + rect1);
                //Testing out DestRotate()
                System.out.println("Testing out DestRotate()");
                System.out.println("Rotated (destructively) rect1: " + rect1.DestRotate());
                System.out.println("Rectangle 1: " + rect1);

                //Located Rectangle Testing
                LocatedRect lrect1 = new LocatedRect(1.0, 3.0, 2.0, 5.0);
                System.out.println("LocatedRectangle 1: " + lrect1);
                //Testing out NonDestRotate()
                System.out.println("Testing NonDestRotate()");
                System.out.println("Rotated lrect1: " + lrect1.NonDestRotate());
                System.out.println("Rectangle 1: " + lrect1);
                //Testing out DestRotate()
                System.out.println("Testing out DestRotate()");
                System.out.println("Rotated (destructively) lrect1: " +
lrect1.DestRotate());
                System.out.println("Rectangle 1: " + lrect1);

        }
}
```

2a.

```
public class Square extends Rectangle {
        public Square(double side) {
                xSpan = side;
                ySpan = side;
        }

        public Square() {
                this(1.0);
        }

        public double getSide() {
                return xSpan;
        }

        public double setSide(double side) {
                double tmp = xSpan;
                xSpan = side;
                return tmp;
        }

        public void setSpans(double x, double y) {
                System.out.println("You may not use setSpans to set the sides of a
square!");
        }

        public String toString() {                    // Printable form
         return "Square["+ xSpan + ", " + ySpan + "]";
    }
}
```

2b.

```
public class LocatedSquare extends Square {
        private double xL, yL;

        public LocatedSquare() {
                setCorner(0.0, 0.0);
        }

        public LocatedSquare(double x1, double y1, double side) {
                super(side);
                xL = x1;
                yL = y1;
        }

    public void setCorner(double xa, double ya) {
        xL=xa;  yL=ya;
        }

        public double right() { return xL + xSpan; }
        public double left() { return xL; }
        public double top() { return yL + ySpan; }
        public double bottom() { return yL; }


    public String toString() {
         return "LS[" + left() + ", " + right() + ". " +
                              bottom() + ", " + top() + "]"; }
         // toString() gives a printable format. Note this overrides
         // the toString() method in Square.
}
```

2c. You cannot do Part B by having LocatedSquare extend both Square and LocatedRect because you cannot extend more than one class in Java. You would want to use an interface in this specific instance.

```
3a. private Person spouse;

3b. public Person getSpouse() { return spouse; }

3c. public boolean marry(Person q) {
                if (q == null) {
                        System.out.println("Spouse is null");
                        return false;
                } else if (this.getSpouse() != null || q.getSpouse() != null) {
                        System.out.println("One person is already married!");
                        return false;
                } else if (this.getParent1() == q || this.getParent2() == q ||
q.getParent1() == this || q.getParent2() == this) {
                        System.out.println("What is wrong with you...");
                        return false;
                }
                this.spouse = q;
                q.spouse = this;
                return true;
        }

3d.     public void divorce() {
                if (this.getSpouse() != null)
                       this.getSpouse().spouse = null;
                this.spouse = null;
        }
```

There is no point in checking that the owner of the method marry() isn't null because if it is, it will cause an error prior to the function being run. It will not work even if we have a check because it will crash at runtime.