15-214 Course Review Sheet

Fall 2013

Object-Oriented Programming

- **Dynamic Dispatch** - determining which methods are called based on the runtime class of an object
- **Encapsulation**
- **Inheritance**
- **Subtyping** - the rules and how it differs from inheritance
- **Behavioral Subtyping** - and recognizing deviations from it
- **Composition** - e.g. in contrast to inheritance
- **Immutable Objects** - understand the difference beteen immutable and mutable objects

More terminology:

- receiver
- member function / method
- base class / superclass
- derived class / subclass

Java

Understand the following keywords and concepts in Java:

- access modifiers (`public`, `protected`, `private`, and default/package protection)
- `abstract` classes and methods
- `this` and `super`
- `final` for methods and for variables
- `static` keyword
- `instanceof` keyword
- casting
- Java generics
- `@Override` annotation
- identity vs. equality (e.g. `==` vs. `equals()`)

Other key concepts:

- Inner classes
- Variable shadowing
- Marker interface

Object Contracts

Methods from `java.lang.Object`:

- `String toString()`
- `boolean equals(Object obj)`
- `int hashCode()`

Exceptions

Know when to you use exceptions (and when not to use exceptions). Understand `try`, `catch`, and `finally`. Be able to follow exceptional control flow.

Exception types:

- **Unchecked** - any subclass of `RuntimeException`; used to indicate programming errors or unrecoverable errors (e.g. `NullPointerException`, `ArrayOutOfBoundsException`)
- **Checked** - not `RuntimeException`; callers should explicitly decide to handle or pass (e.g. `IOException`, `FileNotFoundException`)

Specification and Verification

- Preconditions and postconditions
- Invariants - both loop invariants and class invariants

You should be able to write simple specifications and invariants for sample code.

Testing

Understand why unit testing and its advantages. Understand best practices when writing unit tests using JUnit. Be able to write a good set of unit tests for a sample function.

A unit test suite will typically include:

- some representative cases
- some invalid inputs
- boundary conditions
- stress tests

Terminology and concepts to know:

- **Blackbox** - testing without knowing internal implementation
- **Whitebox** - testing based on a known implementation
- **Assertions**

- **Regression testing**
- **Test-driven development**

Understand the different structural coverage types (method, statement, branch, and path) used in whitebox testing. Also understand the idea of specification coverage used in blackbox testing.

Design Principles

Understand what these principles are and why they are important:

- **Design for Change** - organize programs so possible changes are easy to make
- **Information Hiding** - a principal approach to designing for change; hide design decisions that are likely to change within the implementation of a module (e.g. data representation is often hidden in private fields)
- **Low Coupling** - want classes to be as independant as possible
- **High Cohesion** - want classes with a very specific set of functionalities.

Explain why Worse can be Better

Design Models[1]

Know basic UML notation and understand how to use UML to write a simple class diagram. Understand the notation for inheritance/subtyping vs. association and the notation for multiplicities.

Design Patterns

Know all the design patterns covered in class. You should be able to come up with a pattern name based on its definition, structure, or problem, and vice versa. However, mere memorization is insuffficient for mastery; understanding is critical. Understand the problem each pattern addresses and why it is effective at addressing this problem. Understand the consequences and common variations of the pattern.

We may ask you to:

- identify patterns in source code
- come up with the name given a description or vice versa
- draw the the basic structure of the pattern in UML or sketch code that implements it
- suggest a pattern applicable to a problem situation
- make arguments about the pattern's benefits/drawbacks in a given setting

The design patterns you should know are:

- Adapter
- Command

[1] We posted a cheatsheet here: https://docs.google.com/document/d/1dtWtA5rUAZ8rX2SOZ5zbSsan6nGXEmxQRrLS177yJRw/edit?usp=sharing

- Composite
- Decorator
- Facade
- Factory Method
- Iterator
- Model-View-Controller
- Observer
- Singleton
- Strategy
- Template Method

Frameworks

Hollywood Principle - "Don't call us; we'll call you."

Terminology:

- API (Application Programming Interface)
- client
- hot spots and cold spots
- extension point
- protocol
- callback
- lifecycle methods

You should be able to:

- Explain the difference between a library and a framework, and describe examples of each.
- Describe tradeoffs between the generality of a framework and the support that it provides for any given program.
- Should extending a framework require modifying the source code of the framework?

GUIs

GUIs react to user events. Need code to react to events.

Relevant topics:

- **Event-based programming** - control flow driven by external events (e.g. event listeners in Swing)
- **Model-View-Controller** - understand it and how it is relevant to GUIs.

- **GUI Thread** - understand that GUIs are typically single-threaded and that blocking the GUI thread will cause the program to appear to "freeze up"

Java Collections

Interfaces:

- `java.util.List`
- `java.util.Set`
- `java.util.Queue`
- `java.util.Map`

Other features:

- `java.util.Iterator`
- `Collections.sort`
- `java.util.Comparable`

Java Stream I/O and Networking

Describe the fundamental abstraction of input/output streams.

Know how to read / write objects via streams.

Know how to create sockets and communicate through sockets.

Concurrency

- concurrent vs. parallel
- deadlock
- livelock
- thread starvation
- race conditions (low-level and high-level)
- `synchronized` keyword
- atomicity
- course-grained vs. fine-grained locking
- `wait()` and `notify()`

You should also:

- Be able to reason about potential speedup using **Amdahl's law**, as well as concepts such as work, depth, and breadth.
- Understand higher-level concurrency abstractions, including `ExecutorService` and `ForkJoinPool`.

Static Analysis

Characterize the kinds of errors that static analysis can find.

Describe the tradeoffs between static analysis and other quality assurance approaches, such as testing. Explain these tradeoffs in a particular context, such as finding race conditions. Discuss how you might do a cost-benefit analysis when considering the adoption of a static analysis tool.

Apply principles of applying static analysis in practice, e.g. the need to customize a generic static analysis tool such as FindBugs to a particular engineering organization.

Explain how design intent and other forms of specification (e.g. as provided in JSure) can help make analysis both more modular and more effective.

Define soundness, completeness, false positives, and false negatives. Discuss tradeoffs among these.

Explain how issues of aliasing, uniqueness, and thread-locality bear on properties like safe concurrency.

Distributed Systems

Describe the high-level challenges that make distributed systems especially difficult. Examples: inconsistency, security, and failure (especially partial failure and silent failures).

Describe the benefits and costs of caching, including tradeoffs between simplicity, consistency, bandwidth, and computation. Understand the consequences of caching on the client vs. the server.

Explain the advantages and disadvantages of different partitioning/replication/sharding approaches.

Apply the map-reduce paradigm to solve simple problems in parallel. Understand how the algorithm deals with server failure.

Describe the notion of data consistency and transaction.

Tell if a trace of operations is serializable (or linearizable).

Understand the basic properties of the UDP and TCP communication protocols.

History of Objects

What were the key language innovations introduced by Simula 67 that distinguish object-oriented programming languages? Which is the most important, according to the last course lecture?

Explain the analogy Alan Kay makes between objects in Smalltalk and computers.

Name a feature of a modern language that derives from Smalltalk.