CISC 3115 Final Review Guide

Readings

You are responsible for Chapters 1 through 9, 12, 13, and 16 of *Head-First Java, 2nd Edition,* as well as section 9.1 from *Introduction to Programming with Java, 8th Edition.*

Study Tactics

Work as many exercises from the book as you can.
Review all the questions I've asked you in this class.
Review all code from exercises, CodeLabs, etc.
Re-read readings (and reading guides) now that you've worked with ideas in lecture/lab.

Main Topics

Note: the final exam is *cumulative*, so all the material from the midterm review guide is part of your review materials for the final. The questions on the final will certain focus on topics we covered after the midterm—but of course, you will need to have a solid grasp on the first half of the course in order to do well.

Don't worry about memorizing methods… I'll provide the signatures of any methods (especially from Swing and Collections) you might need. But you'l have to know how to *use* them…

Unit 4

Swing and the Abstract Windowing Toolkit
Event-handling
Layout
Model-View-Controller concept
- as application design
- as component design

Inner classes

Unit 5

Generic types
The Java Collection Framework
Using generic methods and classes
Lists, Maps, and Sets

Unit 6

Recursive definitions
Recursive methods
Binary search

Exam Structure, Roughly

50%: short-answer/multiple-choice
30%: writing code
20%: analyzing code

Application Activities

(Unit 6 application activities coming soon…)

Remember that instead of writing `frame.getContentPane().add(button);`
you can just write `frame.add(button);`

What is `JFrame`'s superclass?

a) `javax.swing.Frame`
b) `java.awt.Frame`
c) `javax.swing.JComponent`
d) `java.awt.Component`

What is `JButton`'s superclass?

a) `javax.swing.Frame`
b) `java.awt.Frame`
c) `javax.swing.JComponent`
d) `javax.swing.AbstractButton`

Which class is common to the "ancestors" of both `JButton` and `JFrame`?

a) `java.awt.Container`
b) `javax.swing.JComponent`
c) `javax.swing.AbstractComponent`
d) `javax.swing.Base`

In Swing, Michael can designate a class as an event handler by
a) Extending the appropriate base class
b) Declaring the appropriate instance variable
c) Implementing the appropriate interface
d) Adding the appropriate Swing component.

Consider the process through which a Swing button-click causes "something to happen." How many objects are directly involved in this process? (Not counting, necessarily, the component that contains the button.)
a) 1
b) 2
c) 3
d) 4

To register an object as a listener for a button's clicks,
a) Pass a reference to the button to the object's `listenTo()` method
b) Create a new `EventRegistry` object, passing references to the object and the button.
c) Pass a reference to the button to the object's `actionPerformed()` method
d) Pass a reference to the object to the button's `addActionListener()` method

To draw graphics on a Swing component, Guri must ___________ its _______________ method

a) extend... `paint()`
b) overload... `paint()`
c) override... `paintComponent()`
d) implement... `paintComponent()`

If Michael's Swing code involves "painting" graphics, how often should Guri call the "painting" method (the one from the previous question)?

a) Never
b) When you change the graphics to be painted.
c) Whenever the Swing window is moved
d) 60 times a second, to make sure it looks smooth to the human eye

The next few questions are based on the `CircleSizes/CirclePanel` code handout.

First, look at variable declarations in `CircleSizes`. Note that `frame` is declared (line 17) as a local variable in `makeGUI`, but the two buttons and the `CirclePanel` are declared (lines 7-9) as instance variables (even though they don't get objects to refer to (19-21) until `makeGUI`!) So, why not just declare the buttons and the `CirclePanel` inside `main` along with `frame`?

a) That is a perfectly fine choice; either approach will work.
b) This will compile and run, but the buttons won't "work right."
c) This will compile, but we'll get an exception when it runs.
d) This won't compile.

I am now going to demonstrate a problem with the application's behavior [diameter goes negative]. Which methods need to be modified to fully address this problem?

a) `CircleSizes.actionPerformed`
b) `CirclePanel.CirclePanel`
c) `CirclePanel.decDiameter`
d) a and b
e) b and c

Rewrite this so that `CirclePanel` implements `ActionListener` and (therefore) handles the button-clicks. Specifically:

- write the new version of `actionPerformed`, as a member of `CirclePanel`
- write any other new/changed code in `CirclePanel`
- give the line numbers of any lines in `CircleSizes` that need to change (in addition to the obvious ones, 5 and 34-40)

In your folders is a printout of another version of the circle code from last week.

Which of these is *not* an inner class?

a) `CirclePanel`

b) `BigListener`
c) `SmallListener`
d) `InnifiedCircle`
e) These are all inner classes. Jeez.

In this code, for which inner class do we save a reference to an instance?

a) `CirclePanel`
b) `BigListener`
c) `SmallListener`
d) None of the above.

Do we need to save that reference? What if we delete lines 5 and 17, and make line 23

```
frame.add(BorderLayout.CENTER, new CirclePanel());
```

a) That's fine; everything will work.
b) This will run, but it won't behave properly.
c) This will crash when we try to run it.
d) This won't compile.

Thinking about the object-oriented structure of this code, we essentially have 3 kinds of classes we wrote: (1) `InnifiedCircle (IC)`; (2) `CirclePanel (CP)`; (3) the two `Listeners` (2L). Which of these is the best way to describe the relationship among these classes?

a) IC provides basic behavior; CP specializes that behavior; 2L provides additional behavior.
b) IC holds the application data; CP gives us a view of that data; 2L lets us change that data.
c) IC contains many variables; CP lets us access those variables; 2L lets us know when a variable changes.
d) IC creates objects; CP and 2L define the behavior of those objects.
e) IC, CP, and 2L work together to create a beautiful application.

The `InnifiedCircle` code is a few lines shorter than the first version (even after you remove the import statements). What's the best way of describing the reason?

a) I removed three methods from `CirclePanel`.
b) I added two classes.
c) I'm saving fewer references.
d) It's easier to share data among these classes.
e) It's easier to describe the desired behavior.

Hmm. What about encapsulation and inner classes?

a) Inner classes break encapsulation: all those private instance variables exposed!
b) Inner classes break encapsulation: the outer class can totally plunder the inner class!
c) Inner classes give *better* encapsulation; implementation details are completely hidden.
d) They're totally unrelated; as long as there aren't any new public instance variables, it's all good.

Which aspects of an “add”-ed component does a layout manager control?

a) Size and event-handling
b) Size and position
c) Position and event-handling
d) Size and color
e) Position and color

OK, another piece of code, and a sample application. The code printout for `SwingLine` only does layout; no event-handling. And it doesn’t include any of the inner classes the application needs. If we’re going to complete the application using inner classes (in a maximally object-oriented way), how many do we need to write? (Also, be prepared to give the names of any classes or interfaces we need to extend/implement in these inner classes.)

a) 3
b) 5
c) 7
d) 8
e) 9

Write those inner classes.

Let’s focus a bit more on Swing components. Which of the following are responsibilities of the `JTextField` class?

a) Store some text data
b) Display some text data
c) Respond to some keyboard entries
d) a and b
e) a, b, and c

How many different kinds of `String` comparison does the `String` class support (without any additional work from the programmer)? (Hint: check the API.)

a) 1
b) 2
c) 3
d) 4

The documentation for the `Collections.sort(List<T> list)` method says it sorts the elements of the list “according to the natural ordering of its elements.” What does “natural ordering” mean here?

a) The obvious ordering.
b) The ordering of these elements as they are found in nature.
c) The ordering specified by the elements’ `compareTo()` method.
d) The ordering specified by `Object.compareTo()`.
e) None of the above.

Consider the usual x-y graph, but restricted only to integer values. Call this a lattice. Obviously, points on this graph can only have integer coordinates, so consider this fragment of a `LatticePoint` class:

```
class LatticePoint {
     private int x;
     private int y;
     LatticePoint(int x, int y){ … }
     int distance(LatticePoint otherPoint) { … }
          // can only ‘travel’ horiz. or vert
}
```

Write a class that will allow us to sort a Collection of `LatticePoints` by their distance from (0,0).

Here’s a solution to the previous question. Change this to make this a class that will support us sorting points by their distance from *any* point, not just the origin. If you need to add code, indicate where it’s added; if you need to delete or change existing lines, indicate that.

```
1 class DistanceCompare implements Comparator<LatticePoint> {

2     public int compare(LatticePoint one, LatticePoint two) {
3          LatticePoint origin = new LatticePoint(0,0);
4          return (one.distance(origin) - two.distance(origin));
5     }
6 }
```

The book says we need to use a set instead of a list to prevent duplicates. But the `List` interface has a `contains()` method we could use (if our list already contains an element we want to add, then don’t add it). Why is using a set a better choice?

a) `List.contains()` doesn’t use the proper meaning of equality.
b) API code is more efficient than than programmer code.
c) `Set.addAll()` is more efficient than using `List.contains()`.
d) API code is more likely to be correct than programmer code.
e) All of the above.

What’s the fundamental purpose of the `hashCode()` method (properly implemented)?

a) It prevents object duplication.
b) It gives you a quick but rough indication of whether two objects are equal or not.
c) `HashSet` and similar classes use it to organize their internal storage.
d) Both a and b.
e) Both b and c.

Here’s an overriden `equals()` method for the `LatticePoint` class:

```
int equals(Object point) {
       return (x == ((LatticePoint) point).x &&
              y == ((LatticePoint) point).y);
}
```

Based on the book's discussion of hashcodes, which of these is the best return value for my overriden `int hashCode()` method?

a) `1`
b) `x + y`
c) `super.hashCode()`
d) `Math.random() * x`
e) None of these are wise choices.

From our perspective as API users, which of these are important differences between `TreeSet` and `HashSet`?

a) We have to override different methods to get our classes to work with them.
b) `TreeSet` keeps its elements in order; `HashSet` doesn't.
c) `TreeSet`'s internal implementation is much more complex than `HashSet`'s
d) Both a and b
e) Both b and c

Which of these is a title of a song written both by Van Halen and by Kris Kross? (Yes, we're talking about songs written before Java existed.)

a) Zombie
b) Rime of the Ancient Mariner
c) Jump
d) Hotel California
e) I Missed the Bus

For each of the applications below, choose whether the best primary collection is

a) a `List`
b) a `Set`
c) a `Map`

I. An address book (a collection of contacts)
II. Calculating statistics on a student assignment (a collection of scores)
III. A warehouse inventory (a collection of information about items in stock)
IV. Tracking event attendance (a collection of registered attendees)
V. Recording the objects that want to listen to a `JButton` (a collection of `ActionListeners`)