

CS 235 - Final Exam Review Suggestions

last modified: 2015-12-09

- This Final Exam is **comprehensive** -- so, you should use the Exam 1 and Exam 2 review suggestions (as well as Exam 1 and Exam 2) as part of your studying. You are responsible for material covered in class sessions, lab exercises, and homeworks for the entire semester; but, here's a quick overview of especially important material since Exam 2.
 - 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.
 - (Note that final exams are not returned, although you can come by my office after they are graded to look at yours, if you would like. I'll keep them on file for at least two years.)
- General style of the final exam will be similar to Exams 1 and 2, **except** that the final exam will be **comprehensive**, and will deliberately encompass material from the entire semester.
 - (Thus, it would be very wise to understand anything that you missed on Exams 1 and 2; you've got those tests to study from, and you have the review suggestion sheets for Exam 1 and Exam 2, still available from the course web page under "Homeworks and Handouts".)
 - The final exam will, as usual, include a separate "packet" of example code, some to be used directly in questions, and some to be used as reference if you know how to read Java code as you should at this point.
 - The final exam will include a combination of short answer questions, code-fragment reading and writing, and whole-class reading and (possibly) writing. Classes may be applications (or not), and may or may not include auxiliary classes and inner classes.
- 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 write a Java keyword using all-uppercase letters, or if you start a method or instance with an uppercase letter).
- Because of the importance of being able to use the rich Java API, note that I could describe a package, a class in a package, and/or one or more methods/constructors of a class as the Java API does, and:
 - ask you declare instances of those classes,
 - write classes or code fragments using those classes,
 - invoke methods of those classes, etc.
- 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 JDBC

- You are responsible for those JDBC features that have been discussed in lecture and in lab, as well as for those JDBC features that have been used in posted course examples and in course assignments.
- What package do you need to import within a Java class that is going to make use of JDBC?
- Need to be comfortable with the basic steps required to connect to a database and run SQL queries from Java.
 - what object(s) do you need to be sure to `CLOSE` when you are done?
- Need to be able to tell what JDBC code would do; may need to be able to modify it, debug it, or complete it.
- should know when and how to use the methods `executeQuery` and `executeUpdate`;
 - given a desired SQL statement, you should be able to say which of `executeQuery` or `executeUpdate` would be more appropriate for executing it, how you could execute it, and what that call would return.
- in addition to the `Statement` class, you should also be comfortable with `PreparedStatement`
 - what is the difference between how you set up and execute a `PreparedStatement` as compared to a `Statement`? What must be done before each call of a `PreparedStatement`?
 - should know two major advantages of using a `PreparedStatement` over a `Statement`. When would you choose to use a `PreparedStatement` instead? When would it not matter so much?

intro to JUnit

- We discussed unit testing in Java -- you should be able to write a simple unit test method for a JUnit test class; you should know what to `import` for such a JUnit test class.

- Given a method's purpose, you should be able to write a testing method that could appear in a JUnit test class would pass if that method were written correctly, such that that testing method would be run automatically in DrJava if both the class being tested and the `TestCase` subclass were open (and saved and compiled) and you click the "Test" button.
- You should be able to understand/read/write appropriate `assertEquals` and `assertTrue` method calls.
- What features must a method have within a JUnit test class so that such a method will be run when you click the "Test" button in DrJava?
- what is special about the method `setUp` within a JUnit test class? When will this be called? What might you use it for? You should be able to understand/read/predict the behavior of a JUnit test class including these methods.
- Given a simple Java class and a JUnit test class testing that simple Java class, you should be able to reason about what would happen if both were open, saved, and compiled in DrJava and the "Test" button is clicked; this might include describing what would appear in the DrJava "Test Output" window, and what would appear in the DrJava Interactions window.

Simple writing to and reading from files

- Know the packages containing the classes that we used for I/O and file I/O
- What abstraction does Java I/O follow (most commonly)?
- You should be comfortable using `Scanner` for simple interactive input and for file input as well.
- What package should you `import`, for:
 - ...the `File` class?
 - ...the `Scanner` class?
 - ...the `PrintWriter` class?
- You should know a bit about `java.io`'s `File` class.
 - What does a `File` instance correspond to?
 - What does the `File` class help to hide/abstract?
 - What kinds of capabilities are provided by the `File` class? What are some of the methods that the `File` class provides? (and what, notably, is NOT part of this class' capabilities?)
 - How is a `File` instance instantiated?
- You should be able to read and write code involving character-based file I/O using `Scanner` for file input and `PrintWriter` for file output.
- What is a stream? How can one use the `Scanner` class to set up an input stream from a file? How can one use `PrintWriter` to set up an output stream to a file?
 - How do you close a stream?
- What are some of the commonly-used methods for file input and file output using the classes we

discussed?

Intro to generic collections (especially `ArrayList<E>`)

- In what Java package do you find `ArrayList<E>`, `LinkedList<E>` -- indeed, the `Collection<E>` and `Map<K, V>` interfaces, and many implementations of these interfaces?
- what does the `<E>` indicate, anyway? (What makes these generics? Why is that useful?)
 - What type could you make a collection of if you'd like to store a variety of different class instances within a single container?

(Side-note for your future reference: there is a somewhat persnickety operator, `instanceof`, that expects an object and a class as its operands, and should return true if that object is an instance of that class...)
- What are some basic operations we discussed that are included in the `ArrayList<E>` generic class? Note that many of these are also available in other collections in `java.util`.
 - You should be comfortable reading, writing, or answering questions about these methods.
 - A number of these methods are actually part of the `Collection<E>` interface that `ArrayList<E>` implements; you should thus understand that any class implementing `Collection<E>` would implement these methods as well.
- You should be able to name at least one generic collection class provided by Java.
- You should be able to use a so-called `for each` loop to iterate through a collection such as an `ArrayList<E>`
- You should be reasonably comfortable with the `ArrayList<E>` class.
 - And, if I were to give you part of the Java API information for another Java collection class, you should be able to use your `ArrayList<E>` experience along with the API information to be able to perform actions on an instance of that class.
- What are some of the advantages of an `ArrayList<E>` as compared to an array? What are some of the disadvantages of an `ArrayList<E>` as compared to an array? What are some of the differences between these?
 - What can you do more easily (at least, from the programmer's point of view!) with an `ArrayList<E>` than with an array? ...with an array than with an `ArrayList<E>`?
 - When might you use one instead of the other?
 - Given a particular scenario, you should be able to reason about and discuss which might be better to use, and why/why not.
 - I could ask you to declare, instantiate, and use both `ArrayList<E>` instances and arrays, to see if you can, and if you know the differences between them, etc.
 - If I gave you code (or code fragments) using both `ArrayList<E>` instances and arrays -- you should be able to tell which is which.
- You may have to write, read, answer questions about `ArrayList<E>` instances.

- I could give you a code fragment with a number of `ArrayList<E>` method calls, and you should be able to draw/give the contents of such a list instance at various stages of such calls, as well as its final contents after a sequence of such calls.
- Expect to have to write code to iterate through an `ArrayList<E>` instance; you need to be able to use a `for each` loop to do so, but there are also at least two other means for doing so.

intro to packages and jars

- What is the purpose of a Java package? How do you indicate that a class is part of a package?
 - How can another Java class make use of something in a package?
 - Given a package name, where must the classes within that package reside?
- What is the `CLASSPATH`? What is its purpose? How does Java make use of it? How can you use `-classpath` with the `javac` and `java` commands?
- What is a `jar` file? Why is it useful? What can be in a `jar` file? How can you create a `jar` file? How can you use a `jar` file?