Name: ____________________________________

CSSE 220—Object-Oriented Software Development

Exam 2, Oct. 30, 2009

This exam consists of two parts. Part 1 is to be solved on these pages. Ask your instructor for scratch paper if you need more room. 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.

Resources for Part 1: You may use a single sheet of $8\frac{1}{2} \times 11$ inch paper with notes on both sides. Notes may be printed but must not be smaller than 4 point.

Resources for Part 2: 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. Any communication with anyone other than the instructor or a TA during the exam may result in a failing grade for the course.

Parts 1 and 2 are included in this document. You should read over all of the questions before beginning work, but…

You must turn in part 1 before accessing the resources for part 2.

Problem	**Poss. Pts.**	**Earned**
1	6	______
2	12	______
3	12	______
4	4	______
5	4	______
6	9	______
7	16	______
8	12	______
Paper Part Subtotal	**75**	______
C1. MyDotter dots()	5	______
C1. MyDotter leftClickAt()	5	______
C1. MyDotter rightClickAt()	5	______
C2. SierpinskiTriangle shadedArea() (2 per unit test)	10	______
Computer Part Subtotal	**25**	______
Total	100	______

Part 1—Paper Part

1. (6 points) Define *cohesion* and *coupling* and explain why the two properties are often in opposition.

2. (12 points) This problem is a design exercise. First read the problem description below, then answer the questions.

Design a program to track job applcations for all the applicants to Really Old Bank (ROB). An application to ROB lists the applicants name, gender, diplomas and degrees earned, prior jobs held, and position applied for. For each diploma, degree, or job held, the application lists the institution/company, city, state, and dates attended/employed.

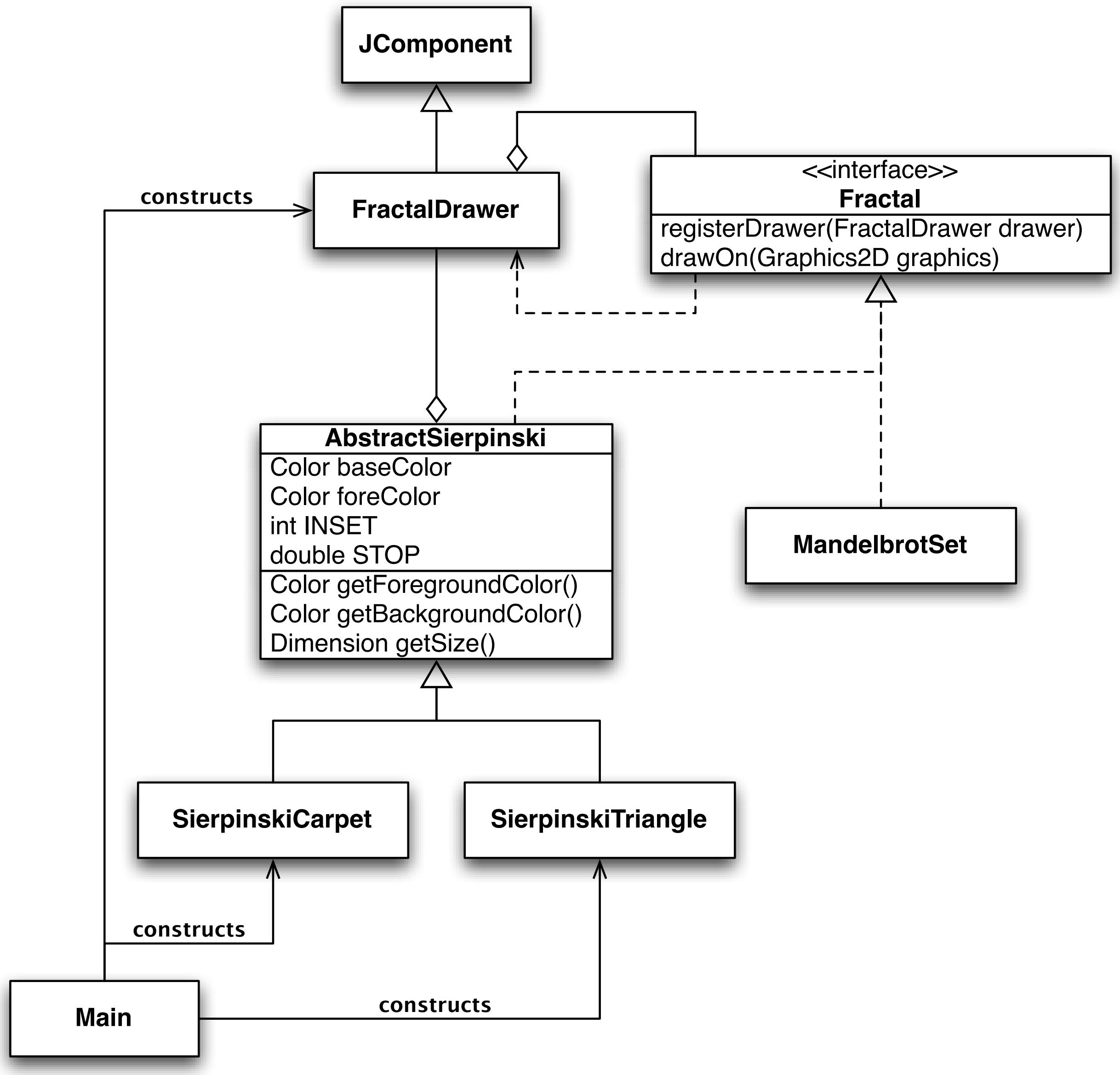
a. List at least **six** *candidate classes* that you might use in implementing a solution to the problem:

b. Pick one of your candidate classes, **circle it above** and briefly describe three responsibilities that it might have:

-
-
-

c. Still considering your chosen class from part b, with what other classes might it need to collaborate?

3. (12 points) Use this UML class diagram to answer the subsequent questions.



a. Which class or classes construct(s) instance of FractalDrawer? ________________________

b. If we added a method to the Fractal interface, *at a minimum*, what other classes or interfaces must be changed?

c. What does the SierpinskiCarpet class inherit from AbstractSierpinski?

4. (4 points) Describe a circumstance where you would make a method `private`.

5. (4 points) (Check **all** that apply.) Major difference(s) between an interface and an abstract class is/are:

___ They are two words for the same idea, so there is no difference.

___ An interface cannot be instantiated, but an abstract class can be.

___ An abstract class cannot be instantiated, but an interface can be.

___ An interface cannot provide instance fields or implementations of methods, but an abstract class can do so.

6. (9 points) Suppose we're using **selection sort** on an array that initially contains the values

6	7	1	2	5

Show the values after…

a. …the first swap: ___ ___ ___ ___ ___

b. …the second swap: ___ ___ ___ ___ ___

c. …the third swap: ___ ___ ___ ___ ___

7. (16 points) Consider the following related declarations:

```
interface Top {
    public void alpha();

    public void beta();
}

class Zero implements Top {
    public void alpha() {
        System.out.println("X");
    }
    public void beta() {
        System.out.println("Y");
    }
}
```

```
class One implements Top {
    public void alpha() {
        System.out.print("A");
        this.beta();
    }

    public void beta() {
        System.out.println("B");
    }
}

class Two extends One {
    public void beta() {
        System.out.println("C");
    }
    public void gamma() {
        System.out.println("D");
    }
}
```

a. Draw a UML diagram to represent the given interface and classes:

b. Continuing the same problem, suppose we declare and initialize these variables:

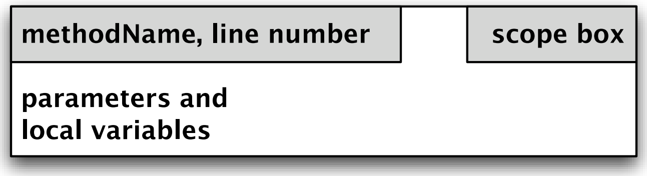
```
Top p = new Zero();
Top q = new One();
One r = new Two();
Two s = new Two();
```

For each line of code below, **circle** what would be output. If a line would give an error, then circle the type of error. Consider each line separately. That is, if a line would give an error, then assume that line doesn't affect any others.

Code	Output Choices (circle one in each line)								
p.gamma();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*
q.gamma();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*
r.gamma();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*
s.gamma();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*
p.beta();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*
q.beta();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*
r.beta();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*
s.beta();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*
p.alpha();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*
q.alpha();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*
r.alpha();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*
s.alpha();	X	Y	AB	AC	B	C	D	*runtime error*	*compile error*

8. (12 points) For this problem use the frame technique we practiced in class and the class declaration at left. Trace the execution of the call g(8) in main() and answer the question at the bottom of the page. A frame template is provided for your reference.

```
 1  public class Recur {
 2      public static void main(String[] args) {
 3          Recur rec = new Recur();
 4          int answer = rec.g(8);
 5
 6          System.out.println(answer);
 7      }
 8
 9      private int g(int n) {
10          if (n <= 1)
11              return 2;
12          int p = this.g(n / 2);
13          return n - p;
14      }
15  }
```



For the code above, *what would the final output be*? ____________________________

Name: ______________________________

Part 2—Computer Part

Resources for Part 2: 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. Any communication with anyone other than the instructor or a TA during the exam may result in a failing grade for the course.

You must turn in the preceding pages before accessing the resources for part 2.

Instructions. You must actually get these problems working on your computer. Almost all of the credit for this 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.

After you have handed in part 1, **begin part 2 by checking out the project named *Exam2* from your course SVN repository**. (Ask for help immediately if you are unable to do this.)

When you have finished the problems, and more frequently if you wish, you should **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 Descriptions

C1. (15 points) The package dots contains an interface Dotter. The class MyDotter in that package is declared to implement Dotter. Your task is to:

a. study the Dotter interface
b. study the use of the interface in DotDisplay, and
c. finishing the implementation of MyDotter.

To test your program, run the dot.Main class. With a correct implementation, left-clicks in the window will add orange squares to the display. Right-clicks will add orange circles. The Dotter interface is shown in Figure 1 on page 9

C2. (10 points) This problem is about recursion. Recall Sierpinski's triangle, which we saw on a homework assignment and is shown in Figure 2 on page 10. Unlike the homework, you don't have to draw the triangle. Instead, you have to implement a method to calculate the area of the shaded (red) portion of the triangle.

The class SierpinskiTriangle in the recur package includes stubs and documentation for the method that you should implement. Your solution **must use recursion**. Figure 3 on page 11 shows the stub and documentation. You'll find JUnit tests for this problem in SierpinskiTriangleTest.

```java
package dots;

import java.awt.Shape;
import java.util.ArrayList;

/**
 * Implementations of this interface store lists of dots at given coordinates.
 * "Dots" may be circles or squares.
 *
 * @author Curt Clifton
 */
public interface Dotter {
    /**
     * @return a list of all the "dots" in this collection
     */
    ArrayList<Shape> dots();

    /**
     * Adds a "dot" to this collection at the given coordinates.
     *
     * @param x
     * @param y
     */
    void leftClickAt(int x, int y);

    /**
     * Adds a "dot" to this collection at the given coordinates.
     *
     * @param x
     * @param y
     */
    void rightClickAt(int x, int y);
}
```

Figure 1: Dotter interface.

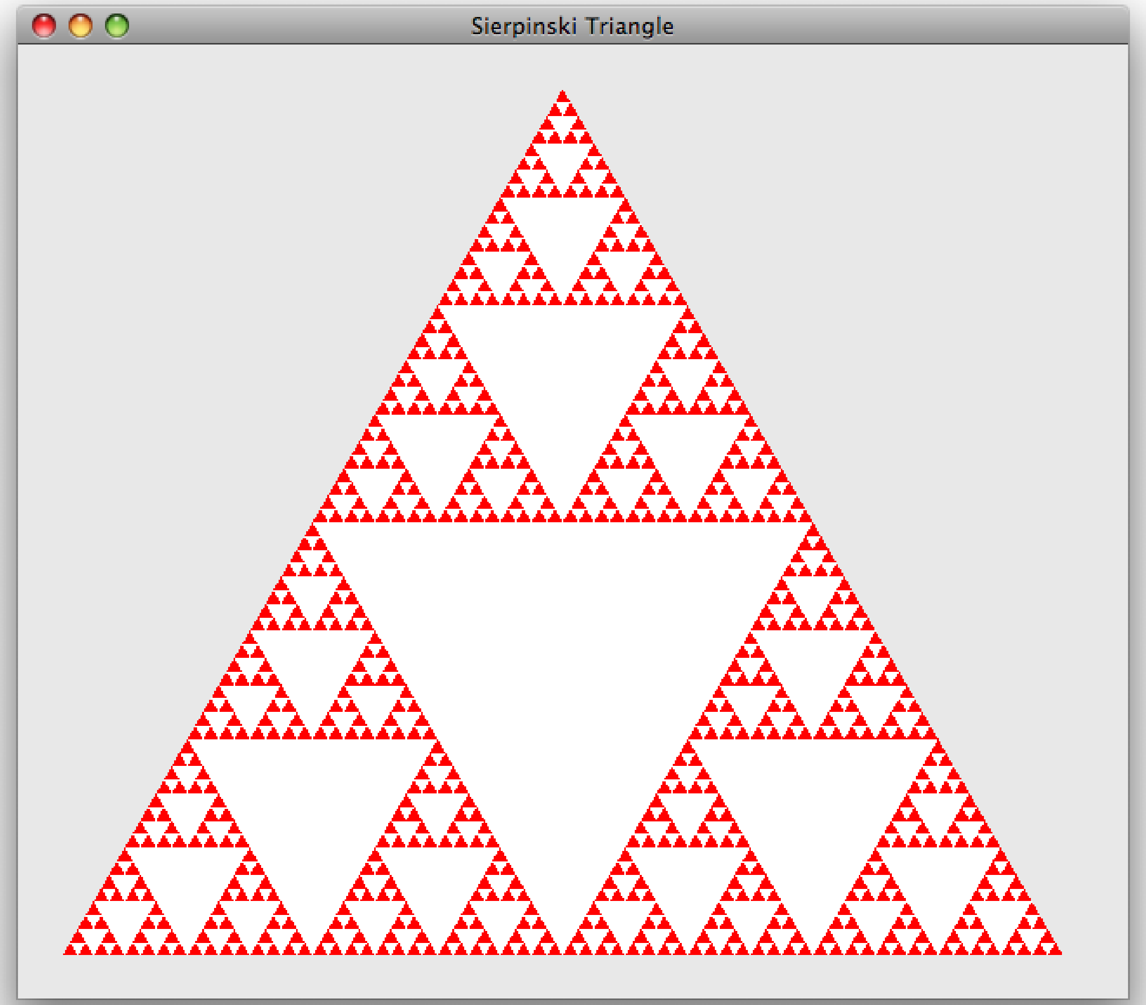


Figure 2: Sierpinski's Triangle

```
package recur;

/**
 * A Sierpinski triangle.
 *
 * @author Curt Clifton. Created Oct 29, 2009.
 */
public class SierpinskiTriangle {
    private final long sideLength;

    /**
     * Constructs a Sierpinski triangle with sides of length n.
     *
     * @param n
     */
    public SierpinskiTriangle(long n) {
        this.sideLength = n;
    }

    /**
     * @return the area inside this triangle
     */
    public double area() {
        double height = this.sideLength * Math.sqrt(3) / 2;
        return height * this.sideLength / 2;
    }

    /**
     * Calculates the area of the shaded portion of the triangle. For a side
     * length of 1, the shaded area is just three-quarters of the total area.
     * For larger triangles, calculates the shaded area recursively.
     *
     * @return the area of the shaded portion of this triangle
     */
    public double shadedArea() {
        /*
         * TODO: implement this method recursively, you may use a helper method
         * but are not required to do so.
         */
    }
}
```

Figure 3: SierpinskiTriangle class with documented stub to be implemented recursively.