

CS 235 - Exam 2 Review Suggestions

last modified: 2015-11-04

- You are responsible for material covered in class sessions, lab exercises, and homeworks; but, here's a quick overview of especially important material.
 - Note that there is some redundancy below (some topics are "approached" from different directions in more than one section).
- You are permitted to bring into the exam a single piece of paper (8.5" by 11") on which you have **HANDWRITTEN** whatever you wish on one or both sides. This paper must include your name, it must be handwritten by you, and it will **NOT** be returned.
 - Other than this piece of paper, the exam is closed-note, closed-book, and closed-computer.
- This exam will be similar in style to Exam 1, although most of the questions will focus on "new" material (material covered since Exam 1). However, concepts from Exam 1 will still be involved -- we have necessarily been building on earlier material.
 - That said, note that, while buttons, basic `ActionListener` use, `try-catch` blocks, etc., were Exam 1 material, you have done much more with them since Exam 1. So, Exam 2's questions may involve these, so that you can demonstrate your now-higher level of mastery.
 - Likewise, because of the importance of becoming comfortable with the rich Java API, this is possible to show up again here (and on the final):

I could describe a package, some classes in that package, and/or methods/constructors and:

 - ask you declare instances of those classes,
 - write classes using those classes,
 - invoke methods of those classes, etc.
- remember that Java is **case-sensitive**; an answer may lose points if it uses the wrong case (if a class name does not start with an uppercase letter, for example, or if you start a variable name with an uppercase letter, or if you write a Java keyword using uppercase letter(s)).
- Java class coding standard questions are still fair game;
 - both "enforced" coding standards and "expected" coding standards;
 - you are expected to follow Java's **and** the class's coding standards on your exam responses, although an exam question may specify that an opening javadoc comment, for example, might not be required for that question's answer.
- note that, at this point, you should be able to look at a Java class (or collection of classes) and determine if it is an application or not,

...as well as being able to tell:

 - what it is a subclass of,
 - what interfaces (if any) it is implementing,

- what types its instances would be,
- what packages it is importing, and
- its visibility

Intro to Java layout managers

- You are expected to be very comfortable with `FlowLayout`, `BorderLayout`, `GridLayout`, and `BoxLayout`; you are expected to be reasonably comfortable with `GridBagLayout`, and using no layout manager (absolute positioning)
 - Which of these are provided by the AWT? which of these are provided by Swing?
 - You are expected to be able to read code using, and answer questions about, all six of the above; you are especially likely to be expected to write code involving `FlowLayout`, `BorderLayout`, `GridLayout`, and `BoxLayout`.
 - (If you are asked to write code involving `GridBagLayout` or absolute positioning, then example code that happens to make use of these will be provided in the exam examples packet.)
- You should be familiar with the features/functionality/advantages of all of the above-mentioned layout managers (and not using a layout manager); there could be questions involving these.
 - There could be general questions about layout managers in general, or what layout manager would be especially appropriate for use in a particular scenario, or fragments of code where you might be asked, for example, what layout is in use from the distinctive features in the code, or you might be shown a screen shot of a Java GUI and asked which layout was most likely used in different parts/panels/sub-panels of that GUI.
 - Given code, you should be able to determine what layout manager a particular panel or applet is using. You could also be asked to set the layout for a particular panel or applet to a particular layout manager.
 - (If you want a grid with columns and rows of equal sizes, the most appropriate layout manager to use would be...? What if you want columns and rows of different sizes, or where components take up more than one cell? What if you want components placed left-to-right, top-to-bottom as they are added to the container in question? etc.)
- Be comfortable with basic panel layout, using a panel with sub-panels using possibly-different layout managers to achieve desired layouts;
 - You might be given code using one or more layout managers, and asked to describe or sketch what you would see when the code is run;
 - You might be given a screen-shot of some container, and asked what layout manager was most likely used for that container;
 - Given a displayed or described desired layout, you should be able to write code that would result in that layout.
 - (and, yes, I will keep in mind `GridBagLayout`'s unpredictability in actual number of rows and columns displayed, since it infers those from what the user sets up...)

Intro to threads

- What is a thread? Why might it be useful? What are some of the things a thread might be used for in Java?
- You are responsible for knowing how to set up a thread using the `Runnable` interface; you should know how to create a thread using a class implementing the `Runnable` interface, how to start up such a thread, and how to (politely/in a non-deprecated fashion) cause that thread to terminate.
- (You are **not** responsible for synchronizing threads, although you **should** know that that possibility exists for Java threads, as does the possibility for communication between Java threads.)
- When can a thread be said to have terminated?
- What method must a class implementing the `Runnable` interface be sure to implement? When is this method called on a thread's behalf?
 - How do you write this method so that the thread it controls can be terminated in a non-deprecated fashion?
- Be comfortable with the static `Thread` method `sleep()`
- Why is Swing not thread-safe, in general? What are the basic rules, then, for using threads and Swing?
 - What are the two threads started up in a Swing GUI application?
 - It is considered bad form to sleep in one of the above two threads; which? What class is it recommended that one use instead of sleeping in that thread?
- You may have to write code involving threads, you will have to read code involving threads, and you will have to answer questions about threads.
 - What is a particular thread doing? How many threads are there? What are their names?
 - What happens in a particular thread when the application or applet starts up, or when a certain event occurs?, etc.
 - What do you see when a given application or applet involving threads starts up?
 - Be able to modify given thread-related code to meet different specifications; you might be asked to fill-in-the-blank to complete thread-related code.
 - Be able to read thread-related code, describe what it is doing, or what it might do in a given scenario.

Intro to simple graphics

- How do you paint/draw on a `JPanel`? What method do you implement? What is this method's parameter? How can you request that this method to be called on your container's behalf?
- What components is it appropriate to paint/draw upon? What components should you **not** try to paint/draw upon?
- What should be the first executable line of the method you use to paint/draw upon a `JPanel`? What

does this line do?

- Be able to set up desired colors and fonts, and draw strings, lines, and rectangles at a particular pixel location. Understand the basic coordinate system used for such pixel locations.
- You should also be comfortable with creating an `ImageIcon` instance from a `.jpg`, `.png`, or `.gif` image, and then painting that `ImageIcon` instance onto a panel;
- If you were given an unfamiliar `Graphics` class method's Java API description, you should be able to write expressions or statements using it appropriately.

Intro to simple animation in Java

- How can one use threads and graphics to implement simple animation in Java, and use buttons to start and stop such animation?

More components and listeners

- You should be able to read/write code using the discussed new components and listeners, especially those both discussed in lecture and practiced on a homework.
 - And, if I were to give you part of the Java API information for another Java Swing component, you should be able to use your experience along with the API information to be able to make appropriate use of that component.
- You should be able to choose which component(s) -- both from among these and among those covered earlier in the semester -- would be most appropriate for different situations; you should be able to defend such choices as well.
 - ...and which listener(s) would be appropriate for different situations.
- You are especially responsible for `ActionListener`; you are also responsible (but to a lesser extent) for `ItemListener` and `AdjustmentListener`.
- What do you need to do to create active, sensitive versions of these new components? How can you find out about the state of such components when, say, a button is clicked?
 - Which interface should be implemented?
 - What kind of listener should be added, and to what?
 - Which method(s) must be implemented for each listener?
 - What component methods are useful when handling such an event on that component?
- Given code, you might be asked:
 - What is the effect of parts of this code or of a user action on one of these components?
 - What happens (or does not happen), and why (or why not)?
 - ...to modify the code so that it does do something specified (e.g., correcting a problem, adding particular functionality, etc.)
- You could be given a scenario, and asked which component(s) and/or listener(s) are most appropriate

for use in that scenario, and why.

- You could be given a display of a frame or panel, and asked what kinds of components are within that/those containers. You could be asked to write code that would produce such a (depicted or described) display.

JCheckBox, ItemListener

- What are some kinds of situations where JCheckBoxes might be appropriate?
- When might one want to make a JCheckBox sensitive to ItemEvents? to ActionEvents? when might you not make it sensitive at all (and check its state when some other component is acted upon, such as a submit JButton?)

JRadioButton, ButtonGroup

- How does a JRadioButton differ from a JCheckBox?
- How do you set up a collection of JRadioButtons so that they behave as classically expected?

JComboBox

- Why is the "combo" in its name?
- When might a JComboBox be preferred to, say, a collection of radio buttons?

JMenuBar, JMenu, JMenuItem

- understand the basics for setting up menu items in a menu in a menu bar
- which container can have a JMenuBar added to it?

JScrollbar, AdjustmentListener

- Understand when a scrollbar is useful/its benefits, and what kind of listener one might add to it.