CSSE 220 – Object-Oriented Software Development

Exam 2 Topics

The exam has two parts, just as Exam 1 did:

- A closed-book, paper-and-pencil section. Topics for this section are drawn *only* from the first list below. Questions might be short answer, fill-in-the-blank, multiple-choice, true/false, or problems whose answer is a short code fragment.
- An open-book, on-the-computer section. You write code and turn it in via your individual repository. See the second topic list below.

For the closed-book portion, you can use a single 8.5 inch by 11 inch "cheat sheet" (back and front, with whatever you want on it, handwritten or typed). You may work with others to make your cheat sheet, but you will probably find it most useful if you do much or all of it yourself.

For the open-book portion, you may use any written source, anything you can find on the internet, and your computer. You may not communicate with any human being except your instructor in any way during the exam, however.

The closed book portion will be about 40% of the total points; the open book portion will be about 60% of the total points.

Closed-book topics, things you should know:

1. **Object-oriented design** (as in your VectorGraphics and other projects):
   a. What is ***cohesion***? Does good OO design call for low cohesion or high cohesion? Why?
   b. What is ***coupling***? Does good OO design call for low coupling or high coupling? Why?
   c. What are ***immutable*** classes? Does good OO design call for mutable classes or immutable classes? Why?
   d. Why does OO design encourage the use of ***inheritance***?
   e. Why does OO design encourage the use of ***interfaces***?
   f. What are ***CRC cards*** and how are they used in OO design?
   g. What are ***UML class diagrams*** and how are they used in OO design?
2. **Polymorphism** (as in the polygon package in your OnToInterfaces project)
   a. What is polymorphism? Why is it helpful?
   b. Why do you sometimes need to **cast** objects? What is the notation for doing so?
   c. Suppose that class X extends class Y. Which of the following are legal?

```
X a = new Y(...);
Y b = new X(...);
```

   d. Why do you sometimes need to use the **instanceof** operator? What is the notation for doing so?
   e. Suppose you have an ArrayList<Shape> and then iterate through the ArrayList, asking each Shape to draw itself. Where is polymorphism being used in this example? Why is polymorphism helpful here (what's the alternative)?

3. **Interfaces** (as in your BigRational project)
    a. If X ***implements*** interface Y, what does that mean?
    b. Why is an interface like a contract? What is the contractual agreement?
    c. Why are interfaces useful? How are they used?
4. **Inheritance** (as in the banking package of your Inheritance project and your BallWorlds project)
    a. What is inheritance? Why is it useful?
    b. If X ***extends*** Y, which is the subclass and which is the superclass?
    c. What class is at the top of the inheritance hierarchy?
    d. How and why do you use **super.blah(…)** and **super(…)**?
    e. A subclass can inherit methods, ***override*** methods (either replacing or augmenting the overridden method), or add entirely new methods.
    f. A subclass inherits all fields unchanged and can add new fields. What is ***shadowing*** and why is it bad?
5. **Exceptions** (as in your FilesAndExceptions project)
    a. What is an Exception?
    b. How do you ***throw*** an Exception? When do you do so?
    c. How do you declare that a method ***throws*** an Exception?
    d. How do you handle an Exception with a ***try/catch*** block?
    e. What is the difference between ***checked*** and ***unchecked*** Exceptions?
    f. When you write a statement that may throw a checked Exception, what are your two choices regarding that Exception. (Answer: let it propagate to the method that called this one, or handle it via a try/catch).
6. **Recursion** (as in your Recursion and Recursion2 project)
    a. What is recursion?
    b. What is a base case and why is it important?
    c. How do you write recursive methods?
    d. What is a memory table and how do you use it?
7. **Visibility** (as in your text and the BallWorlds project)
    a. What are the four visibility modifiers? What does each allow?
    b. What visibility do fields usually have? Methods? Classes?
8. What are ***static fields*** and when do you use them? ***Static methods***? (As in your Static project)

On-the-computer topics: Expect two problems: one that requires a graphical user interface and one that requires a recursive solution. In these problems, you will write code that demonstrates your understanding of:

1. **Graphical User Interfaces using Swing** (as in the SwingDemo2, LinearLightsOut and VectorGraphics projects)
    - Constructing and displaying JFrames, JComponents, JPanels, JButtons, etc.
    - Adding components to containers. Using a LayoutManager.
    - Drawing on components.
    - Responding to events: button-presses, mouse-clicks, etc.
        - How to respond to events
        - Style: Who should respond to an event?
2. **Implementing has-a relationships** (as in the SwingDemo2, GameOfLife, LinearLightsOut and VectorGraphics projects)
    - How to pass objects to other objects:
        - Through constructors
        - Through setters
    - Why pass objects to other objects
3. **Implementing is-a relationships – inheritance** (as in the BallWorlds and Banking projects)
    - How to have one class extend another, and what that gains for you
    - Your choices for methods:
4. **Implementing is-a relationships – interfaces** (as in the BigRational project and elsewhere)
5. **Recursion** (as in your Recursion and Recursion2 project)
    - How do you write recursive methods?
    - How do you use a memory table?
6. **Anonymous classes** (as in lecture notes and your textbook)
    - How to write an anonymous class
    - Using variables from the outer class/method in the (inner) anonymous class
    - Why use anonymous classes
7. **Static** fields and methods (as in the Static project)
    - How to use them. When to use them.