Review sheet for Final Exam
(List of objectives for this course)

Please be sure to see other review sheets for this semester

Please be sure to review tests from this semester

Week 1
Introduction

Chapter 1
Objectives
1. Describe steps of software lifecycle and what each step involves
2. Distinguish between top-down and object-oriented design
3. Understand pre- and post-conditions
4. Understand difference between formal verification and empirical testing
5. Distinguish between methodology vs. ad hoc programming methods

You must know:
1. Software life cycle
   a. Problem spec
      i. statement of exact problem to be solved
      ii. problems – people completeness and language
   b. Program design
      i. Specification of an integrated set of components
         1. components have:
            a. preconditions
            b. postconditions
            c. interface
      ii. Approaches to design
         1. top-down – emphasis on what is being done – break down problem by function
         2. OO design – who is doing something rather than what is being done, decompose along lines of entities in program, leads to classes
   c. Selection of algorithms and data structures
      i. array, list, vector, collections, trees, etc.
   d. Coding and debugging
      i. How can we increase productivity – reuse code and libraries
      ii. Use modern language features (and APIs) such as exceptions, inheritance, streams and event-based programming.
   e. Testing and verification
i. Verification (*formal methods*) very expensive
ii. Testing
   1. unit tests - junit
   2. integration tests – junit but bigger chunks, not just methods
      inside one object but across larger collections or entities
   3. acceptance tests – written by customer
f. Documentation and support
   i. User manual – for the customer
   ii. Technical documents – for the maintainer
g. Maintenance
   i. Deals with issues of correctness (corrective – 20%)
   ii. Deals with changing equipment (adaptive – 20%)
   iii. Changing business requirements (perfective – 60%)
   iv. Expense
      1. good software is maintained (bad software dies and does
         not cost much to maintain)
   v. maintenance matters and costs even if released code has zero
      defects.
2. Cost of phases of software lifecycle
   a. Problem spec - 7%
   b. Program design – 6%
   c. Selection of algorithms and data structures
   d. Coding and debugging – 5%
   e. Testing and verification – 7%
      i. Integration – 8%
   f. Documentation and support and Maintenance – 67%

Week 2

Chapter 2
Objectives
1. Understand conceptual underpinnings of OO programming
   a. Class, object, inheritance, association, dependency
2. Understand terminology of OO programming
3. Explain and use different types of inheritance
4. Pick out terms used in spec. document and determine classes and methods
5. Draw simple UML class diagrams
You must know/be able to:
a. Define objects
   i. State
      1. properties
   ii. behavior
      1. how it reacts
      2. change in state
      3. interaction with other objects
   iii. identity
      1. a way of distinguishing one object from another
2. reference variables used to do this
   iv. encapsulation
      1. Hide the inner workings
   v. Grouped into sets called classes
b. Know equivalence
   i. Name equivalence (reference variables for the same object on the heap)
   ii. Content equivalence (deep, structural property – based on state of objects)
c. Classes – group of objects that share common state and behavior
   i. Class is a blueprint
   ii. Object is instance of blueprint
d. Inheritance
   i. State and behavior of one class are subset of another more general class
   ii. More general class is superclass
   iii. Other class is more specialized instance of superclass
   iv. Is-A relationship holds
   v. Has-a relationship comes when one class component of another
      1. think instance fields
e. Types of inheritance
   i. Specification
      1. like an interface – interface specifies all the behavior but none of the implementation
      2. implementing interfaces fits here
   ii. Specialization
      1. is-a relationship – when we re-implement behaviors that exist in the superclass
      2. extending abstract classes fit here
      3. extending other classes without adding new methods belongs here
   iii. Extension
      1. when we add new behaviors not present in the superclass
   iv. Limitation
      1. new (inherited) class will prevent calling methods
      2. strictly speaking, “limited” methods are re-implemented as no-operations
   v. combination
      1. inheriting from 2 or more superclasses
         a. java limits this to extending one class or abstract class but implementing multiple interfaces
f. OO design
   i. Requirements phase
      1. User’s need (like user stories)
      2. desired effects to be produced by software
   ii. Specification phase
      1. behavior of program
         a. outputs
         b. hardware environment
         c. how to produce the effects desired
iii. design phase
   1. identify classes, behaviors and interactions of classes
      a. nouns
         i. classes
         ii. or components of classes
      b. verbs – methods
   g. UML – class diagrams
      i. Name, state, behavior, associations

**Week 3**

Objectives
   1. Learn about software quality through inspections and reviews
   2. Understand the class design process

You must know/be able to:
   1. difference between walk-through and inspection
      a. walk-through
         i. prep by bringing lists of items
         ii. visually inspect code for about 2 hours
         iii. detect, don’t correct
         iv. document-driven (lists)
         v. verbalize the code – helps find faults
      b. Fagan inspections
         i. Overview
         ii. Preparation
         iii. Inspection
         iv. Rework
         v. Follow-up
         vi. Roles
            1. moderator, reader, author, inspector, specialists, recorder
   2. Things to look for in inspections and reviews
      a. Logic
      b. Adherence to requirements
      c. Formatting:
         i. Meaningful variable and method names
         ii. Consistent formatting (tabs, brackets)
         iii. Capitalization of class names and lowerUpper (camel) format for method/variable names, CONST-VAL format for constants
         iv. Comments for classes, all methods, global variables
      d. Use of public, protected, private, static, final as appropriate
      e. Meaningful, small methods
   3. OO design process
      a. Looking for nouns
         i. State
         ii. Behavior
         iii. Interactions or relationship
Week 4

Objectives
1. Understand how java and JVM run your program
2. Reason about behavior of Java programs based on Heap and stack model of data
3. Reason about behavior of Java programs based on visibility rules
4. Understand reference variables/primitive variables
5. Understand arrays in Java

You should know:
1. what happens when you run a program
   a. java MyClass
      i. jvm loads a bunch of classes and initializes the environment
      ii. jvm loads your classes
      iii. looks for method main in your classes
      iv. main creates objects and we are off to the races
   b. static methods and data “exist” in code area before an object is created
   c. Objects are created on the heap
      i. Live on heap until no reference point to them
   d. Local variables (belonging to methods) and parameters live on stack
      i. Live on stack until the method finishes up or returns
   e. Reference variables don’t create objects, you must use new and constructor to do so
      i. New objects live on heap
      ii. May have multiple reference variables referring to object on heap
      iii. When no more reference variables refer to object on heap it may be garbage collected (returned for later use)
   f. Primitive variables – value is held in the variable
   g. Reference variables – reference to object is held in the variable
   h. Parameters – passed by value (i.e., we copy them on the stack)
      i. Understand why we cannot write a swapper function with primitive variables
2. Visibility rules
   a. Public – if something is public, then any code can see it/ use it and compiler will allow this
   b. Private – if something is declared private, then only code in the same class can see this – we might have called this class visibility.
      Subclasses cannot see these items.
   c. Protected – if something is declared protected, then this class and all of its subclasses can see the item/method
   d. Package – if something is not declared with a visibility, then by default the item or method has package visibility – kind of like public but only for other things in the same package
3. Misc
   a. Final – write once to a variable – either in declaration or once in the code
b. Constructors – get them by default, but once we define a constructor for a class we lose the old default constructor.

4. what happens on instantiation (i.e., when new keyword with constructor is used)
   a. memory allocated – on heap
   b. default initialization occurs - reference variables are initialized to null, integers to 0, doubles to 0, etc.
   c. constructor is invoked

5. Arrays
   a. Declaration
      i. // int [] intArray; - just declares a reference variable NOT an array
      ii. // int [] intArray = new int[55]; - declares a reference and creates an array of int’s
      iii. // int [] intArray = {1,2,3,4,5}; - declares a reference and creates an array of 5 ints and initializes them to 1,2,3,4 and 5 respectively
   b. Where do these things live?
   c. Given code, show where parts live (heap/stack/etc.)
   d. You should understand why the swap method from lab cannot be written
   e. You should understand why you might want a private constructor
   f. You should be able to write a good unit test for a methods that return primitives and reference variables

**Week 5**

Objectives
1. Understand a deep copy
2. Understand Java-based inheritance
3. Understand event-based programming and Java-based GUIs and their heavy use of interfaces

You must know:
1. why inheritance is valuable – polymorphism
2. definition of polymorphism
   a. means “many forms”
   b. objects with different forms, same behaviors
   c. substitution principle – we use subclasses anywhere superclass is used
3. Inheritance
   a. Interfaces – implements keyword
   b. Abstract classes
   c. Classes – extends keyword
   d. Constructors –
      i. Default version calls super()
      ii. We can override
iii. You should understand the order in which default constructors call super()

4. Overriding versus overloading
   a. Override – inherit an implementation of a method and we rewrite the method to get new behavior
      i. We get toString() for free, but usually we rewrite it to get nicer output
   b. Overload – have an existing form of a method and we rewrite the method with new parameter types
      i. + operator can add integers and concatenate Strings
      ii. Don’t have to use inheritance to do this – we can overload a method in a single class

5. Interfaces and Abstract classes
   a. How to declare each
   b. How to inherit based on them
   c. Type of inheritance each implies

6. GUIs and Event-based programming
   a. GUIs – visual tool for interacting with user
   b. Widget – graphical components like buttons, etc
   c. Event – object that represents user action on a widget
   d. Listener – object whose job is to wait for event
      i. Interfaces often used to specify event listeners classes

7. GUI specifics
   a. JFrame –
      i. Constructor
      ii. Setting default close
      iii. setSize() and setVisible() methods
      iv. contentPane() – getting and adding objects
   b. JLabel – constructor and setting text
   c. JButton – constructor and setting text
   d. JTextField – constructor, setting and getting text
   e. Layout Manager

8. Event handling
   a. ActionListeners are interfaces, not classes
   b. Must define actionPerformed(ActionEvent e)
   c. Must addActionListener to register listener with Widget

Week 7

Objectives
1. Exceptions (also see later weeks and look at chapter 10)
2. See a bigger example of OO design from start to finish
3. Understand how to go from problem requirements (user defined) to program specification (how program will do it) to design (id of classes) to use of inheritance and use of UML

DO NOT WASTE TIME MEMORIZING CLASSES OR METHODS FROM THIS CHAPTER
You must:
1. understand role of exceptions
   a. without exceptions, code must look for unusual events or risk missing them
   b. with exceptions in languages, exceptions are types or classes used to describe unusual events
      i. exception object created to describe event
      ii. exception handler analyzes the exception event and takes appropriate action
2. Understand exceptions are instantiated (i.e., have constructors)
3. Understand syntax of try-catch block
   a. “parameter” to catch block is type of exception object
   b. May have several catch blocks for one try block
   c. You have to either handle or rethrow
4. Understand the throws keyword
   a. If exception not handled in your method, the calling method will need to handle an exception

**Weeks 8 and 9**

Objectives:
1. understand reference-based structures – linked lists and stacks
You must know this about linked lists:
   1. understand list structure can grow as large as needed (unlike arrays)
   2. understand behaviors
      a. add, remove, add/remove to/from certain positions, append to end, get and set values already on list
   3. Understand basic recursive class definition (object, reference to next node)
   4. Understand behavior of methods (from text and labs)
   5. Understand how linked list is represented in stack and heap
You must know this about stacks:
   1. understand push, pop, top, isempty and height
   2. understand behavior of stack (namely, first on the stack, last off the stack)
While queues are just a simple extension of linked lists and were not covered explicitly in class, you might want to look at Queues in the text:
   1. we saw an example of a queue in exam 3
   2. we wrote code for a queue in lab exam
   3. understand they are like check-out lines in a grocery store
Week 10

Objectives:
1. Understand recursion
2. Understand risks of developing software

You must be able to/know about
1. Examples of Risks
   a. Schedule slippage
   b. Project is cancelled
   c. System goes bad
   d. Defect rate so high, never used
   e. Business misunderstood
   f. Business changes
   g. False feature rich

2. Examples of how to handle risks
   a. Develop in small iterations with short deadlines, i.e.:
      i. Get user stories
      ii. Break into tasks
      iii. Estimate time
      iv. Get customer to prioritize
         1. Avoid false feature rich
         2. If they don’t want to pay for it don’t do it (yet)
      v. Get acceptance test (so you know when you are done)
      vi. Sign up for wanted tasks
      vii. Pair program
         1. Instant code reviewer on hand
      viii. Test first, write unit tests – this can serve as some of your technical documentation
      ix. Use revision control (CVS) to keep coders from stomping on one another
      x. Keep only working (compiling and running) code in CVS
      xi. Show working code (that passes acceptance test) to client
      xii. Rinse and repeat

3. Recursion
   a. How does it work – base case and forward progress (break problem into smaller instance)
   b. Understand factorial
   c. Show you can trace through the execution of a recursive program
      i. Examples are factorial, reverse, addition, multiplication, exponentiation
      ii. Other examples are based on linked-lists and trees

Weeks 11-14

Objectives:
1. Understand binary trees and how they are structured

You must know/understand how to:
1. know terminology from previous review sheet on trees
2. build a binary tree
3. build a binary search tree
4. add and remove items from binary- and binary search trees
5. traverse binary trees – inorder, preorder, postorder