YOUR NAME: ____________________________

This exam has five problems. Each problem has several parts. The number of points for each problem is shown in square brackets next to the problem. There are 100 total points on the exam.

Write all your answers on the exam itself. Whenever possible, show your work so that partial credit can be awarded.

The exam is open book. You may refer to your notes, handouts from this semester (including the Fall’01 CS111 text), problem sets and solutions, and whatever additional materials would be helpful. However, you may not use another person’s notes, or any materials from previous semesters (excluding the Fall’01 CS111 text).

By the Honor Code, you are not allowed to talk to anyone about the details of this exam before or after taking it, until after all members of the class have taken the exam.

Please keep in mind the following tips:

- Briefly skim the entire exam before starting any problem.
- Work first on the problems on which you feel most confident. You do not have to do the problems in order. All problems are independent from each other.
- Try to do something on every problem so that you can potentially receive partial credit. A guess is better than no answer at all!
- Allocate your time carefully. If you are taking too long on a problem, wrap it up and move on.
- If you finish early, go back and check your answers.

GOOD SKILL!

<table>
<thead>
<tr>
<th>Problem</th>
<th>Topic</th>
<th>Points</th>
<th>Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Java Syntax</td>
<td>15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>BuggleWorld Execution</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Writing Methods</td>
<td>25</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Invocation Trees</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>Debugging</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td></td>
<td>100</td>
<td></td>
</tr>
</tbody>
</table>
This page intentionally left blank
(except for this self-referential sentence).
**Problem 1 [15]: Java Syntax**
For each of the following Java code fragments, indicate:

1. whether the code fragment is an expression or a statement;
2. the kind of expression or statement;
3. if the code fragment is an expression, indicate the type of value to which it evaluates. (You need not indicate anything for a statement).

In the code fragments, assume that `bob` is an instance of the `Buggle` class and that `StringChooser` is the class studied in Lab 1. As examples, parts a and b are done for you.

<table>
<thead>
<tr>
<th>Part</th>
<th>Code Fragment</th>
<th>Expression or Statement</th>
<th>Kind of Expression or Statement</th>
<th>Type (for expressions only)</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>new Buggle()</td>
<td>expression</td>
<td>constructor method invocation</td>
<td>Buggle</td>
</tr>
<tr>
<td>a</td>
<td>String hi = &quot;Hello&quot;;</td>
<td>statement</td>
<td>local variable declaration</td>
<td>N/A (not applicable)</td>
</tr>
<tr>
<td>c</td>
<td>bob.getPosition()</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>bob.setColor(Color.blue);</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>e</td>
<td>bob</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>f</td>
<td>Direction.EAST</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>g</td>
<td>StringChooser.choseLine(&quot;nouns.txt&quot;)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Problem 2 [20]: Buggle World Execution

Consider the two Java classes in Fig. 1.

```java
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 *
    }
}

public 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 $10 \times 10$ 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 \texttt{run()} statement 3

DoItWorld grid after the execution of \texttt{run()} statement 5

DoItWorld grid after the execution of \texttt{run()} statement 6

DoItWorld grid after the execution of \texttt{run()} statement 9
Problem 3 [25]: 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.

```
F
```

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.

```
E
```

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

```
F
A
E
M
```

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, fourSame, and fourOverlay 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 \texttt{e} method here.

Put your definition of the \texttt{fame} method here.
Problem 4 [20]: Invocation Trees

```java
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 2: A subclass of PictureWorld.

Consider the subclass of PictureWorld shown in Fig. 2. Suppose 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.)
meth1(P1): ○
Problem 5 [20]: Debugging

The class declarations in Fig. 3 contain (at least) **10 errors** (syntax errors and type errors).

```java
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

class ExamBuggle extends Buggle { // line 13
    // line 14
    public void mystery1(Color c, int n1) { // line 15
        n2 = n1 + 1; // line 16
        this.setColor(Color.c); // line 17
        forward(n2); // line 18
        this.dropBagel(); // line 19
    } // line 20

    public boolean mystery2() { // line 21
        this.isOverBagel(); // line 22
    } // line 23

    public mystery3() { // line 24
        this.dropBagel(); // line 25
    } // line 26
} // line 27
```

Figure 3:

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.
<table>
<thead>
<tr>
<th>Error #</th>
<th>Line #</th>
<th>Brief description of error</th>
<th>Corrected line</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**THIS IS THE END OF THE EXAM.**