

Name: \_\_\_\_\_

# CSSE 220—Object-Oriented Software Development

Exam 2, Oct. 30, 2008

This exam consists of two parts. Part 1 is to be solved on these pages. You may use the back of a page if you need more room. Please indicate on the front if you do so. 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:* Open book and notes, closed computer.

*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.

<b>Problem</b>	<b>Poss. Pts.</b>	<b>Earned</b>
1	8	_____
2	12	_____
3	16	_____
4	4	_____
5	2	_____
6	6	_____
7	4	_____
8	18	_____
9	6	_____
<b>Paper Part Subtotal</b>	<b>76</b>	_____
C1. MyDotter dots()	4	_____
C1. MyDotter leftClickAt()	4	_____
C1. MyDotter rightClickAt()	4	_____
C2. Sentence find() (1 per unit test)	6	_____
C2. Sentence indexOf() (1 per unit test)	6	_____
<b>Computer Part Subtotal</b>	<b>24</b>	_____
<b>Total</b>	<b>100</b>	_____

## Part 1—Paper Part

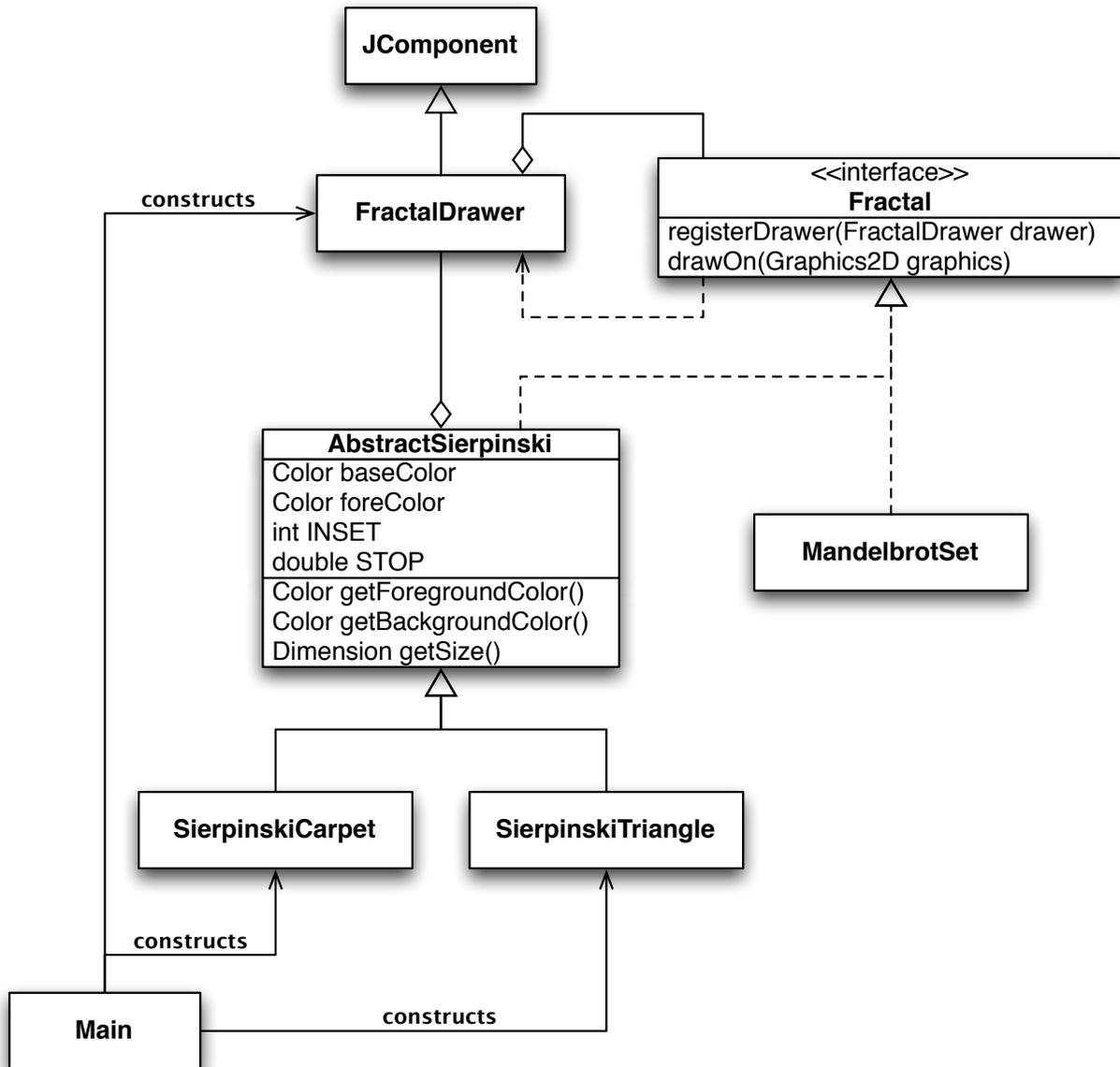
1. (8 points) Define *cohesion* and *coupling* and explain why the two properties are often in opposition.

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

*Design a program to track student report cards for all the students at Big Research University (BRU). A report card at BRU lists courses, instructors, midterm and final grades, credits for each course and total for the term, cumulative and term GPA, and the student's major.*

- a. List several *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. (16 points) Use this UML class diagram to answer the subsequent questions.



- Which class or classes construct(s) instance of **FractalDrawer**? \_\_\_\_\_
- If the **Fractal** interface changes, what other classes or interfaces might be affected?
- What does the **SierpinskiTriangle** class inherit from **AbstractSierpinski**?

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

5. (2 points) (Check **all** that apply.) Major difference(s) between an interface and an abstract class is/are:

- An interface cannot provide instance fields or implementations of methods, but an abstract class can do so.
- An interface cannot be instantiated, but an abstract class can be.
- They are two words for the same idea, so there is no difference.
- An abstract class cannot be instantiated, but an interface can be.

6. (6 points) Thinking about the Ball Worlds programming project...

a. Why did it make sense for Shrinker and Exploder to inherit from Bouncer?

b. Why didn't Exploder inherit from Shrinker?

7. (4 points) What has been the most challenging part of the Vector Graphics programming project thus far?

8. (18 points) Consider the following two classes:

```

class One {
    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");
    }
}

```

Suppose we declare and initialize these variables:

```

One p = new One();
One q = new Two();
Two r = 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();	AB	AC	B	C	D	<i>runtime error</i>	<i>compile error</i>
q.gamma();	AB	AC	B	C	D	<i>runtime error</i>	<i>compile error</i>
r.gamma();	AB	AC	B	C	D	<i>runtime error</i>	<i>compile error</i>
p.beta();	AB	AC	B	C	D	<i>runtime error</i>	<i>compile error</i>
q.beta();	AB	AC	B	C	D	<i>runtime error</i>	<i>compile error</i>
r.beta();	AB	AC	B	C	D	<i>runtime error</i>	<i>compile error</i>
p.alpha();	AB	AC	B	C	D	<i>runtime error</i>	<i>compile error</i>
q.alpha();	AB	AC	B	C	D	<i>runtime error</i>	<i>compile error</i>
r.alpha();	AB	AC	B	C	D	<i>runtime error</i>	<i>compile error</i>

9. (6 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(4)` in `main()` and answer the question at the bottom of the page. A frame template is provided for your reference.

```
public class Recur {  
    public static void main(String[] args) {  
        int answer = g(4);  
        System.out.println(answer);  
    }  
  
    private static int g(int n) {  
        if (n == 0)  
            return 1;  
        int p = g(n - 1);  
        return n - p;  
    }  
}
```

methodName, line number	scope box
parameters and local variables	

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

## 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. (12 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 circles to the display. Right-clicks will add orange squares. The `Dotter` interface is shown in Figure 1 on page 8

C2. (12 points) This problem is about recursion. The class `Sentence` in the `text` package includes stubs and documentation for two methods that you should implement. Your solutions **must use recursion**. Figure 2 on page 9 shows the stubs and documentation. You'll find JUnit tests for this problem in `SentenceTest`.

*Hints:*

- For `find()`, if the text starts with the string you want to match, then you're done.
- For `indexOf()`, think about adding 1 to the result of a recursive call.

```

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.

```

package text;

/**
 * This class implements a sentence with some basic search methods.
 *
 * @author Curt Clifton.
 */
public class Sentence {
    private String text;

    /**
     * Constructs a sentence object with the given text.
     *
     * @param text
     */
    public Sentence(String text) {
        this.text = text;
    }

    /**
     * Searches for the given string in this sentence.
     *
     * @param t
     * @return true if t is in this sentence
     */
    public boolean find(String t) {
        // TODO: implement this method recursively, you may use a helper method
        throw new RuntimeException("delete this line");
    }

    /**
     * Finds the starting position of the first substring of this sentence that
     * matches the given string. Or returns -1 if t is not a substring of this
     * sentence.
     *
     * @param t
     * @return the starting position, or -1 if not found
     */
    public int indexOf(String t) {
        // TODO: implement this method recursively, you may use a helper method
        throw new RuntimeException("delete this line");
    }
}

```

Figure 2: Sentence class with documented stubs to be implemented recursively.