15-214 Exam Review Sheet

Pre-Midterm Topics

1. Object-oriented Concepts

Terms you should be able to define:

- subtype polymorphism and dynamic dispatch
- abstract classes and methods
- superclass / base class
- subclass / derived class
- subtyping (and subtype/supertype) – semantics as substitutability
- inheritance
- overriding

How does object-oriented programming differ from procedural programming (e.g. as in C)?

Most fundamentally, OO organizes code around data objects rather than procedures

OO tends to be driven more bottom-up than top-down, which facilitates re-use as it emphasizes reusable fundamental concepts

OO designs tend to follow concepts from the problem domain

This minimizes required change when the problem changes

This also makes the program easier to extend with new problem domain concepts, assuming they follow patterns set out by earlier concepts.

Benefits of object-oriented interfaces

They specify the design and semantics of an object independent of its implementation

Allows separate development of clients and implementation

Shields clients from changes to the implementation

Allows clients to treat two implementations uniformly, including mixing them or interoperating

2. Java Semantics

Java concepts you should understand:

- final (as applied to fields, methods, and classes)
- static (as applied to fields and methods – we will not test you on static inner classes)
- public/private/protected

- packages (as a way to organize code)
- parameterized types

Understand and be able to draw the conceptual layout of objects and arrays in the heap.

Explain subtyping rules

Implementor of an interface must have at least the methods the interface does

Method implementations must have the same argument types

Method implementations must have the same result type, or a subtype of the type specified in the interface

The method should behave as specified in the interface (e.g. with comments or pre/post-conditions)

Explain or simulate method dispatch in Java

Choose among overloaded methods (compile time)

Identify typechecking errors when no appropriate method exists (compile time)

Choose appropriate method implementation (run time dispatch)

Exception handling in Java

How to throw / catch exceptions in Java

Differences between unchecked exceptions (RuntimeException) and checked exceptions.

java.lang.Object contracts

Know the documented contracts of .equals(Object obj), .hashCode(), .clone() methods defined in java.lang.Object.

Equality contracts can be violated in many different ways. You should be able to tell whether a given equals() method implementation is compliant with the contracts or not.

3. Design Principles

Design for change – the most important OO design principle

Understand the variations in your problem domain – both as the problem exists right now, and how the problem is anticipated to change in the future. I.e. answer what are the concepts, and how do they vary?

Encapsulate those variations using subtyping so that you can plug in different implementations of a concept, and clients will be unaffected.

Encapsulation

An object hides its internals from the outside world, which can only access the object through public methods. "Internals" really means whatever is likely to change.

Representation is something that is often (but not always) likely to change. Therefore fields are often made private or protected so that the representation of an object can be changed without affecting clients.

Reuse

Identify commonalities between concepts that vary, and provide libraries that capture those commonalities. Inheritance is sometimes the most convenient way to share those commonalities because the shared code is part of the same object as the variant code. Using different objects for shared and variant code, connected by interfaces, can sometimes be more flexible, however.

Design process

Use nouns/verbs as a guideline for objects and their properties, and their methods

Identify the responsibilities of an object and the objects it collaborates with

Refine the design using the criteria above

4. Specification, Verification, Hoare Logic, and Testing

Write down pre- and post-conditions, class invariants, or loop invariants for example functions or classes.

Write appropriate unit test cases (including expected answers) for example functions.

5. Design Patterns

Understand what a design pattern is and why it is useful.

Know the design patterns discussed in class: Strategy, Observer, Decorator, Factory Method, Abstract Factory, Singleton, Command, Adapter, Façade, Template Method, Iterator, and Composite.

For each pattern, be able to identify it in source code, come up with the name given a description or vice versa, be able to draw the basic structure of the pattern in UML or sketch code that implements it, suggest a pattern applicable to a problem situation, and make arguments about the pattern's design consequences (benefits and drawbacks) in a given setting.

6. Graphical User Interfaces (GUIs)

How the application code and Swing code interact with each other?

Explain how event handling works in Swing.

Understand Model-View-Controller pattern and the role of each participant.

7. Frameworks

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

Define and use framework-related terms such as API, plugin, extension point, protocol, callback, and lifecycle method.

Should extending a framework require modifying the source code of the framework?

Post-Midterm Topics

8. Java Collections Framework

Distinguish the different types of collections (e.g. Lists, Sets, Maps) and tell the characteristics of each.

Understand the use of Iterators on the collections.

Use the Collections class (c.f. Collection<E> interface) to operate on or return collections.

9. Java Steam I/O, Networking

Describe the fundamental concept of input/output streams.

Know how to read / write objects via streams.

Know how to create sockets and communicate through the sockets.

Understand the concept of higher levels of networking abstractions such as Remote Method Invocation.

10. Concurrency

Be able to reason about potential speedup using Amdahl's law, as well as concepts such as work, depth, and breadth.

Define, distinguish, identify, and give examples of deadlocks, race conditions, and data races.

Apply the map-reduce paradigm to solve simple problems in parallel

Understand the semantics of synchronization in Java

synchronized keyword, wait() / notify()

Understand the higher-level abstraction of concurrency (ExecutorService / ForkJoinPool)

11. 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.

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.

Describe how static analysis can analyze all possible executions of a program even without running all of those executions explicitly.

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.

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

12. Product Lines

Describe the challenges of software product lines.

List different approaches for implementing software product lines.

Describe how design patterns and frameworks can help implementing software product lines.

13. 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.

Explain the advantages and disadvantages of different partitioning approaches.

Describe the notion of data consistency and transaction.

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

14. History of Objects

What was the key language innovation introduced by Simula 67 that distinguishes object-oriented programming languages?

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

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

Name an advantage, and a disadvantage, of a Smalltalk-like pure object-oriented language, in which everything (e.g. including integers) is an object on which you can call methods.