213 Course Content Summary (Fall 2024)

This summary highlights what material is and is not testable. Many questions may rely on you understanding and applying this knowledge; it is not sufficient to memorize this list, you must be able to use the material. Some items may have been specifically mentioned in class as being testable and may not be mentioned here.

- Most topics listed refer to the slide headings.
- "Know" means have memorized.
    - "Know the days of the week" means have memorized Monday, Tuesday, ...; and "understand" each of them.
    - You don't need to memorize anything word-for-word: know it in your own words.
- "Understand" means: once told the topic (or items), be able to talk about it.
    - "Understand all flavours of ice-cream" means given a flavour (say chocolate mint chip), be able to describe it; compare (similarities) and contrast (differences) it to other flavours. But, don't memorize all the flavour names (not asked "List all 31 flavours").
- Do not need to know:
    - Import statements for any class or interface.

1. Lecture content

Before Midterm

0 - Admin

- Review expectations and consequences for academic honesty.
  Be warned: I am **passionate** about it!
- Know the three components of the course
- Nothing testable.

1 - Intro to Java

- Ideas from the pre-recorded content
    - Know Java & OOD terminology/ideas:
        - field, method, public, private, constructor
        - accessor, mutators, classes, objects, instantiating objects, object references.
        - basics idea of a class's package
        - main(), JavaDoc (for class-level comment),
        - basic exceptions
    - Know Java primitive types, promotion, demotion, casting, constants.
    - Know Java's memory management (basic idea of garbage collection)
    - Know how method parameters work with primitive and object types (pass by....)
    - Know how to work with a Java array: create, access, get size, pass to method.
    - Know ArrayList: how to create, add, remove, get size, get element
    - Know for-each loop structure.
    - Know String: create, access a character, concatenation, operator overloading, get length, equals() vs ==.
    - Know how to parse a string to an int.
    - Know how to print to the screen, and read form the keyboard (Scanner).
        - Understand reading complete lines and working with whitespace.

        - Understand closing a scanner.
    - Understand files, and working with scanner on a file.
    - Know basic try/catch and exceptions.
- Java Review (covered in class)
    - one name per item (argument matches fields)
    - Pass by value
    - Multiple reference
    - Strings and immutability
    - List, ArrayList, wrapper classes
- Know static
    - Know static methods and static fields, access rules
    - Know what is a static factory method, and how to use one.
- Know what is clean code, and understand use of a coding standard/style guide.
    - Do not need to memorize the course's coding style guide, but should know the basics of what is expected in terms of writing clean code.

2 - Anon Classes

- If given an interface, know how to instantiate anonymous classes/objects and use those, such as how the Strategy pattern is used.
- Know how to print to the screen (i.e., System.out) using print(), println(), and printf()
    - Know conversion specifiers: %d, %x, %f, %s, %b, %n
    - Know formatting with columns, and with commas.
- Understand the wrapper classes for primitives; know Integer.
- `File`
    - Know purpose of File class and understand methods shown in slides.
    - Know what a Java interface is.
    - Understand how to use the `FileFilter` interface.
        - Able to use a named object, or an anonymous object for `listFiles()`.
        - Know the syntax for each option.
- `Comparable`
    - Know the purpose of the Comparable interface.
    - If given the interface, able to use it with `Collections.sort()` using named/anonymous objects/classes
- `Comparator`
    - Know the purpose of the `Comparator` interface.
    - If given the interface, able to use it with `Collections.sort()` using named/anonymous objects/classes
- Know anon classes and anon objects: If given the interface, be able to apply it by creating a named class, or an anonymous class; a named object or an anon object.
- Understand the Strategy pattern as it applies to examples we saw in lecture.

3 - OOD Design Process

- Know terminology: OOD, OOP, Domain
- Understand phases: Requirements, Design & Implementation, Verification, Evolution
    - Know goal, process / products for each of requirements, design, and implementation
    - Know why design is a “*wicked*”, “*sloppy*”, and “*heuristic*” process.

        - Know what *skeleton* code is. Know *continuous* vs *big-bang* integration.
    - Class Design
        - Know terms: class, object, state, behaviour, identity, instance
        - Able to identify classes via finding nouns, identifying utility classes, systems, and agents.
        - Understand when to, and when not to use String
        - Know enums (able to create, use, and explain benefits)
        - Able to assign responsibilities to classes (verbs)
            - Able to work through tradeoffs of assigning functionality into classes.
    - Class Relationships
        - Know dependency, how to explain it, how to identify it. Know coupling.
        - Know aggregation, how to explain it, how to identify it.
        - Know inheritance.

4 - OOD Design Techniques

- CRC Cards
    - Know their full name, their purpose, and how to create them, where they fit into development process.
    - Know their limitation.
- UML Class Diagram
    - Know its purpose.
    - Know how to draw one including:
        - 3 sections, methods, fields
        - relationships: dependency, aggregation, inheritance (class “is-a” and interface “implements”)
        - how to indicate an interface
        - how to indicate the multiplicity of aggregation connections (Ex: 1, 0..1, *)
        - add a comment to a diagram
        - show objects
    - Know how to provide different levels of detail in a UML class diagram such as:
        - Just the class names and relationships
        - Class name with field and method names with visibility modifiers
        - All names, relationships, and type information.

5 - Lambda & Streams

- Know Inner classes
    - Know how to work with an inner class, what it can access and why.
    - Know how to use lambda expressions, and why they are (sometimes) better.
- Know method reference; how to use them
    - Able to work with inner classes, lambda expressions, and method references interchangeably.
- Know streams
    - What they do; how to use them.
    - Know terminology of source stream, intermediate operation, terminal operation
        - Able to use intermediate operations: `map()`, `filter()`, `sorted()`
        - Able to use terminal operations: `forEach()`, `count()`, `collect()`, `reduce()`
    - Understand: `IntStream`, `LongStream`, `DoubleStream`; `mapToDouble(), ….`

    - Understand fluent API

6 - Generics

- Know generics are a form of compile-time polymorphism (parametric)
- Know how to make a *method*, a *class*, or an *interface* generic
- Know the purpose of the type parameter
- Arrays and Generics / Covariant vs invariant
    - Know you cannot create an array of anything generics
    - Know covariant vs invariant; which of arrays or lists are covariant and invariant. Know what this means in practice (able to recognize how they work in code).
    - Know what Java does to handle type errors with covariant vs invariant types. Which are handled at compile time vs which are handled at runtime.
    - Know what type erasure is.

7 - Class Design

- Able to consider class design alternatives
    - Understand ideas behind the three different Day class implementations.
    - Know what deprecated means
- Encapsulation
    - Know benefit of encapsulation, able to give a definition for it.
    - Able to recognize when it's violated.
    - Know immutable: how to recognize it, how to code it, benefit of it.
        - Understand shared reference problem, and why clone() can be used to solve it.
        - Know the difference between final and immutable: how are they similar? How are they different?
    - Know the meaning of a final variable, and what it prevents from changing.
- Understand and able to apply the command/query separation guideline.
- Know iterators, their purpose, and be able to use them.
    - Able to explain how it violates command/query guideline. Understand how this can lead to problems.
    - Know what a side effect is.
- Know purpose and use of `Iterable`
    - If given the interface, able to create and use an `Iterable` class.
    - Able to create functions in a class which return an anonymous `Iterable` object; understand why you might want to.
    - If given the `Collections.unmodifiableCollection()` prototype, able to make and use an unmodifiable collection; know why we'd want to.
    - If given the `Iterator` interface, able to create a simple custom iterator.
    - Understand class relationships for a system using `Iterable` and `Iterator`

8 - Interface Quality

- Know two perspectives of looking at a class.
- Know the four ways of analyzing public interface quality:
    - Know cohesion.
        - Know the single responsibility principle
        - Able to apply these ideas to improve a class design.
    - Know completeness (convenience).

    - Able to recognize when an interface is incomplete.
  - Know clarity: intention revealing names, meaningful abstractions.
  - Know consistency: able to identify and correct inconsistent features of an interface.
- Know the 5 “other ways to review quality”.
- Able to use these ideas to critique code.

9 - Contract vs Defensive

- Know programming by contract
  - Able to give a definition for it, recognize it, and explain it.
  - Know how it assigns responsibility for correctness,
  - Know what are preconditions, postconditions, invariants
- Know defensive programming
  - Know how it assigns responsibility for maintaining a consistent state in an object,
  - Know asserts and how to use them for defensive programming.
- Understand all the stated ways of handling errors.
- Know asserts
  - What they are, how to write them
  - Know when to use them, and when not to use them.
  - Understand how to enable them in the JVM
  - Understand the limitations of asserts

10 - Interface Polymorphism

- Know what an interface is, how to write one, how to use one.
  - Know what a concrete type is.
  - What can go into an interface.
  - Know @Override and how to use it.
- Know polymorphism and how to use it.
  - Know 3 types of polymorphism: ad-hoc, parametric, subtype. Know which are compile-time vs run-time in Java.
  - Know benefit of using it.
  - Know what late binding is, and how it gives lose coupling.
- What is an idiom
  - How to make a function instantiate an anonymous class.
- Know the interface segregation (narrow interface) design principle.
- Know how to have code depend on an interface vs on a concrete class (design principle).

END OF MIDTERM MATERIAL

(Slides #11 - Inheritance will not be on the midterm).

Material After Midterm

11 - Inheritance

- Know inheritance and the relationship it creates.
  - Able to read/draw UML diagrams with it.
  - Able to apply it to an example, and explain/apply/recognize polymorphism with inheritance.
- Know the uses of `super` in constructors and method calls; contrast it with `this`.
- Know overriding and the access derived classes get to the base class.
  - Know `final` when applied to a method.
  - Know who is MC Hammer. :)
  - Know shadow variables.
- Know what is a class hierarchy.
  - Know single vs multiple inheritance, and which Java supports.
  - Understand programming to an interface.
  - Know how to apply the design principle of preferring composition over inheritance
- Know abstract classes & methods
  - Able to read, understand, write and explain code using abstract methods and classes
  - Able to compare (similarities) and contrast (differences) between abstract classes and interfaces.
  - Understand the ideas presented on the “Abstract Questions” slide. You don't need to memorize these answers.
- Know Java visibility modifiers and the access rules for each.
  - Know when to use protected, and how to avoid it.

12 - REST Introduction

- Understand front-end vs back-end; role browser plays.
- Understand HTTP:
  - URLs, ports, HTTP vs HTTPS
  - HTTP Methods: get, post, delete, put
  - Understand HTTP status codes (200, 201, 4xx, 500)
  - Know 4 ways to pass values to server
  - Know query strings
- REST API
  - Know acronyms for REST, API, and HTTP
  - Understand structure of URLs in a REST API;
  - Able to design REST API for simple problem. Understand the role of things (nouns) and actions (verbs) in API design.
  - Know what CRUD stands for.
  - Understand properties of RESTful APIs (uniform interface, stateless, cacheable)
  - Know JSON encoding of objects, arrays, strings, integers.

13 - Spring Boot

- Know what dependency injection is and its benefits.
  - Understand Spring Boot's use of DI
    (You do not need to be able to implement DI using SpringBoot)
  - Know the benefit of using DI, and of using a DI framework.
- Controller:
  - Know what a controller does in a REST API architecture.
  - If given an empty controller class, able to:
    - add new end points for GET and POST
    - pass values in via path variables
    - pass values in via the POST body
    - pass values in via GET query string
- Response Codes
  - Know the purpose of returning a response code.
  - Understand how to set the response code for a specific endpoint.
  - Understand how to associate a response code and custom message with an exception. (Don't memorize how to code it, just fully understand the code).

14 - Inheritance Design

- Know how encapsulation works with inheritance.
- Know the requirements for inheritance; able to discuss each one listed in class.
  - Know the LSP, how to apply it.
    - Know how it applies to examples discussed, how immutability affects it.
    - Do not need to know all of the "SOLID" principles; just ones covered in course.
  - Know how a reference can be used instead of inheritance: benefits and drawbacks.
- Know self-use, what it is, why it's a problem for inheritance, how to fix.
- Implementation inheritance vs interface inheritance
  - Use of a wrapper object with forwarding / delegation

15 - Patterns

- Know what a pattern is.
  - Understand how a pattern is described (such as in the gang-of-four book).
- Know iterator pattern:
  - Know it's purpose & benefit over other ways of achieving the same functionality.
  - How to use an iterator in the client, how to write your own iterator (for say an array).
  - Know the UML for iterator pattern, able to apply it to a specific example if given the classes/code.
- Know the observer pattern:
  - Know its purpose, advantages, examples, and UML.
  - Know how to write code to user/create observers.
- Know model-view: purpose of each, idea of separating them, how observer fits in.
- Know the facade pattern
- Know how to recognize a pattern.

16 - Exception

- Know try-catch-finally blocks and flow of execution and exception propagation.
- Understand exception hierarchy.
- Know checked vs unchecked exceptions and the difference. Know which reduces coupling.
- Know how to create custom exceptions.
- Know how to create clean exception handling code using multiple methods.
- Know Java 7's exception features: try-with-resources.
- Able to clean up complicated exception code with `foo()` / `fooThrows()`

17 - Smells

- Know smells: duplicate code, long methods, needing comments
  - How to refactor to solve each smell.
  - Understand how to extract a class to cleanup complicated algorithms/code.
- Know DRY
  - How to fix duplicate code in a single class, sibling classes.
- Know the Template Method pattern
  - When to apply it, how to apply it, what it's UML looks like.
  - Comparison to Strategy Pattern
  - Able to solve duplicate code using either template method or strategy patterns; able to write "client" code as well.

18 - Object

- Know the Object class
- Know `toString()` and how to effectively use a base class implementation.
  - Know its use (for an obvious canonical form vs formatting complex object)
- Know `equals()`: When it's used, how to correctly implement it
  - Know what `instanceof` does for us
  - Know what `Objects.equals()`, and `Double.compare()` do.
  - Understand the equals() contract
- Know `hashCode()`: how to implement it, how to use `hashCode` on fields.
  - Know how to use `Objects.hash()`
  - Able to explain why `equals()` must use at least all the same fields as `hashCode()` uses.
- Understand inheritance with equals() and hashCode()

19 - Threads

(Only as much as was covered during lecture)

- Know what threads are, and why they are useful.
  - Understand how a thread is created and executed.
  - Understand the timing for threads, and how to suspend a thread.
    - Know terms: time slice, starvation, fairness, priority.
- Know what a race condition is
  - Understand the explanation of the race cases presented in lecture, able to apply logic to similar problems.
  - Know what a Heisenbug is, know what thread-safe means.

- Understand locks
  - Understand producer/consumer example: why it fails without locks.
  - Understand deadlock and dining philosophers problem.
  - Understand how to interrupt a thread.