

Name: _____

CSSE 220—Object-Oriented Software Development

Exam 1, Jan. 4, 2011

This exam consists of two parts. Part 1 is to be solved on these pages. If you need more space, please ask your instructor for blank paper. Part 2 is to be solved using your computer. You will need network access to download template code and upload your solution for part 2. *Please disable IM, email, and other such communication programs before beginning the exam. Any communication with anyone other than the instructor or a TA during the exam, or about the exam with students in the other section, may result in a failing grade for the course.*

Allowed Resources: Open book, notes, and computer. Limited network access. You may use the network only to access your own files, the course ANGEL site and web pages, the textbook's site, Sun's Java website, and Logan Library's Safari Tech Books Online.

Parts 1 and 2 are included in this document.

Part 1 is to be completed on paper. You may use the resources listed, but *you must not enter any code for part 1 on your computer.*

Problem	Poss. Pts.	Earned
1	12	_____
2	3	_____
3	6	_____
4	6	_____
5	3	_____
6	6	_____
7	6	_____
8	8	_____
Paper Part Subtotal	50	_____
Computer Part Subtotal	50	_____
Total	100	_____

Part 1—Paper Part

The next several questions all refer to an `AngryBirdsGame` class. Below is a listing of this class showing its fields, constructors, accessor methods, and mutator methods. The javadocs are omitted to save space. DO NOT TYPE THIS CLASS IN ECLIPSE.

```
public class AngryBirdsGame {
    private int stage;
    private int score;
    private int level;
    public static final int STAGES_PER_LEVEL = 22;

    public AngryBirdsGame(int stage, int score, int level) {
        this.stage = stage;
        this.score = score;
        this.level = level;
    }

    public AngryBirdsGame() {
        this(1, 0, 1);
    }

    public void nextStage() {
        this.stage++;
        if (this.stage > AngryBirdsGame.STAGES_PER_LEVEL) {
            this.level++;
            this.stage = 1;
        }
    }

    public void addScore(int score) {
        this.score += score;
    }

    public int getScore() {
        return this.score;
    }

    public int getLevel() {
        return this.level;
    }

    public int getStage() {
        return this.stage;
    }
}
```

1. (12 points) Below are several code snippets, many of which use the AngryBirdsGame class. For each snippet, first *draw a box-and-pointer diagram* showing the result of executing it. Then *give the output* of the print statement at the end of the snippet.

```
AngryBirdsGame g =  
    new AngryBirdsGame(22, 4980, 1);  
g.nextStage();  
System.out.println("Level: " + g.getLevel());
```

(a) Output: _____
Diagram:

```
AngryBirdsGame game1 =  
    new AngryBirdsGame(1, 50, 1);  
AngryBirdsGame game2 = game1;  
game2.addScore(3850);  
System.out.println("Score: " + game1.getScore());
```

(b) Output: _____
Diagram:

```
AngryBirdsGame[] games = new AngryBirdsGame[2];  
games[0] = new AngryBirdsGame(15, 15000, 2);  
games[1] = new AngryBirdsGame();  
AngryBirdsGame ag = games[games.length - 1];  
System.out.println("Level: " + ag.getLevel());
```

(c) Output: _____
Diagram:

2. (3 points) In question 1(c) above, we used an array to hold our AngryBirdsGames. For an avid player who enjoys playing Angry Birds, such a data structure would not be appropriate. Explain why an ArrayList would be a better fit.

3. (6 points) For each code snippet below predict its output. Some refer to the AngryBirdsGame class. (You do *not* need to draw a diagram, but you may if it might help you.)

```
AngryBirdsGame g =  
    new AngryBirdsGame(1, 200019, 2);  
double x = g.getScore() / g.getLevel();  
System.out.printf("x = %8.2f%n", x);
```

(a) Output: _____

```
AngryBirdsGame b =  
    new AngryBirdsGame(1, 400001, 4);  
double y = b.getScore() / (double) b.getLevel();  
System.out.printf("y = %8.2f%n", y);
```

(b) Output: _____

```
int m = 4;  
if (m % 3 == 0) {  
    System.out.println(m + m);  
} else {  
    System.out.println(m + "three");  
}
```

(c) Output: _____

4. (6 points) Write down the output of running each code snippet below. If enough information is not provided, indicate that we cannot tell. DO NOT TYPE THE CODE SNIPPETS FOR THIS QUESTION IN ECLIPSE.

```
int value = 1;
for (int i = 0; i < 5; i++) {
    value = value * 2;
}
System.out.println("value = " + value);
```

(a) Answer:

value = _____

```
int sum = 0;
int count = 0;
int n = 5;
for (int i = 1; i <= n; i += 2) {
    sum += i;
    count++;
}
System.out.println("count = " + count);
System.out.println("sum = " + sum);
```

(b) Answer:

count = _____

sum = _____

```
int x = 3;
int c = 0;
boolean inc = false;
while (4 > x) {
    if ((x > 0) && !(inc)) {
        x--;
    } else {
        x++;
        inc = true;
    }
    c++;
}
System.out.println("c = " + c);
```

(c) Answer:

c = _____

5. (3 points) Explain the difference in behavior between

```
AngryBirdsGame a =  
    new AngryBirdsGame(. . .);  
AngryBirdsGame b =  
    new AngryBirdsGame(. . .);  
if (a == b){  
    . . .  
}
```

and

```
AngryBirdsGame a =  
    new AngryBirdsGame(. . .);  
AngryBirdsGame b =  
    new AngryBirdsGame(. . .);  
if (a.equals(b)){  
    . . .  
}
```

6. (6 points) *Unit Testing*: Consider the documentation for the method below.

```
/**  
 * Converts from United States Dollar (USD) to Canadian Dollar (CAD).  
 * Assumes 1.00 USD = 1.02 CAD.  
 *  
 * For example, usdToCad(10.0) yields 10.2.  
 *  
 * @param usCurrency  
 * @return Canadian equivalent  
 */  
public static double usdToCad(double usCurrency) {  
    // body code elided  
}
```

List three pairs of arguments and return values that would constitute a good test set for this method. For each argument you list, say briefly why it should be in the unit test.

7. (6 points) For each of the code snippets below, first *circle* the bug, then *fix it* so that no more errors exist in the code.

(a) *Circle* the bug in the code below and *correct* what's wrong:

```
Scanner scanner = new(System.in);
```

(b) *Circle* the bug in the code below and *correct* what's wrong:

```
Ellipse2D.Double circle;  
  
double w = circle.getWidth();
```

(c) *Circle* the bug in the code below and *correct* what's wrong:

```
int NUM_OF_GAMES = 4;  
  
AngryBirdsGame[] games = new AngryBirdsGame[NUM_OF_GAMES];  
  
System.out.println(games[0].getScore());
```

8. (8 points) Objects and classes

(a) What keyword indicates that *a variable cannot change its value from its initial value*?

Answer: _____

(b) In the statement: `clock.setTime(12, 30, "PM");`

an *explicit parameter* is _____.

an *implicit parameter* is _____.

(c) In a method definition, what keyword refers to the *implicit parameter*?

Answer: _____

(d) Generally, what *visibility* should *fields* have?

Answer: _____

Part 2—Computer Part

Instructions. You must actually get this problem working on your computer. Almost all of the credit for the problem will be for code that actually works. There are several different small methods to write, so you can get a lot of partial credit by getting some of them to work. If you get every part working, comments are not required. If you do not get a method to work, comments may help me to understand enough so I can give you (a small amount of) partial credit.

Begin part 2 by checking out the project named *Exam1* from your course SVN repository. (Ask for help immediately if you are unable to do this.)

When you have finished the problem, and more frequently if you wish, **submit your code by committing it to your SVN repository.** We will check commit logs, so you must be careful not to commit anything after the end of the exam. For grading, we will ensure that the included JUnit tests have not been changed.

Problem Description

C1. (50 points) College students often build debt while they are in college. The kinds of debt they accumulate might include *Credit Card Debt*, a *Student Loan*, an *Auto Loan*, a *Personal Loan*, and, on rare occasions, a *Mortgage*. The provided Debt class is fully implemented and models a debt item that might appear on a student's credit report.

For this problem, you will complete an application that takes several Debt objects, calculates the total debt and the percent of each kind of debt. Your application should display a pie chart with a legend like in Figure 1 on page 10.

The provided DebtAnalysis class is already implemented. It has a main method that creates a list of Debts and calls a helper method to do the display. (In a non-exam situation, this main method would get the data from the user or a file. We hard-coded the data to simplify the problem.) The helper method creates a JFrame to display the results, constructs a new DebtComponent instance, and adds the instance to the frame.

Your task is to finish implementing the DebtComponent class so that it analyzes the list of Debts and draws the pie chart with legend.

The program comments contain numbered “TODO:” items. Complete them in the given order. JUnit tests for this problem are in DebtComponentTest.

Code that uses the DebtComponent class starts in DebtAnalysis. You only need to implement DebtComponent. Debt and DebtAnalysis are completed already.

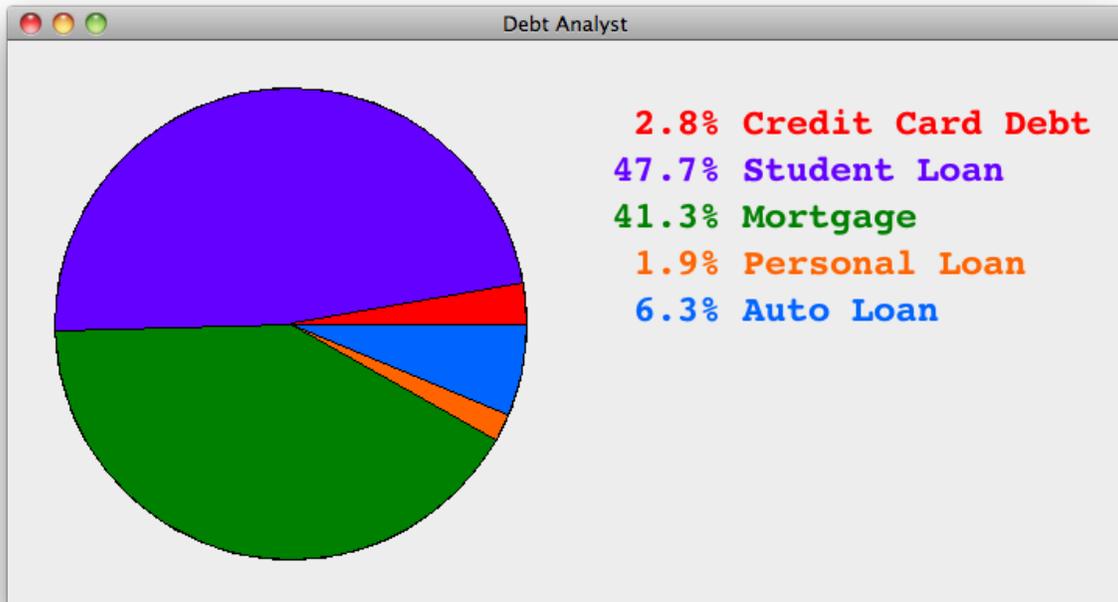


Figure 1: Example pie chart with legend

JUnit Test or Graphics	Points	Earned
analyzeDebts tests (2 each)	6	_____
findTotalOfKind tests (2 each)	12	_____
findPercentageOfKind tests (2 each)	12	_____
Graphics drawing		
Labels appear	3	_____
Correct label text	3	_____
Correct label formatting	3	_____
Circle in right location	3	_____
Pie slices of right size	3	_____
Pie slices in right locations	3	_____
Colors correct	2	_____
Computer Part Subtotal	50	_____