Problem 1

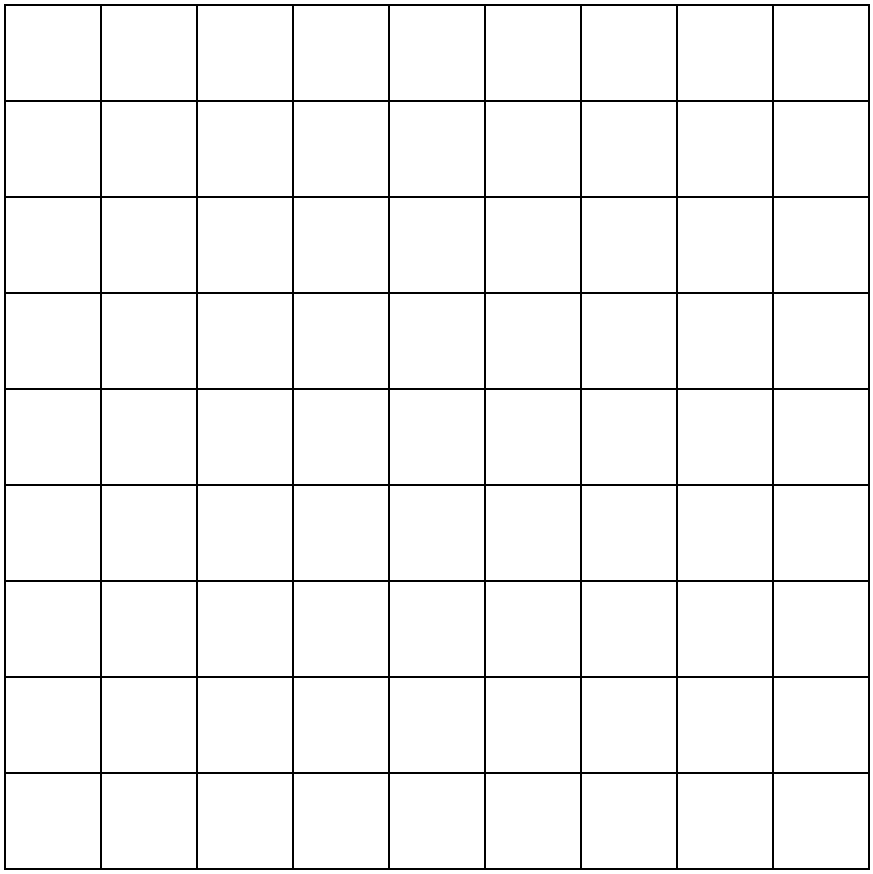
Suppose you are given the following code:

```
public class horseShoeWorld extends BuggleWorld {
  public void run() {
    horseShoeBuggle a = new horseShoeBuggle();
    horseShoeBuggle b = new horseShoeBuggle();
    horseShoeBuggle c = new horseShoeBuggle();

    a.setPosition(new Point(4,6));
    a.setColor(Color.red);
    a.setHeading(Direction.WEST);
    a.forward();
    a.setColor(Color.green);
    a.forward(2);
    a.right();
    a.forward(2);
    a.right();
    a.forward(3);
    a.setColor(Color.red);
    a.forward();

    b.setPosition(new Point(2,4));
    b.setColor(Color.red);
    b.setHeading(Direction.EAST);
    b.forward();
    b.setColor(Color.pink);
    b.forward(3);
    b.right();
    b.forward(2);
    b.right();
    b.forward(4);
    b.setColor(Color.red);
    b.forward();

    c.setPosition(new Point(4,5));
    c.setColor(Color.red);
    c.setHeading(Direction.NORTH);
    c.forward();
    c.setColor(Color.orange);
    c.forward(1);
    c.right();
    c.forward(2);
    c.right();
    c.forward(2);
    c.setColor(Color.red);
    c.forward();
  }
}
```

Also, assume you are given the following horseShoeBuggle class declaration. It is initially empty. Note that under these conditions run() works properly.

```
class horseShoeBuggle extends Buggle {
}
```

Part a) A grid is provided below, in which you should sketch the pattern that is drawn by the horseShoeBuggles. In your sketch, you should indicate that a square has been colored a particular color by the using the capital letter corresponding to the first letter of the color (ie R=red, G=green, etc). You should also indicate the final position and direction of each horseShoeBuggle in your diagram. To do so, represent each horseshoeBuggle by a triangle and place the triangle in the grid cell of the horseShoeBuggle's final position. Also, the triangle should point in the direction (North, South, East or West) in which the horseShoeBuggle ends up heading. Do not worry about indicating the color of the horseShoeBuggle when it is in its final position.



Part b) Notice that there is a pattern to the code in the run() method. Write a method in the horseShoeBuggle class named **horseShoe** to capture this pattern.

Notice that in the example given, there are horseshoes of length 3, 4 and 5. When you write your method, be sure to write it such that it is possible to invoke the method and make horseshoes of any length (provided there is space in the grid).

```
class horseShoeBuggle extends Buggle {
// fill in body of class declaration below




}
```

Part c) Given the methods defined in Part b, rewrite the horseShoeBuggle class run method to take advantage of those methods.

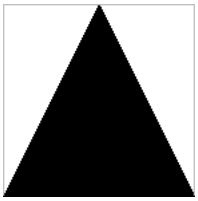
```
class horseShoeBuggleWorld extends BuggleWorld {
  public void run () {









  }
}
```

Problem 2

Suppose you're given the following black box method:

```
  public Picture triangle (Color c)
```

that creates the Picture shown below (the light gray outline just indicates the border of the picture frame but is not part of the Picture itself)



Additionally, you're given the following methods:

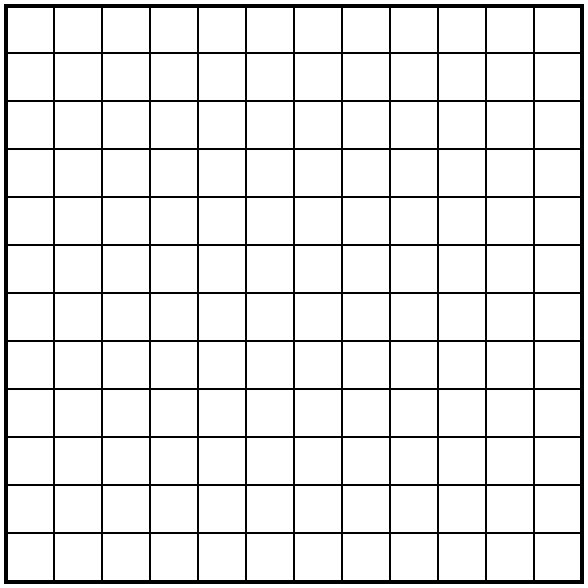
```
public Picture fourPics (Picture p1, Picture p2,
                         Picture p3, Picture p4) {
  return above(beside(p1,p2), beside(p3,p4));
}

public Picture fourOverlay (Picture p1, Picture p2,
                            Picture p3, Picture p4) {
  return overlay(overlay(p1,p2), overlay(p3,p4));
}

public Picture rotations (Picture p) {
  return fourPics(clockwise270(p),p,clockwise180(p),clockwise90(p));
}
```

Part a) Suppose two primitives are created by the following statements

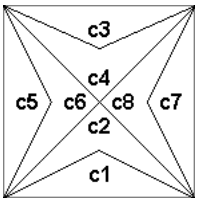
```
Picture a = triangle(Color.red);
Picture b = triangle(Color.black);
```

Evaluate the following expression. Draw the picture that it represents in the picture frame below. Note that the shaded lines inside the picture frame are not part of the picture, but are just there to help you draw your answer. Shade in the areas that are black and stripe the areas that are red.

```
fourOverlay(above(beside(empty(),beside(a,empty(),0.66),0.25),
                  empty()),
            flipHorizontally(above(beside(empty(),
                                          beside(b,empty(),0.66),0.25),
                                   empty())),
            above(empty(),beside(a,empty())),
            above(empty(),beside(empty(),a)))
```



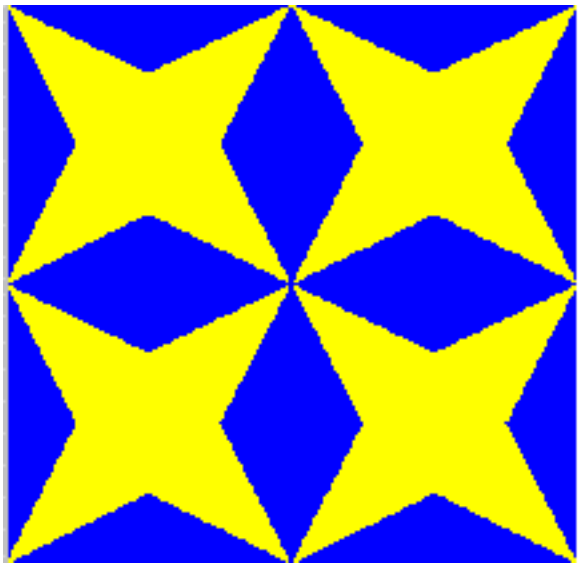
Part b) Using the methods with which you have been provided (i.e., triangle, fourPics, fourOverlay, rotations, and the PictureWorld contract), write a method to draw the star picture. A star picture can have up to eight colors indicated below. To assist you, we have provided you with the skeleton for your method. You must fill in the body of the method, where indicated. If you wish to define additional methods, place them on the next page.



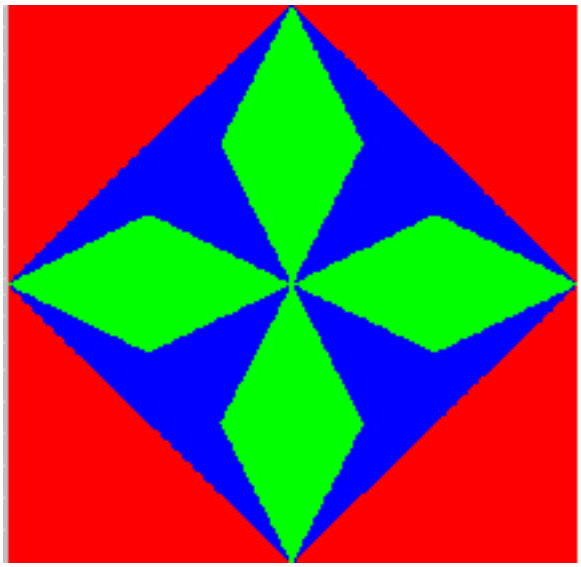

```
public Picture star (Color c1, Color c2, Color c3, Color c4,
                     Color c5, Color c6, Color c7, Color c8) {
  // fill in the body of the method below




}
```

```
// define additional methods here
```

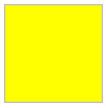
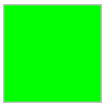
Part c) Write methods that return the two pictures below. You have been provided with skeletons for the two methods. You should use the star method from Part b in your expression. You should assume that the star method from Part b works correctly. This means that even if you were unable to answer Part b, or you answered it incorrectly, you can still answer this problem.



pattern1



pattern2

 Color.blue  Color.yellow  Color.green  Color.red

```
public Picture pattern1 () {
  // fill in the body of the method below
```

```
}
```

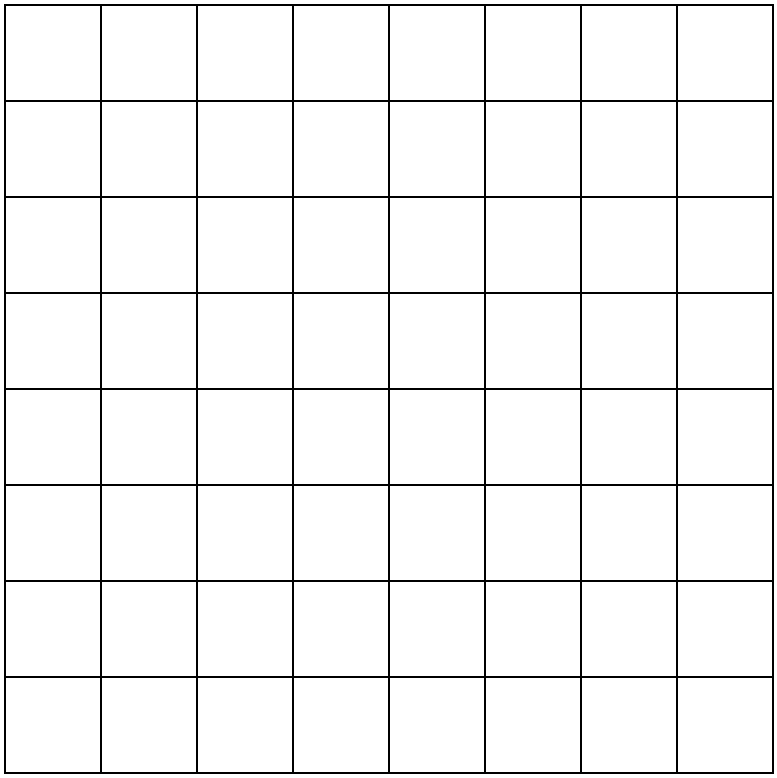
```
public Picture pattern2 () {
  // fill in the body of the method below




}
```

Problem 3

Below are the declarations of two classes: `TBuggleWorld` class that is a subclass of `BuggleWorld` and a `TBuggle` class that is a subclass of `Buggle`.

```
class TBuggleWorld extends BuggleWorld {
  public void run() {
    TBuggle tara = new TBuggle();
    tara.t(new Point(4,7), Direction.EAST, Color.yellow, 5);
    new TBuggle().t(new Point(3,5), Direction.NORTH, Color.blue, 3);
  }
}

class TBuggle extends Buggle {
  public void t(Point p, Direction d, Color c, int size) {
    this.setPosition(p);
    this.setHeading(d);
    this.setColor(c);
    this.forward((size+1)/2);
    this.backward(size);
    this.forward((size-1)/2);
    this.right();
    this.forward(size);
  }
}
```

Part a) A grid is provided below, in which you should sketch the pattern that is drawn by the TBuggles. In your sketch, you should indicate that a square has been colored a particular color by the using the capital letter corresponding to the first letter of the color (ie R=red, G=green, etc). You should also indicate the final position and direction of each TBuggle in your diagram. To do so, represent each TBuggle by a triangle and place the triangle in the grid cell of the TBuggle's final position. Also, the triangle should point in the direction (North, South, East or West) which the TBuggle ends up heading. Do not worry about indicating the color of the TBuggle when it is in its final position.



Part a) Suppose that Object Land contains an instance of the TBuggleWorld class that has the reference label TBW1. Use the skeleton of the Java Execution Model provided to draw the execution diagram for the execution of the statement

```
TBW1.run();
```

Pay attention to the following notes:

- You should follow the established conventions for drawing execution diagrams.
- Your Object Land should show three objects: the given instance of the `TBuggleWorld` and two instances of the `TBuggle` class. Label your TBuggles TB1 and TB2. You need not show any `Point`, `Color` or `Direction` instances in Object Land. For Points you can represent them instead by a circle around the coordinates within parentheses, as follows:

  (1,2)

- Your diagram should show the state of each object after the completion of the execution of the `run` method.
- The next page provides a skeleton to save you time in drawing your diagram. You must fill in the skeleton whre appropriate.
- Please try to make your diagram as neat as possible.

The code is duplicated below for your convenience.

```
class TBuggleWorld extends BuggleWorld {
  public void run() {
    TBuggle tara = new TBuggle();
    tara.t(new Point(4,7), Direction.EAST, Color.yellow, 5);
    new TBuggle().t(new Point(3,5), Direction.NORTH, Color.blue, 3);
  }
}

class TBuggle extends Buggle {
  public void t(Point p, Direction d, Color c, int size) {
    this.setPosition(p);
    this.setHeading(d);
    this.setColor(c);
    this.forward((size+1)/2);
    this.backward(size);
    this.forward((size-1)/2);
    this.right();
    this.forward(size);
  }
}
```

Execution Land