CSE 331, Spring 2011
Final Exam
Monday, June 6, 2011

Name: _____

Section: _____ **TA:** _____

Student ID #: _____

- You have **110 minutes** to complete this exam. You must stop working once the instructor calls for papers. You may receive a deduction if you keep working after the instructor calls for papers.
- The exam is open-book/notes. You must work alone and may not use any computing devices including calculators. Cell phones, music players, and other electronics may NOT be out during the exam for any reason.
- Please be quiet during the exam. If you have a question or need, please raise your hand.
- Corrections or clarifications to the exam will be written at the front of the room.
- Please obey the University Code of Conduct during the exam.
- Unless otherwise specified, you don't need to write any `import` statements or any comments on your code. Your code will be graded on "external correctness" and also might be graded on style or "internal correctness" if the particular question states as such.
- When you have finished the exam, please turn in your exam quietly and leave the room.

Please do not begin working until notified by the instructor.
Good luck!

Score summary: (for grader only)

Problem	Description	Earned	Max
1	Java Object Methods		15
2	Specs and Comments		15
3	JUnit Testing		20
4	Object-Oriented Design		20
5	Design Patterns		15
6	Graphical User Interfaces		15
TOTAL	**Total Points**		**100**

1. Java Object Methods

Given the following class, add all of the following things:

(a) Make it possible to compare objects of this type for **equality**.
(b) Make these objects behave properly when placed into **hash-based collections**, following the general contract expected by the language, as well as distributing objects with unequal state relatively evenly.
(c) Implement a **natural ordering** on the objects that compares them by *year* in ascending order, breaking year-ties by *number of courses* ascending, breaking course-ties by comparing *names* in ABC order.
(d) Make it possible to create a **deep copy** of the object and all of its state.
(e) Make it possible for clients to save objects of the class to disk using object **serialization**. (You may assume that any other classes in the system have already been modified appropriately to allow them to be serialized.)

In all cases, you should follow the general contract expected by the language. If an illegal parameter value is passed to any method, throw an `IllegalArgumentException` except if the method's contract would expect otherwise. You may assume that the `Course` class is already implemented in such a way as to satisfy (a) - (e) above.

Add any methods necessary and make any necessary modifications to the class header. You should not add any fields or constructors to the class. None of your methods should modify any `Student` object's state.

You may use the next page for additional writing space if necessary.

```
// Invariants: courses != null,  name != null,  1 <= year <= 4,  0.0 <= gpa <= 4.0

public class Student
{
    private List<Course> courses;
    private String name;
    private int year;
    private double gpa;

    ...

    // your code goes here
```

1. **Java Object Methods (WRITING SPACE)**

2. Specs and Comments

Given the following method, write a well-formed specification for it as a Javadoc comment header. Follow the guidelines taught in class for writing a proper method specification. In your comment header, make sure to include all of the following important items that we learned in class:

- A description of the method's behavior.
- Proper tags for other information about the method's parameters, return values, and exceptions thrown.
- Information about assumptions or guarantees made by the method. For full credit, your spec should include any assumptions that must be true for the method to properly complete its behavior, and at least one guarantee about the method's behavior if those assumptions are met.
- Any objects modified by the method, along with the specific effects that will occur on those objects.

Use Javadoc syntax the way the code would appear in a Java source file. You may use custom `@` tags as needed.

(Note: The code actually contains a subtle bug. Your spec should describe the code's intended behavior.)

```
public static int replaceAll(List<String> list, String from,
                             String to, boolean caseSensitive) {
    if (list == null || from == null) {
        throw new IllegalArgumentException();
    }

    int count = 0;
    for (int i = 0; i < list.size(); i++) {
        String element = list.get(i);
        if (caseSensitive && element.equals(from) || element.equalsIgnoreCase(from)) {
            list.set(i, to);
            count++;
        }
    }

    return count;
```

```
}
```

3. JUnit Testing

Write a brief JUnit test file to test the `replaceAll` method from the previous question. In particular, your test should ensure that the method's core functionality works properly for an interesting set of parameter values, and that the method deals with illegal parameter values as expected. Your tests should fail if the method has an infinite loop. Your JUnit code should follow the guidelines for good unit testing as covered in lecture.

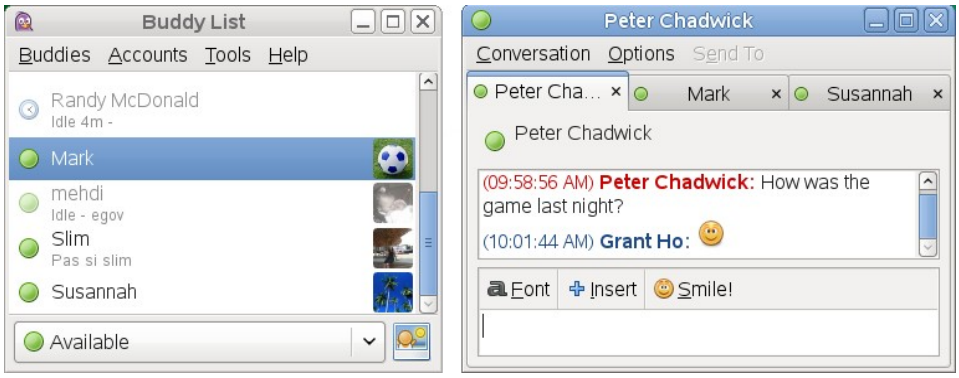
We do want you to show us a set of tests that examine several different cases, but it isn't our intention to make you write 200 lines of code. Test at least 2-3 "valid" cases and an "invalid" case for each unique type of invalid call. To reduce redundancy and the size of your code, you may want to make a helper method(s). Recall that you can quickly construct a list of strings using the `Arrays.asList` method. For example:

```
List<String> list = Arrays.asList("a", "b", "c");
```

4. Object-Oriented Design

Suppose you are asked to solve the following problem. Write down the important classes you would use for the program's model (the core underlying data; not the user interface) and a brief description of the state and behavior for which each class would be responsible. (You can just list field/method names, and/or describe the functionality in English.) For full credit, your design should follow the general object-oriented design principles taught in class.

You are going to write an instant messenger chat program similar to AOL Instant Messenger or Google Chat. The program will allow a user to create one or more accounts, each with a user name, email address, password, and avatar image. Each account is associated with a buddy list of friends. You can also "block" a user if you don't want to see that user's messages to you. A user can log in and log out. A logged-in user can set a status such as Available, Away, or Do Not Disturb. The user can have instant message (IM) conversations with other users, including multiple active conversations at any time. The application saves logs of past conversations that can be browsed later.



Write each class in roughly the following format:

- Class Name
- Brief Description
- Fields
- Methods

You don't need to include trivial "getter" and "setter" methods. Here is an example for a `Point` class:

Point
description: Each Point object represents a 2D (x, y) Cartesian point in the integer coordinate space.
fields: x, y
methods: translate(dx, dy), distance(p2), draw(g)

Write your answer on the next page.

4. Object-Oriented Design (WRITING SPACE)

5. Design Patterns

Each problem below contains a written description of a programming situation that utilizes (or ought to utilize) a pattern we have discussed in class. Write the appropriate pattern name next to each situation.

(a) You are writing a financial class named `Portfolio`. Your `Portfolio` class has an `add` method that takes an `Investment` argument; this method allows a user to add items to his/her financial portfolio. However, `Portfolio` is declared to extend `Investment`, so it is possible to add an entire other portfolio to your `Portfolio`.

Pattern being used: _____

(b) Your World of CSEcraft game is slow, because you frequently have to load bulky character classes from the hard disk. The character classes are unique and immutable, and each consumes a lot of memory. Any two clients that want to access the same character class could theoretically share the same object. What pattern would be useful here?

Pattern to use: _____

(c) Your poker game goes through various game phases, such as: a pre-round phase where players must place "ante" bets onto the table; an initial round of ante betting; a "flop" phase where three shared cards have been dealt; and various betting round phases where players bet various amounts and match, raise, or fold. After a player wins a round, the game goes back into another pre-round (or between-round) phase. The app is complicated and it is important for various code to know which phase the game is in, in order to behave properly. Which pattern would be useful here?

Pattern to use: _____

(d) Your network project is centered around a particular `ServerData` object on the server that is important to all the connected clients. When a client connects to the server, it wants to know about any changes that occur in the state of the `ServerData` object. When this object's state changes, any connected clients want to know about this so that they can respond accordingly. But it is slow and expensive for the clients to repeatedly ask the server data object if its state has been changed, so we'd rather have the server data object inform them when such a change occurs. What pattern would be useful to use here?

Pattern to use: _____

(e) In your code, you often find yourself constructing `Shape3D` objects, then setting several properties of the object, and so on. To make this less tedious, you write a `ShapeMaker` class with static methods that you pass appropriate arguments to, and it returns a fully built `Shape3D` with all properties set appropriately.

Pattern being used: _____

(f) Your application uses various kinds of Java input streams to read data. You decide that you'd prefer to read all data in lowercase. So you provide a `LowercaseInputStream` that accepts another `InputStream` as a parameter to its constructor and wraps the original stream with one that reads all of its data as lowercase characters. The lowercase input stream implements the same interfaces as the original stream, so it can be used in place of the original.

Pattern being used: _____

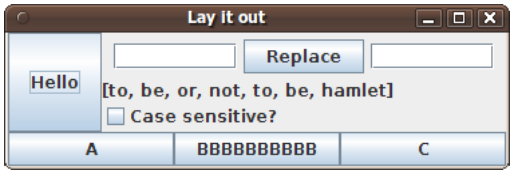
(g) Your project needs to connect to a MySQL database. The existing Java JDBC API for connecting to the database demands that you pass it various objects that implement the `DatabaseConnector` interface. Such an object must implement several complex required methods, but you only want to use one or two of them. So you write a `SimpleConnector` class that your actual connectors can extend, which implements default versions of most of the methods and leaves the rest empty for subclasses to override with their specific behavior.

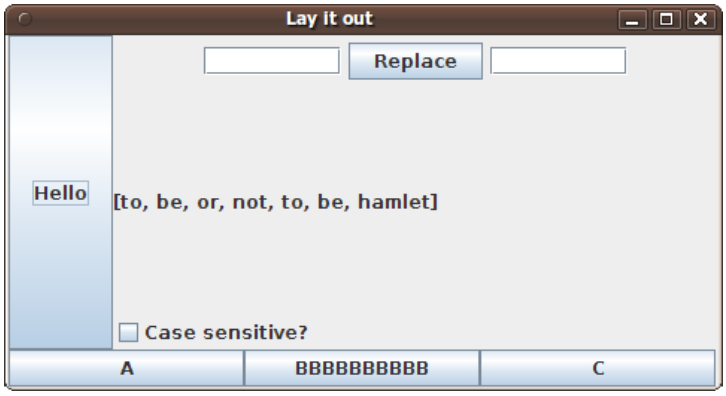
Pattern being used: _____

6. Graphical User Interfaces

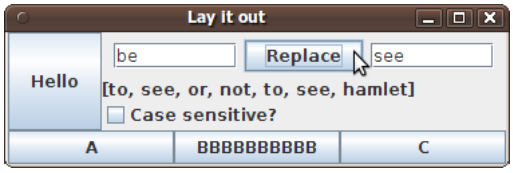
Finish the following Java program that creates a GUI and shows it on the screen. Create the window with a **layout** as pictured and fill it with the components necessary. The window's title should be "Lay it out" and initially the window should be sized exactly to fit the components inside it. When the window is closed, the program should terminate.

The following screenshots show the program's initial appearance and after resizing the window:





In addition to matching the layout shown, modify the code so that when the user clicks the Replace button, any occurrences of the word in the left text field are replaced by the word in the right text field:



Start from the following partial code. Complete the class by writing your answer on the next page.

```
public class LayoutGui {
    // This is the method from the previous questions.
    public static int replaceAll(List<String> list, String from,
                                 String to, boolean caseSensitive) { ... }

    // Fields;  bad style to initialize at declaration, but oh well.
    private JButton a = new JButton("A");
    private JButton b = new JButton("BBBBBBBBBB");
    private JButton c = new JButton("C");
    private JButton hello = new JButton("Hello");
    private JButton replace = new JButton("Replace");
    private JCheckBox sensitive = new JCheckBox("Case sensitive?");
    private JFrame frame = new JFrame();
    private JLabel listLabel = new JLabel();
    private JTextField from = new JTextField(8);
    private JTextField to = new JTextField(8);
   private List<String> list = Arrays.asList("to", "be", "or", "not", "to", "be", "hamlet");

    public LayoutGui() {
        // your code begins here ;  write your answer on the next page
```

6. Graphical User Interfaces (WRITING SPACE)