Sample questions for CS111 Midterm Exam 1.

Problem 1: Buggle World Execution

Consider the two Java classes in Fig. 1.

```
public class DoItWorld extends BuggleWorld {

  public void run () {
    DoItBuggle dewey = new DoItBuggle();   // run statement 1
    int n = 5;                             // run statement 2
    dewey.setPosition(new Point(n,n-2));   // run statement 3 *
    dewey.brushUp();                       // run statement 4
    dewey.doit(Color.green, n-1);          // run statement 5 *
    dewey.doit(Color.blue, n+1);           // run statement 6 *
    dewey.forward();                       // run statement 7
    dewey.brushDown();                     // run statement 8
    dewey.forward(3);                      // run statement 9 *
  }
}

class DoItBuggle extends Buggle {

  public void doit (Color c, int n) {
    Color oldColor = this.getColor();
    this.setColor(c);
    this.forward(n);
    this.brushDown();
    this.backward(n-2);
    this.brushUp();
    this.backward(2);
    this.left();
    this.setColor(oldColor);
  }

}
```

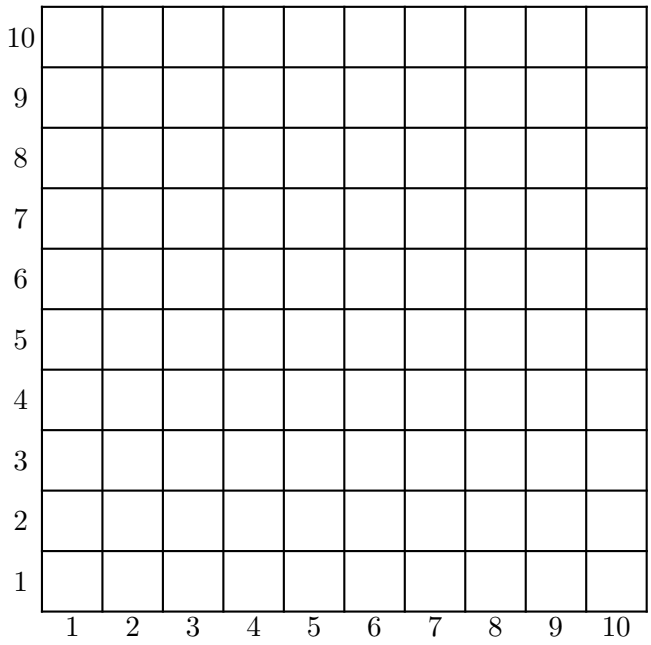
Figure 1: Two Java classes.

Suppose that the `run()` method is invoked on an instance of `DoItWorld` which has a 10×10 grid of cells. In the four grids on the following page, show the state of the grid directly *after* the execution of each of the statements in the `run()` method body marked with a `*`.

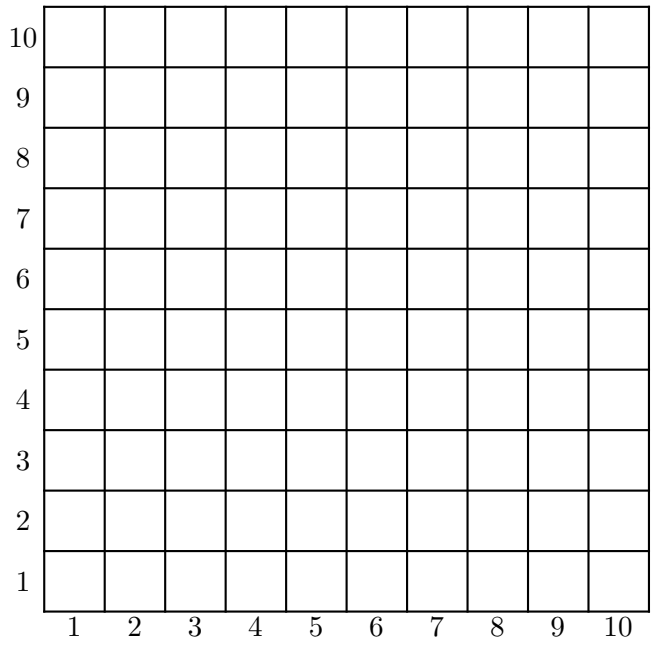
In each grid, you should show the following:

1. Draw buggle `dewey` as a triangle "pointing" in the direction that the buggle is facing.
2. Indicate the current color of the buggle by putting the *first letter* of the color name inside the triangle (e.g. `B` for blue, `G` for green, etc.).
3. Indicate the color of each non-white grid cell by putting the *first letter* of the color name inside the cell (e.g. `B` for blue, `G` for green, etc.).

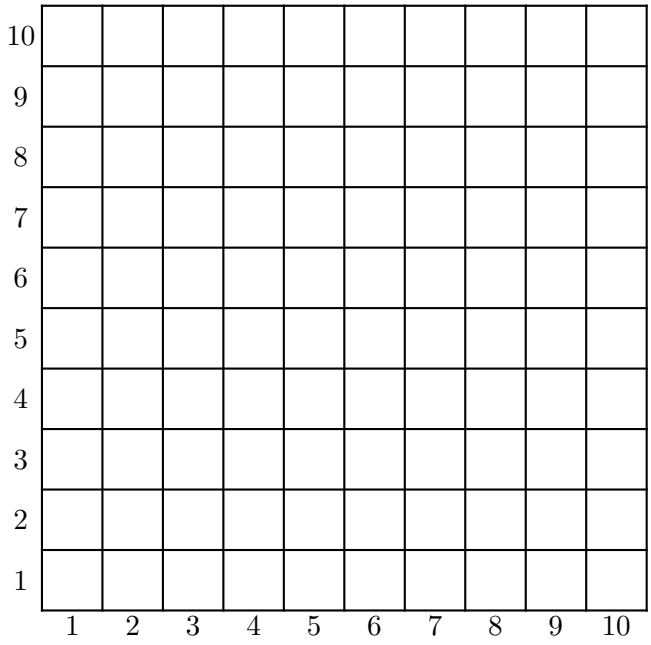
`DoItWorld` grid after the execution of `run()` statement 3



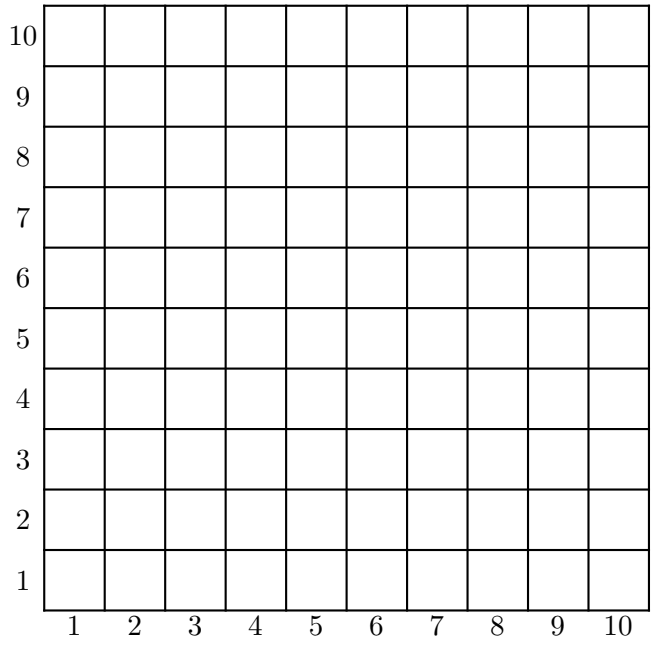
`DoItWorld` grid after the execution of `run()` statement 5



`DoItWorld` grid after the execution of `run()` statement 6

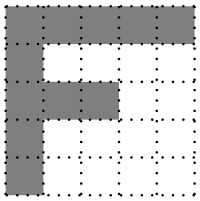


`DoItWorld` grid after the execution of `run()` statement 9



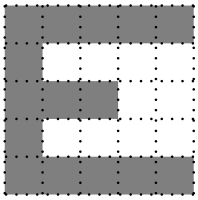
Problem 2: Writing Methods

Suppose that `LetterWorld` is a subclass of `PictureWorld` that supplies you with a method named `f` with the following contract:

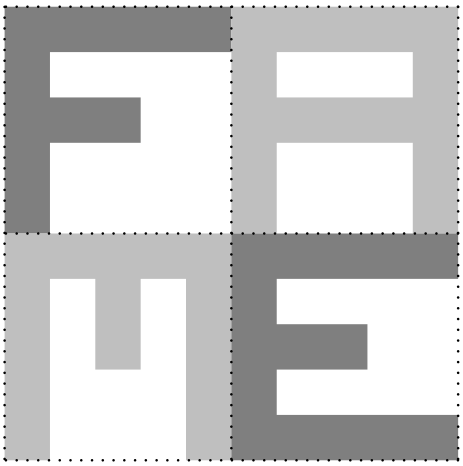
> `public Picture f (Color c)`
> Returns a picture of the letter "F" in color `c`, as shown below.
>
> 
>
> The dotted lines indicate the boundaries of the unit square, and are *not* part of the picture. The letter is a solid color `c` and does *not* have any boundary line drawn in a separate color.

On the next page your task is to write *two* methods:

1. A method named `e` that takes a single color parameter and returns the following picture of the letter "E" in that color.

2. A method named `fame` that takes two color parameters and returns the following picture:

   The "F" and "E" have the color of the first parameter, while the "A" and "M" have the color of the second parameter.

You may assume that both methods are defined within the `LetterWorld` class, and so may use the `f` method in addition to the methods in the `PictureWorld` contract (e.g., `clockwise90`, `flipDiagonally`, `above`, etc.). You may assume that the `fourPics` and `fourSame` methods defined in class and on the problem sets are also available. Your `fame` method may use your `e` method, which you may assume works correctly (even if your definition of `e` is actually incorrect or missing).

Put your definition of the `e` *method here.*

Put your definition of the `fame` *method here.*

Problem 3: Debugging

The class declarations in Fig. 2 contain (at least) **10 errors** (syntax errors and type errors).

```
public class ExamBuggleWorld extends BuggleWorld {  // line 1
                                                     // line 2
    public void run () {                             // line 3
        Color c = Color.cyan();                      // line 4
        int n = 4                                    // line 5
        ExamBuggle emma = ExamBuggle();              // line 6
        emma.mystery1(c,n);                          // line 7
        emma.mystery1(3,Color.red);                  // line 8
        boolean answer = emma.mystery2();            // line 9
        this.mystery3();                             // line 10
    }                                                // line 11
}                                                    // line 12
                                                     // line 13
class ExamBuggle extends Buggle {                    // line 14
                                                     // line 15
    public void mystery1(Color c, int n1) {          // line 16
        n2 = n1 + 1;                                 // line 17
        this.setColor(Color.c);                      // line 18
        forward(n2);                                 // line 19
        this.dropBagel();                            // line 20
                                                     // line 21
    public boolean mystery2() {                      // line 22
        this.isOverBagel();                          // line 23
    }                                                // line 24
                                                     // line 25
    public mystery3() {                              // line 26
        this.dropBagel();                            // line 27
    }                                                // line 28
                                                     // line 29
}                                                    // line 30
```

Figure 2:

In the table on the next page, for each of 10 errors in different lines of the above program give:

1. the line number of the error,
2. a *brief* description of the error, and
3. a corrected version of the line (i.e., with the error fixed).

You may list the errors in *any* order. You do *not* have to list them in the order in which they occur in the program.

Error #	Line #	Brief description of error	Corrected line
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			