CSCI209 2nd Exam Prep

Content Covered

Packages
- Purpose, usage

Access Modifiers

Inheritance
- final methods and classes
- Polymorphism, dynamic dispatch
- Abstract classes

Interfaces

Jar Files

Class path

Collections Framework
- Common interfaces, implementations
- Generics
- How/when to use

Exceptions
- Types of exceptions
- Handling exceptions
- Benefits, limitations

Eclipse

Files

Streams
- Classifications of stream classes
- How to use
- Justification for design

~~**Garbage Collection** –~~ cut for time (but check out the slides for funzies!)

Similarities, differences between Java and Python
- Compiling vs Interpreting
  - Compiler optimizations

Software Development Models

- Waterfall
- Iterative/spiral
- Test-driven development

Testing

- Different levels of testing (unit, integration, system, …)
- Black-box testing vs. White-box testing
- Code coverage
    - Coverage criteria
    - Strengths and limitations
- JUnit testing framework
- How to write good unit tests

Design Principles - naming principles we've applied

- Design goals
- Open-closed principle
    - Open to extension, closed to modification
- DRY (Don't repeat yourself)
- Don't repeat yourself (DRY)
- Shy code (avoiding coupling)
- Tell the Other Guy (Tell, Don't Ask)
- Single Responsibility Principle
- Program to Interfaces/Dependency Inversion Principle. (Depending on interfaces/abstraction is inverted from the way we tend to think.)
- Appropriate solutions
- Applications of the principles to scenarios/examples from class
- Justifying solutions using appropriate terminology and design principles
- DRY (Don't repeat yourself)

Code smells

- What are they? How are they used?
- Example: Duplicate code
- What are examples of code smells? Why are they problems? How do you fix them?

Refactoring

- Resolving code smells using abstraction

Design patterns

- Composition (and why used instead of inheritance)

Exam Structure

- In-class
- One letter-size page of notes for reference
- Sections: very short answer, short answer, applied/coding

What I expect from you on the exam

- To know Java/OO-programming/design terminology
    - Need to be able to communicate with others about your design/implementation
- To focus on the most important/distinguishing characteristics/traits
    - Example: if there is a 4-point question is "What is Java? Explain a benefit of Java.", do not start copying all of the content from any slides you can find about Java. You should know the highlights and be able to summarize what is Java, e.g., "Java is a compiled, statically typed programming language." Followed by "Benefit: Java is portable because it is compiled into bytecode, which is executed by a JVM. Java bytecode can be executed by any machine that has a JVM."
    - Example: if the question is "What are the benefits of using arrays?", do not answer with any variation of "Because arrays are beneficial" or "Because arrays store data." Tell me *why* arrays are good (e.g., "arrays allow us to group data of the same type together, which makes passing them to methods and processing larger amounts of data easier.") You want to make clear that you don't "just" know the terms but that you also understand their meaning and implications.
    - Example: When I ask about the *limitations* of something, that means that, even if that something is implemented perfectly, it still has limits. (It doesn't mean that the approach is "bad" or "wrong"—just that there are limits to its scope/possibilities/potential/reach.) What are those limits?
- To synthesize the information in your answer into your own words while still being concise and precise.
    - You should not simply copy information from a slide; often answers come from multiple slides
- To design a solution and be able to defend it
- To be able to read and understand Java programs, with or without documentation
- To be able to write a program (given an algorithm or creating your own algorithm, given a problem) or class
    - Syntax must be very close to correct (correct keywords, punctuation, special characters, variable naming, operations)
- To be able to recall the information without looking up every (or many) questions. If you need to look up answers, you will not complete the exam in time. You should be well-prepared for the exam before starting.
- To consider the point value of the question. If the question is worth 4 points and is an "essay" question, I'm only looking for a few sentences/bullet points, but make sure you're answering each part of the question clearly and precisely.

- To keep an eye on the time remaining.
- To know the syntax for variable declarations/definitions, if statements, for loops, methods, class definitions, print statements; that the `Object` class defines the methods `equals` and `toString`; basically, all the things that we've seen/written many times over.
- Make it clear that you truly understand the material and are not just using some terms and hope they're in the right order. Think of this like if you were answering an interviewer's questions. They don't know what you "should" know.
- Be explicit in connecting the dots: if you state that X is a benefit, you then should say how/why that is a benefit and how it is accomplished. Answers often don't have to be long. They just need to address each question.
    - Answer a question with multiple questions in order. Don't make me hunt for your answers.
- Recently, students have included in their answers material that we have not covered. You should only include material that was covered in class.

What I do NOT expect from you on the exam

- To know the API for all Java classes that we have covered/used.
- Perfect essays, complete sentences
    - Example: if the question is, "What are the benefits of using arrays?", I do not expect you to answer with, "The benefits of using arrays are the following: …" I just want to know about what follows "the following".
- To write comments for code unless specifically requested
    - However, write comments if they help you to organize your thoughts.
- To compile your code/solutions

Suggestions on how to prepare

- Exam is ***terminology heavy***. Make sure you know the terminology (much of it is in the list above). I hope that, with all the practice we've done with using and reviewing terminology, your knowledge of the terminology should be close to automatic.
- Read through slides for vocabulary, review questions, exercises.
- Think about the designs we have discussed in various situations/assignments and what the tradeoffs are to each design.
- Practice reading through programs (e.g., your assignments), tracing through them, and saying what the output/behavior should be.