CSE 331 Final Exam 3/12/12

Name _____

There are 12 questions worth a total of 100 points. Please budget your time so you get to all of the questions. Keep your answers brief and to the point.

The exam is closed book, closed notes, closed electronics, closed telepathy, etc.

Many of the questions have short solutions, even if the question is somewhat long. Don't be alarmed.

If you don't remember the exact syntax of some command or the format of a command's output, make the best attempt you can. We will make allowances when grading.

Relax, you are here to learn.

Please wait to turn the page until everyone is told to begin.

Score _____ / 100

1. _____ / 12
2. _____ / 12
3. _____ / 15
4. _____ / 10
5. _____ / 8
6. _____ / 4
7. _____ / 6
8. _____ / 10
9. _____ / 12
10. _____ / 5
11. _____ / 4
12. _____ / 2

Question 1. (12 points) Proofs. Give a proof involving appropriate assertions, preconditions, and postconditions that shows that the following code fragment stores the absolute value of variable `x` in variable `a`.

```
a = x;
if (a < 0) {
   a = -x;
}
```

Write your proof below, copying the code and adding appropriate assertions as needed. You should assume that all variables contain integer values and you may assume that overflow will not occur, i.e., you assume that it is always possible to compute and store the value `-x` in an integer variable. You may not alter the given code except that you can add an empty "else" part to the `if` statement if that is useful in structuring your answer.

Question 2. (12 points) Specification. Consider the following four specifications for `double sqrt(double x)`, a method that returns the square root of its argument.

A
```
@requires x >= 0
@return y such that |y*y - x| <= 0.0001
```

B
```
@requires x >= 0
@return y such that |y*y - x| <= 0.01
```

C
```
@return y such that |y*y - x| <= 0.0001
@throws IllegalArgumentException if x < 0
```

D
```
@requires x > 0    // i.e., requires x > 0, not x >= 0
@return y such that |y*y - x| <= 0.0001
```

For each of the following pairs of specifications, circle the **stronger** specification, or circle "neither" if the two specifications are either equivalent or incomparable.

(a) A B neither

(b) A C neither

(c) A D neither

(d) B C neither

(e) B D neither

(f) C D neither

Question 3. (15 points) Equality. The three parts of this question on the following pages use the following class that represents a two-dimensional point. Constructors and other methods are omitted.

```
public class Point {
  private double x;
  private double y;

  public boolean equals(Object o) {
    if (o instanceof Point) {
      Point other = (Point) o;
      return x == other.x && y == other.y;
    } else {
      return false;
    }
  }
}
```

Answer the parts of this question on the following pages. You may remove this page for reference if you wish.

Question 3. (cont.) (a) (5 points) Consider the following subclass of class Point.

```
public class LabelPoint extends Point {
  private String label;

  public boolean equals(Object o) {
    if (o instanceof LabelPoint) {
      LabelPoint other = (LabelPoint) o;
      return super.equals(o) && label.equals(other.label);
    } else {
      return false;
    }
  }
}
```

Indicate whether each of the following properties holds for the `equals` method. If the property does not hold (answer is false) give a counterexample showing why the property fails.

(a) T / F Reflexivity

(b) T / F Symmetry

(c) T / F Transitivity

Question 3. (cont.) (b) (5 points) Consider the following subclass of class Point.

```
public class Point3D extends Point {
  private double z;

  public boolean equals(Object o) {
    if (o instanceof Point3D) {
      Point3D other = (Point3D) o;
      return super.equals(o) && z == other.z;
    } else if (o instanceof Point) {
      return super.equals(o);
    } else {
      return false;
    }
  }
}
```

Indicate whether each of the following properties holds for the `equals` method. If the property does not hold (answer is false) give a counterexample showing why the property fails.

(a) T / F Reflexivity

(b) T / F Symmetry

(c) T / F Transitivity

Question 3. (cont.) (c) (5 points) Consider the following subclass of class Point.

```
public class ColorPoint extends Point {
  private Color color;

  public boolean equals(Object o) {
    if (o instanceof Point) {
      return o.equals(this);
    } else if (o instanceof ColorPoint) {
      ColorPoint other = (ColorPoint) o;
      return super.equals(o) && color.equals(other.color);
    } else {
      return false;
    }
  }
}
```

Indicate whether each of the following properties holds for the `equals` method. If the property does not hold (answer is false) give a counterexample showing why the property fails.

(a) T / F Reflexivity

(b) T / F Symmetry

(c) T / F Transitivity

Question 4. (10 points) Hashcodes. Consider the following class that represents entries in a mailing list.

```
public class AddressCard {
  private String first, last;      // first and last names
  private String address;          // street address, city, state
  private int zip;                 // zip code

  @Override
  public boolean equals(Object o) {
    if (o instanceof AddressCard) {
      AddressCard other = (AddressCard) o;
      return first.equals(other.first)&&last.equals(other.last);
    } else {
      return false;
    }
  }

  @Override
  public int hashCode() {
    return _____________________;
  }
}
```

(a) Here is a list of possible expressions that could appear in the `return` statement in the above `hashCode` method. Circle `ok` for each choice if it would result in a legal, correct `hashCode` implementation for this class. Circle `bug` if it would result in an incorrect `hashCode` implementation. You do not need to justify your answer. Your answer should not consider whether a particular answer is a good `hashCode` function, only whether it is correct or not.

(i) ok / bug `return last.hashCode();`

(ii) ok / bug `return first.hashCode()*31 + last.hashCode();`

(iii) ok / bug `return zip;`

(iv) ok / bug `return 42;`

(b) Of the legal `hashCode` implementations in the list above, which one is likely to be the best quality hash code and why? If none of the above implementations are legal simply write none. (be brief)

Question 5. (8 points) Rep exposure. Consider the following code fragment from a class definition:

```
public class ObnoxiousFinalExamQuestion {
  private List<String> names;
  private final List<String> fixedNames;
  private final List<JButton> buttons;
  ...
  public List<String> getNames()      { return names; }
  public List<String> getFixedNames() { return fixedNames; }
  public String getFirstName()        { return names.get(0); }
  public JButton getFirstButton()     { return buttons.get(0); }
}
```

Assume that all of the lists are defined after an instance of the class is constructed and that each list contains at least one item.

Which of the methods in this class could potentially cause a representation exposure? Below, circle yes if the method could cause a representation exposure. Circle no if it does not. For each method that does create a potential representation exposure, give a 1-sentence explanation of why it does.

(a) yes / no `getNames()`

(b) yes / no `getFixedNames()`

(c) yes / no `getFirstName()`

(d) yes / no `getFirstButton()`

Question 6. (4 points) Types and design. Suppose you are writing an application that uses "incrementable" objects. An object is "incrementable" if it has `increment()` and `decrement()` methods that modify its state to be slightly greater than or less than its previous state, respectively. You enforce this by requiring the client to supply objects of type `Incrementable`. Should `Incrementable` be a Java class, an abstract class, an interface, some combination of these, or none of them? Give a brief justification for your answer.

Question 7. (6 points) Version control. List three (3) distinct benefits of using version control software like Subversion. Include at least one benefit that applies even when working alone, and one that only applies when working in groups. Be brief.

(1)

(2)

(3)

Question 8. (10 points) Model, View, Controller. We'd like to make some improvements in the route-finder application from homework 6. For each of the changes below, indicate which module or modules of the program code would reasonably require modifications to implement the change by circling M, V, and/or C for Model, Viewer, and/or Controller. If none of those parts of the MVC architecture would require changes leave the answers blank.

You should answer the question based on how the Model/View/Controller architecture is organized, even if parts of the architecture overlapped in the actual classes in your code for homework 6. (i.e., base your answers on the standard MVC structure, not on specific quirks of your implementation.)

You do not need to justify your answers, but you may add brief justifications if you believe it will help us evaluate your answer more fairly.

(a) M / V / C Change the buttons so they have a purple background and gold text.

(b) M / V / C Store the path and building data in a database instead of a text file.

(c) M / V / C Add an option to display only handicapped-accessible routes.

(d) M / V / C Add the new Molecular Engineering Center to the list of buildings and remove path segments around the hub to reflect construction.

(e) M / V / C Fix the application so that pressing 'enter' on the keyboard has the same effect as clicking the "find route" button

.

Question 9. (12 points) Design patterns. We discussed several patterns from the list of well-known ones. As a reminder, the list included the following, not all of which we discussed:

- Creational: Factory, Singleton, Builder, Prototype, Interning, Flyweight
- Structural: Adapter, Composite, Decorator, Flyweight, Proxy
- Behavioral: Interpreter, Observer, Iterator, Strategy, Model-View-Controller, Visitor

For each of the following design problems, state the most appropriate design pattern to use to construct a solution and give a brief one-sentence explanation for your choice.

(a) Your program can send content to a printer. You want to ensure that different parts of the program don't command the printer to print output belonging to multiple documents simultaneously, mixed together.

(b) You are writing a library that allows client code to read and write compressed data for storing in files. You've written several classes that implement `encode()` and `decode()` methods of a `CompressedFile` interface. These classes use different algorithms with different characteristics. Rather than requiring the client to choose among them, you want the client to be able simply to request a `CompressedFile` object and be given an instance of the best available implementation for the particular kind of file.

(continued next page)

Question 9. (cont.) State the most appropriate design pattern for each application.

(c) You have an application that reads and writes files on the local file system. Without changing the existing application any more than absolutely necessary you would like to modify it so it can read and write files stored at a different location on the network, even though reading and writing data over the network requires using a different API and a different set of methods than reading and writing data stored in local files.

(d) You are writing a program that maintains a grid of objects. Although there are several million objects in the grid, there are only a few hundred distinct objects, but there are many copies of each one. Unfortunately each of the objects require a fair amount of storage and creating several million individual objects is using way too much storage.

Question 10. (5 points) A rather generic (adj.) question about generics (noun). Your colleague A. Hacker is trying to create a small class to hold an unordered collection of items, possibly containing duplicates. Hacker wants to use very simple generics so that different instances of the collection can hold different kinds of items. Here is the code:

```
/** a collection of elements of type E */
public class Bag<E> {
  E[] vals;        // data is stored in vals[0..size-1]
  int size;

  /** construct a new empty Bag */
  public Bag() {
    vals = new E[10];
    size = 0;
  }

  /** add new item x to this Bag */
  public void add(E x) {
    // ensure vals is large enough for the new item
    if (vals.length == size) {
      E[] newvals = new E[vals.length*2];
      for (int k = 0; k < vals.length; k++) newvals[k]=vals[k];
      vals = newvals;
    }
    // add new item
    vals[size] = x;
    size++;
  }
}
```

Unfortunately, the code doesn't compile, although an earlier non-generic version that used `Integer` instead of `E` for the element type worked just fine.

(a) Briefly describe the problem.

(b) How could the problem be fixed? Either describe your changes below, or, maybe better, circle the code above that needs to be fixed and show how to fix it.

Question 11. (4 points) Usability. Many applications have "confirm" dialogs that ask the user to click yes or no before some permanent change is made to the user's data. Another way to guard against accidental changes is to allow the user to "undo" changes. A confirm dialog is usually much easier to implement compared to an undo operation.

From the standpoint of good user interface design, is either of these methods preferable to the other? Give a brief justification of your answer.

Question 12. (2 points) What is the "rubber duck" method of debugging? (circle the letter of the best answer)

(a) Run many different inputs through your program, noting which ones "float" and which "sink" to collect as much information as possible about the bug.

(b) Talk through your code by describing it to a rubber duck or other inanimate object.

(c) Prevent bugs in the first place by making your code as simple as possible – so simple that even a rubber duck could understand the algorithms.

(d) If you can't find the bug, step away from the computer and give your mind a break, e.g. by taking a long bath with a rubber duck. Then come back and keep trying.

Have a great spring break!