CS 235 - Exam 2 Review Suggestions

last modified: 2021-11-08

- You can receive (a maximum) ***5 POINTS BONUS*** on Exam 2 if you do the following:
  - Make a **hand-written** Exam 2 study sheet.
  - Submit a photo or scan of it saved as a .pdf, .png, .jpg, or .tiff to Canvas **by 3:00 pm on Friday, November 12** such that I can read at least some significant CS 235 Exam 2 material on it
  - Please let me know if you have any questions about this, and I hope it helps you in reviewing course concepts more effectively before Exam 2.
  - You are **encouraged** to have this available as you are taking Exam 2.
- Exam 2 will be given in Canvas.
  - It will be **available from 3:00 - 5:00 pm on Friday, November 12**.
  - It will be set up such that **you may only attempt Exam 2 once**.
  - That said, remember that **I will be in BSS 317 and also available in the regular Zoom CS 235 session from 3:00 - 5:00 pm on Friday, November 12.**
    - If you are in BSS 317, you can ask me questions directly during Exam 2. If you are not, you can ask me questions via Zoom during Exam 2.
    - This Zoom session will be set up so all microphones are muted, and you will only be able to chat with me.
    - **You are encouraged to LET ME KNOW about technical difficulties, so we can make provisions as needed!**
    - **And, I will have a small displayed window in the Zoom session with any typos or announcements that have come up so far during Exam 2.**
  - Exam 2 will be set up such that **you will be shown one question at a time**,

    BUT there will be a list of question-links on the right-hand-side of the Canvas screen, and you can go back and forth between questions during that **one** exam attempt.
  - You are expected to work **individually** on the exam -- it is not acceptable during the exam to discuss anything on the exam with anyone else.
  - You may look up information from your Exam 2 study sheet, on-line, or from the course textbook during the exam, but note that if you take too long looking up material, you may have trouble completing the exam during the time period.
  - I expect there will be a few multiple-choice questions, and the rest will be short-answer questions.
    - BUT given the Canvas quiz question options, for those short-answer questions, you will be shown a larger area for your answer than you will need!
    - You will be reading and writing code, statements, and expressions in this format. There will

be questions about concepts as well.

- You could be asked to write Java expressions, statements, fragments, methods, or up to and including entire Java classes and applications; read the questions carefully, so you do not give more answer than you have to.
- You may be asked questions **about** Java, or about various aspects of Java.
- You could be asked what a Java fragment does or means; you could be given an expression or fragment or class, and asked its value or what it does or what it would output in a given situation.
- You could be asked to modify an expression or fragment or class, or to correct an expression or fragment or class, as well.
- You might be asked to complete incomplete code (you could be given partial code, and asked to complete or modify or debug it in some way).

- A link to a packet of references and additional instructions - intended for use with Exam 2 - will be posted from 3:00 to 5:00 pm on the course Canvas site, linked from the Exam 2 Instructions.
  - So, you can have it open in another browser window while you are taking Exam 2.
  - This is intended both for reference and for use directly in some exam questions.
  - For exam purposes, the usual comments will be removed -- however, unless clearly indicated to the contrary, you can safely assume that this example code is otherwise working, correct Java.
- Your studying should include careful study of posted examples and notes as well as the lab exercises and homeworks thus far.
- You are responsible for material covered in class sessions, lab exercises, and homeworks; but, here's a quick overview of especially important material.
- You are responsible for the material covered through the end of Week 11 (Friday, November 5th), and through and including the Week 11 Lab Exercise and Homework 7.
- 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 2 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.
- Also still fair game: 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

- You also are still responsible for knowing the Java conventions discussed and also the CS 235 course Java style standards. You are also expected to follow these in your answers (**including indentation**).
  - You should use the **Formatting Practice Question** linked from the course Canvas site's Home page to practice writing formatted Java statements and fragments BEFORE Exam 1!
  - If you find you are having trouble with this, make sure to come by virtual student hours or ASK ME before Exam 2, so you won't lose points for poorly-indented answers.
  - (Yes, you will be dinged for { } not placed according to the CS 235 course style standards. The required style for these has been demonstrated in all of the course examples, and mentioned in the "Important Notes" sections on Homeworks 1, 2, 3, 4, 5, and 6.)
  - Also 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)).
  - But, an exam question *may* specify that an opening javadoc-style comment is not required for *that* particular question's answer, for exam purposes.

Intro to Java layout managers

- You are expected to be very comfortable with `FlowLayout`, `BorderLayout`, and `GridLayout`.
  - You are expected to be able to read code using, and answer questions about, all three of the above.
- You should be familiar with the features/functionality/advantages of all of the above-mentioned layout managers; 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 is using. You could also be asked to set the layout for a particular panel 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 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 what you would see when the code it 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.

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?
  - But -- that said -- because `Runnable` is a so-callled functional interface, you should be able to write a lambda expression to implement a `Runnable` instance, and understand that the anonymous method there is assumed to be implementing that `Runnable`-interface-required method...!
  - How do you write this method -- whether using a lambda expression or writing a subclass implementing the `Runnable` interface -- so that the thread it controls can be terminated in a non-deprecated fashion?
- Be comfortable with the static `Thread` method `sleep()`
- We didn't discuss it sufficiently for me to feel it is reasonable to ask detailed questions about it -- but you should KNOW that there are issues related to Swing and thread-safety. For example:
  - You should know that this is why we create a GUI application's frame the way that we do (using the `EventQueue` object's `invokeLater` method, with a lambda expression argument creating the frame).
  - You should know to look up what is considered safe and not if you decide to create threads in an applicaton and those threads may muck with Swing components (so you can double-check which Java Swing methods are in the small subset that *are* indeed thread-safe, such as, happily. `repaint` and `setText`).
- 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 starts up, or when a certain event occurs?, etc.
- What do you see when a given application 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 in Java

- 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 may you want to 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 starting 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 at a particular pixel location.
- 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 lab exercise.
    - (But I am not including `JScrollBar` and `AdjustmentListener` on Exam 2, because the Week 11 Lab Exercise was long and I don't think all the lab teams had a chance to try it out/discuss it. It will be fair game for the Final Exam.)
    - However, 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 are also responsible (but to a lesser extent) for `ItemListener`.
- 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?
    - Alternatively, because the listeners we have discussed so far have been so-called functional interfaces, how can you use a lambda expression to directly create an instance of that interface such that the anonymous method there is assumed to be implementing that interface's required method?
    - 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 `JCheckBox`es 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?