

Exam I

Given: 24 Oct 2000

Due: end of exam period

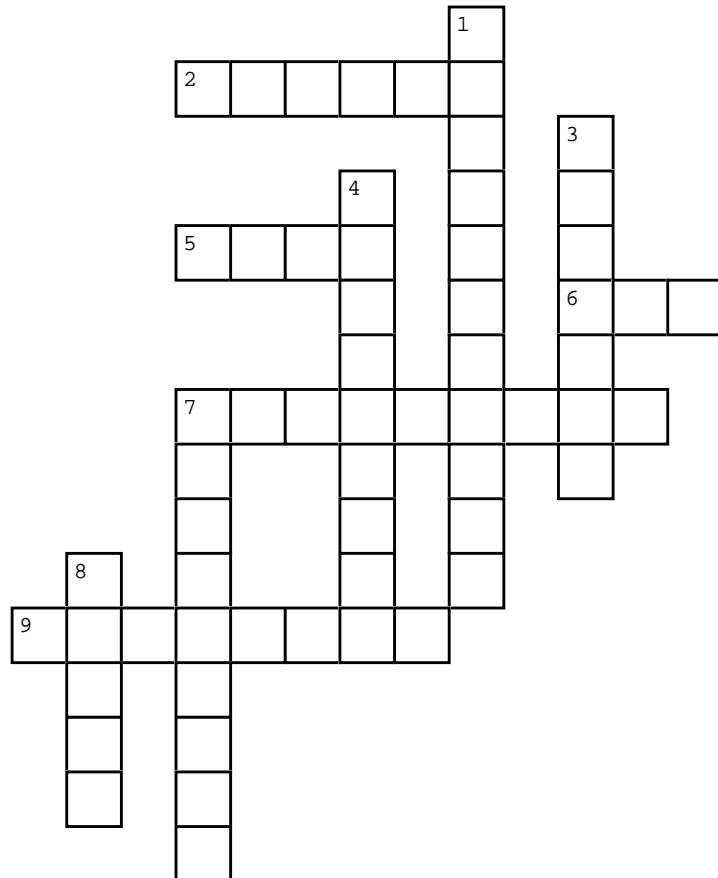
Name:

Lab Section:

- This exam is open book and open notes.
- Please check that you have all pages, numbered 1 through 14. Write your name on each piece of paper, in case the pages become separated.
- Write your answers concisely and legibly *directly on this exam*. Do not use extra sheets of paper.
- Do your own work. No discussion or collaboration with other students is permitted.
- If a question is unclear to you, please raise your hand and somebody will come to help you.
- The exam is divided into parts as described below. By each section heading, an estimate of the time required to complete that section is provided to help you pace yourself. If you get stuck on a question, don't waste too much time on it. Go on to the next question and the answer may occur to you later.
- Partial credit will be given where appropriate. If you see how to approach a problem but don't see the final answer, be sure to at least write down your approach.

Section	Score	Possible
1. Definitions		10
2. Expressions		10
3. Methods		25
4. Objects		30
5. Recursion and Iteration		25
TOTAL		100

1. Definitions (5 minutes – 10 points)

**ACROSS**

- 2) A primitive type that can represent fractional values
- 5) _____case -- a predicate that, when true, ends a recursive algorithm
- 6) Provides the names of method, their return types and parameters
- 7) a nonrecursive way to perform repetition
- 9) a method that returns a descriptive String for a class

DOWN

- 1) _____ condition: the complement of a while loop's predicate
- 3) Such methods alter the values of instance variables
- 4) a form of reduction applied at the method level
- 7) The instance variables of such objects never change
- 8) _____ variable: these do not exist after a method returns

2. Expressions (5 minutes – 10 points)

- (a) (4 points) For each expression below, draw the tree that represents the expression's computation and state the expression's resulting type (or error, if the expression would cause an error). Assume that any variables present in an expression are of type `int`.

i. $4 - 3 + 1$

Draw the tree below

Return type:

Resulting value:

ii. $4 + 3 * 5 + \text{" is not " } + 35$

Draw the tree below

Return type:

Resulting value:

iii. `101 + " is " + true`
Draw the tree below

Return type:

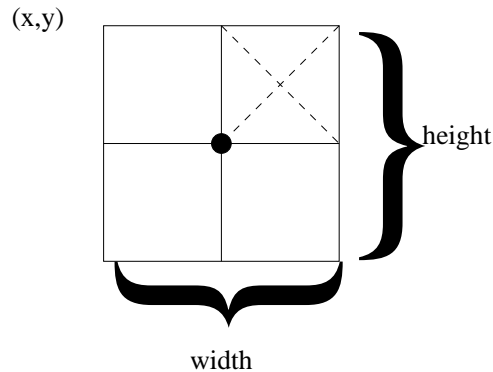
Resulting value:

iv. `false || 4 < 2`
Draw the tree below

Return type:

Resulting value:

(b) (4 points)



The rectangle above is the object `rect`, characterized by its upper-left corner, width, and height, as is the custom with `canvas` rectangles:

```
int x = rect.getShapeX()
int y = rect.getShapeY()
int width = rect.getShapeWidth()
int height = rect.getShapeHeight()
```

Consider the method

```
private void foo(int x, int y, int width, int height)
```

that accepts the above characterization of a rectangle.

Subrectangles are formed by bisecting both dimensions of the rectangle. Below, write the code to call `foo` on the subrectangle denoted with the dashed-X in it:

(c) (2 points) In Lab 4, you used randomness to change the angle between branches. Below, write the Java expression that computes a random integer `rand` in the range $-45 \leq \text{rand} \leq 45$.

3. Methods (10 minutes – 25 points)

- (a) (10 points) In Lab 5, if `rigor(x,y)` computes the same values for all points (x,y) along the perimeter of a suitable rectangle, then all pixels within that rectangle will obtain the same value from `rigor`. This implied that they could all be set to the same color by adding a filled `Rect` to the `DrawingPane` instead of setting each pixel one-by-one.

Below is a skech of the method `horizExplore` that takes in

- `x`, the starting x-coordinate. Recursive calls will use $(x+1)$ in this argument position.
- `y`, the fixed y-coordinate: this doesn't change in the recursion.
- `endX`, the last x-value for exploration.
- `value`, the value to be matched from `rigor` for all points in the recursively touched space.

```
private boolean horizexplore(int x, int y, int endX, int value) {
    if (
        ) // base case, fill in

    return true; // all match, so return true

    if (rigor(x,y) != value) // mismatch here, no need to look further
        return false;

    else { // fill this in

    }
}
```

- (5 points) Fill in the predicate for the `base case` part of the method.
- (5 points) The recursive call is missing; fill it in.

- (b) (15 points) For $a \geq 0$ and $b > 0$, the computation of $a \% b$ can be expressed by the following recursive method.

```
public int mod(int a, int b) {  
    if (a < b) return a;  
    else return mod(a-b, b);  
}
```

- i. (5 points) Show the evaluation of $\text{mod}(7, 2)$ using the substitution model.

- ii. (5 points) Show the evaluation of $\text{mod}(5, 3)$ using the data flow model.

iii. (2 points) What is the base case for this method?

iv. (3 points) Using the substitution model, show what happens if the method is called with $b = 0$.

4. Objects (15 minutes – 30 points)

Rip this page out if you want to—you don't answer anything directly on this page.¹

```
import canvas.*;

public class Team {

    private String name;
    private String owner;
    private int score;

    public Team(String name, String owner) {
        this.name = name;
        this.owner = owner;
        score = 0;
    }

    private void bumpScore(int howMuch) {
        score = score + howMuch;
    }

    public void touchDown(boolean extrapt) {
        bumpScore(6);
        if (extrapt) bumpScore(1);
    }

    public void fieldGoal() {
        bumpScore(3);
    }

    public int getScore() {
        return score;
    }

    public String toString() {
        return name + "(" + owner + ")" + " score is " + score;
    }

}
```

¹Plus, ripping it out may make you feel better

```
public static void testTeam() {
    Team a = new Team("A", "Owner A");
    Team b = new Team("B", "Owner B");
    Team c = new Team("C", "Owner C");

    Team x = b;

    a.touchDown(true);
    b.touchDown(false);
    c.fieldGoal();

    int mystery1 = a.getScore();
    int mystery2 = b.getScore();
    int mystery3 = c.getScore();

    x.fieldGoal();

    int mystery4 = a.getScore();
    int mystery5 = b.getScore();
    int mystery6 = c.getScore();
    int mystery7 = x.getScore();

    Transcript.println(a);
    Transcript.println(b);
    Transcript.println(c);
}
```

Answer the following questions about `Team`.

- (a) (3 points) Which, if any, of the instance variables in `Team` are immutable?
- (b) (6 points) What is the strongest assertion you can make about the instance variable `score`?

Answer the following questions about `testTeam()`.

- (a) (2 points) What is the value assigned to `mystery1`?_____
- (b) (2 points) What is the value assigned to `mystery2`?_____
- (c) (2 points) What is the value assigned to `mystery3`?_____
- (d) (2 points) What is the value assigned to `mystery4`?_____
- (e) (2 points) What is the value assigned to `mystery5`?_____
- (f) (2 points) What is the value assigned to `mystery6`?_____
- (g) (3 points) What is printed to `Transcript`?

(h) (3 points) How many reference variables are declared in `testTeam()`?

(i) (3 points) How many objects are instantiated during execution of `testTeam()`?

5. Recursion and Iteration (20 minutes – 25 points)

Consider the following definition.

$$f(n) = \begin{cases} 1 & \text{if } n == 1 \\ 2 & \text{if } n == 2 \\ 3 & \text{if } n == 3 \\ f(n-1) \times f(n-2) & \text{if } n > 3 \\ \text{error} & \text{otherwise} \end{cases}$$

Here are some examples:

$$\begin{array}{ll} n & f(n) \\ 3 & 3 \\ 5 & f(4) \times f(3) = (f(3) \times f(2)) \times 3 = 18 \\ 7 & f(6) \times f(5) = (f(5) \times f(4)) \times (f(4) \times f(3)) = (18 \times 6) \times (6 \times 3) = 1944 \end{array}$$

- (a) (8 points) Below, write a *recursive* method `foo` that takes as input an integer n , and then returns $f(n)$ as defined above. Do *not* use a loop! Be sure to throw an error when given erroneous input.

(b) (1 points) What is the base case for your method?

(c) (3 points) Using data flow or substitution, show how your method evaluates `foo(5)`.

- (d) (8 points) Complete the three assignment-statements in the method shown below so that $f(n)$ is computed iteratively as `foo(n)`.

```
public static int foo(int n) {  
  
    if (n < 1) throw new Error("n < 0 -- bad, bad, bad");  
  
    if (1 <= n && n <= 3) return n;  
  
    int oneBefore = 3;  
    int twoBefore = 2;  
    int ans = oneBefore * twoBefore;  
  
    int i = 4;  
  
    while ( i < n ) {  
        i = i + 1;  
  
        twoBefore =  
  
        oneBefore =  
  
        ans =  
    }  
    return ans;  
}
```

- (e) (2 points) What value must i have when the loop exits?

- (f) (3 points) Which of the following is a (are some) loop invariant(s) for the `while` loop?

- $ans \equiv f(i)$
- $i \equiv i + 1$
- $i \geq 4$