MORE CS111 PRACTICE PROBLEMS FOR EXAM 1

Problem 1: Booleans and Conditionals

a. (Circle True or False) `true` and `false` denote objects in Java. Briefly justify your answer.

b. Betty Bother has written the following method (in a rather unfortunate programming style) which does not compile properly:

```
public boolean isColdAndHeadingNorth () {
    if (getColor().equals(Color.blue) &&
        getHeading().equals(Direction.NORTH)) {
        return true;
    } else if (!getColor().equals(Color.blue) ||
               !getHeading().equals(Direction.NORTH)) {
        return false;
    }
}
```

(1) Explain the compiler error and (2) rewrite Betty's method in a much simpler style.

c. Which of the following program fragments are equivalent to:

```
if (A && B) {
    return bluePatch;
} else {
    return redPatch;
}
```

1. ```
   if (A && B) {
       return bluePatch;
   }
   return redPatch;
   ```
2. ```
   if (!A) {
       return redPatch;
   }
   if (B) {
       return bluePatch;
   }
   return redPatch;
   ```
3. ```
   if (!A || !B) {
       return  redPatch;
   }
   return bluePatch;
   ```

d. Define a `Buggle` method named `isBoxedIn()` that has no parameters and returns `true` if a buggle is in a cell surrounded by walls on all four sides, and otherwise returns `false`. The final state of the buggle when `isBoxedIn()` returns should be the same as the state of the buggle when `isBoxedIn()` is invoked. You may not use recursion or iteration in your solution, but you may define auxiliary methods if you want
   ```

Problem 2: Invocation Trees

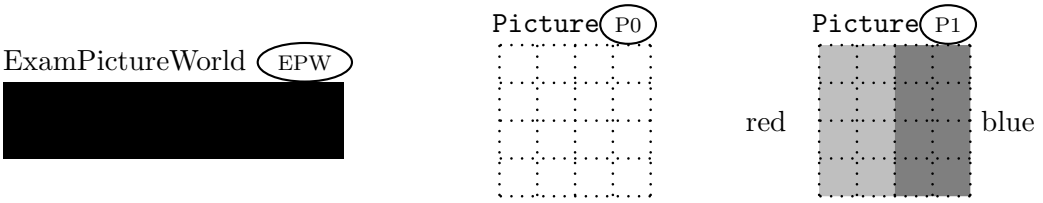
```
public class ExamPictureWorld extends PictureWorld {

    public Picture meth1 (Picture a) {
        Picture b = beside(a, empty());
        return overlay(meth2(b), b);
    }

    public Picture meth2 (Picture c) {
        return clockwise90(above(c, empty(), 0.75));
    }
}
```

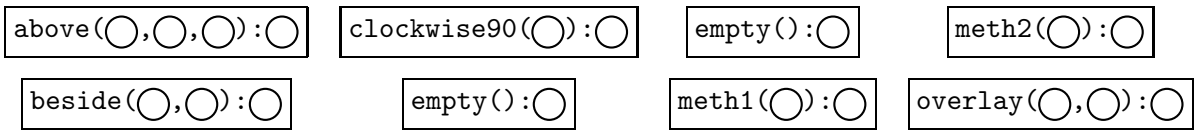
Figure 1: A sublcass of `PictureWorld`.

Consider the subclass of `PictureWorld` shown in Fig. 1. Supose that: (EPW) is an instance of `ExamPictureWorld`, (P0) is a `Picture` instance denoting the empty picture, (P1) is a `Picture` instance denoting the rightmost picture below:



The dashed grid lines are *not* part of the pictures. They indicate coordinates within pictures. The colors names are *not* part of picture (P1). They indicate the color of the two rectangles. Each of the two rectangles is a solid color *without* any separately colored border.

On the next page, you are to draw an invocation tree that models the instance method invocation (EPW).`meth1`((P1)). In the area labeled **Execution Land**, you should draw an invocation tree that contains the following eight nodes, arranged appropriately into a tree. You should use each node exactly once.

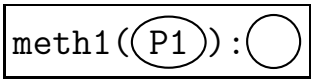


The empty circles in the nodes are skeletons for object references that you should fill in with one of the labels P0, P1, P2, P3, P4, or P5 to refer to the appropriate Picture instance in Object Land (see below). A circle enclosed by parentheses is a reference to an actual argument of the method invocation. A circle appearing after a colon is a reference to the result of the method invocation. The root of the invocation tree is the `meth1()` node, which has already been drawn for you, and whose actual argument has been filled in (you need to fill in its result).

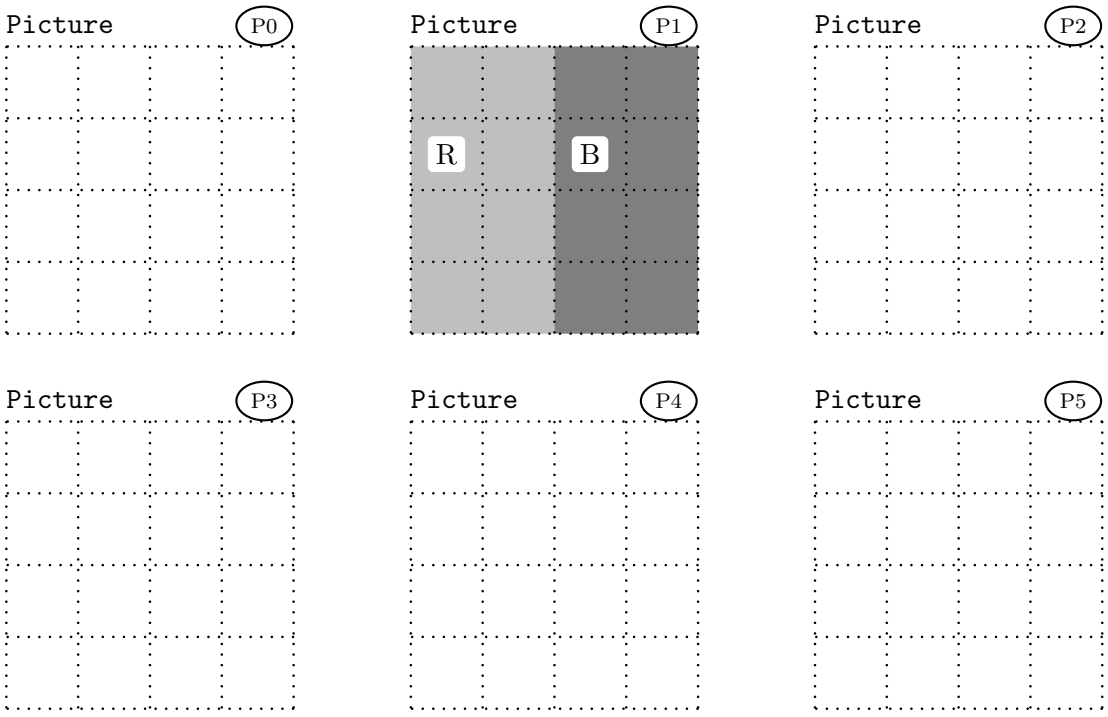
In the area labeled **Object Land** are the skeletons for the six `Picture` instances that are used during the execution. The pictures labeled (P0) and (P1) have already been drawn for you; you should draw the pictures for (P2), (P3), (P4), and (P5). In each picture, you should label red areas with the letter `R` and blue areas with the letter `B`. All other areas are presumed to be white.

(Note: for simplicity, the receiver object (EPW) for each of the method invocations has been omitted. This instance has also been omitted from Object Land.)

Execution Land



Object Land



Problem 3: Java Execution Model

Consider the following two class definitions:

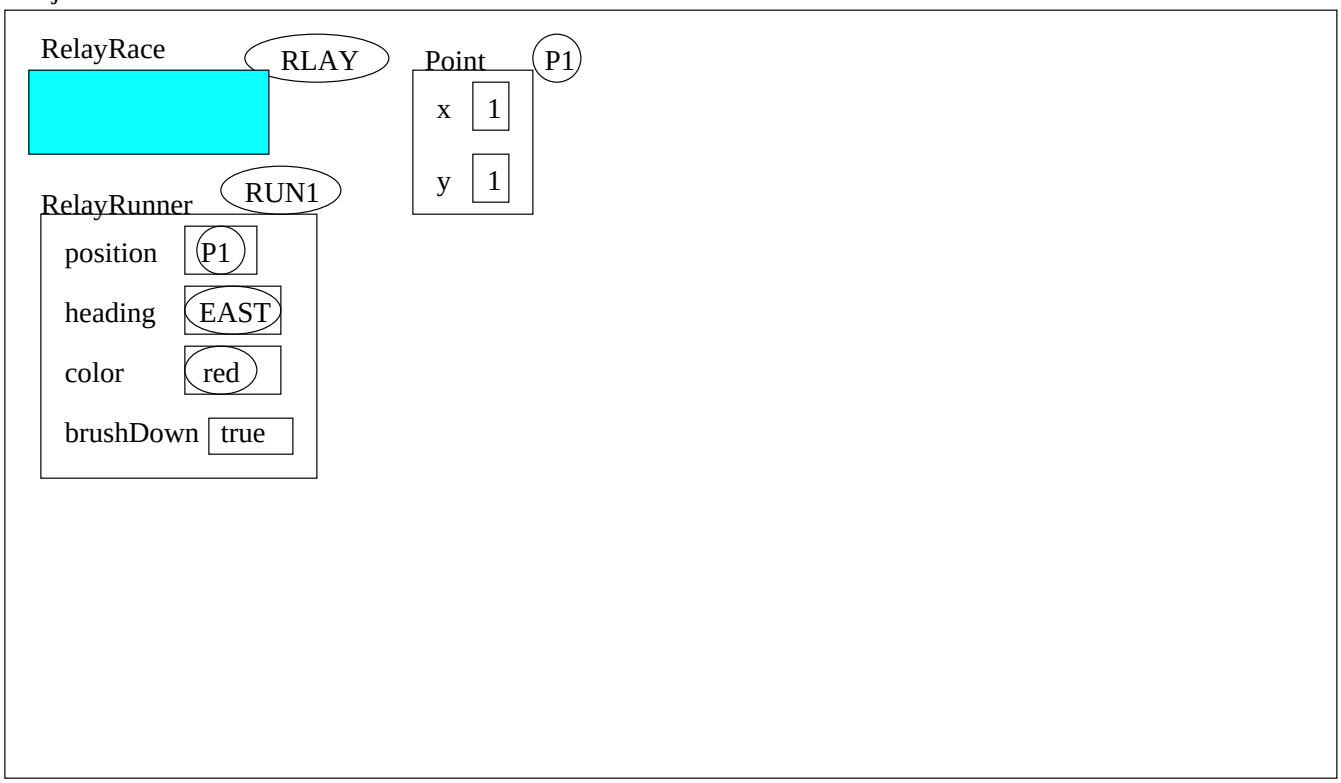
```
public class RelayRace extends BuggleWorld
{
    public void run()
    {
        RelayRunner r1 = new RelayRunner();
        RelayRunner r2 = new RelayRunner();
        RelayRunner r3 = new RelayRunner();

        r2.setColor(Color.green);
        r3.setColor(Color.blue);
        r1.firstLeg(3, r2,r3);
    }
}

class RelayRunner extends Buggle
{
    public void firstLeg(int length, RelayRunner next, RelayRunner last)
    {
        forward(length);
        next.setPosition(this.getPosition());
        next.secondLeg(length, last);
    }
    public void secondLeg(int length, RelayRunner next)
    {
        forward(length);
        next.setPosition(this.getPosition());
        next.thirdLeg(length);
    }
    public void thirdLeg(int length)
    {
        forward(length);
    }
}
```

The Java Execution Model diagram below shows the state of the program after evaluating the first line of the `run()` method. Show the diagram after the completion of the `run()` method. Include in Object Land all instances of the `RelayRunner` and `Point` classes referenced by local or instance variables. Also include all execution frames opened during the execution of the `run()` method. You may abbreviate references to instances of the `Direction` and `Color` classes as ovals surrounding appropriate identifying information (as shown below).

Object Land



Execution Land