213 Course Content Summary (Spring 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, (OOPS optional :))
- 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 more on chap7
  - 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 do the stream exercise
- Generics (#5.1)
  - 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

6 - (there is no #6)

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 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).

