SEG 2105 MT1 REVIEW

Ch 1

- Software Lifecycle:
  - Requirements & Specifications
  - Design
  - Modeling
  - Programming
  - QA
  - Deployment
  - Managing the process

- Quality: 5 attributes (RUMER)
  - Usability
  - Efficiency
  - Reliability
  - Maintainability
  - Reusability

- Agile: emphasize ability to quickly modify software } best for small-medium systems
  - Gather small # of requirements
  - Design & implement all
  - Gather next small # of reqs. etc..

- Waterfall: traditional
  - Develop all requirements
  - Design ∀
  - Test ∀ & Deploy

- Types of Software:
  - Real-time embedded (software): Must react immediately, safety is a concern (Eg. Nuclear Plant control)
  - Data processing: Used to run business, accuracy & security of data are key.
  - Game software
  - Mobile device software
  - Web-based software
  - etc.

Hilary

Ch 2

- Inheritance: IsA, ↑
- Abstract: Abstract classes used to organize code & increase reusability of code
  - Object, Class, Super/Subclasses, Operations (methods), Attributes & Associations (instance variables)
- Modularity:
  - Code divided into classes
  - Classes divided into methods
- Encapsulation:
  - A class holds its features. It can choose what is accessible from the outside ⇒ information hiding
- Contract:
- Interface:
  - No executable statements
  - Abstract data type (no instances)
  - Can declare constants using static final
    - no need to instantiate
    - ↳ can't be modified at runtime
- Exception:
  - Class w/ subclasses for things that could go wrong
  - throws keyword forces people calling method to try{method call} catch(ArithmeticException e){Sysout("there was ÷0");} → code continues
  - using try{}catch(Exception e){} within method is better in general because people calling it don't need to worry w/ catching it. Use if all calls require different implementation of error handling.
- Polymorphism:
  - Plant plant1 = new fern();

SEG2105 MT1 REVIEW

Ch2 continued...

- Dynamic Binding: Plant plant1
  - Needed when variable type is superclass AND more than one polymorphic method among the type & its subclasses
  - Steps: look in object type, if not there, look in superclasses until found or error
    = new Fern();

- Instance vs. Class:
  - Class is the mold, Instance is the object created from the mold

- Casting: Converting from one type to another
  - Upcasting: casting into a superclass, always allowed, usually done automatically. eg. Plant plant1 = new Fern(); is same as Plant plant1 = (Plant) new Fern();
  - Downcasting: casting into child class, should avoid (use generics instead), always done manually, can fail.
    - Success example: Fern f = new Fern(); Plant p = f; Fern f2 = (Fern) p;
    - Failure example: Tree t = new Tree(); Plant p2 = t; Fern f3 = (Fern) t;
      fails because this instance of Plant is not a Fern

Abstraction: using classes, superclasses, methods to simplify & organize code.

Ch 3

- Framework: reusable software for applications that do different things, but have similar designs.
  - Frameworks are incomplete: certain classes/methods used by the framework are missing implementation (hooks, slots)
  - Inversion of Control (Hollywood principle): "Don't call us, we call you"
  - Composed of library/libraries of classes

- Library: Collection of code called from our code that helps do things of related functionality.

- API: Set of all public methods of the library of classes that composes the framework
  ↳ Library is a function of an application, framework is skeleton of application, API connects L&F together

- Product Line: family of products built on common base technology.
  Each product fills-in hooks & slots.
  Eg. Windows home, professional, etc.

- Hook: Method that can be overridden, but returns w/o doing anything by default

- Slot: Method not implemented by default that must be implemented manually by the programmer.

- Fat Client: when most of the calculations & work is done by client.
  Usually involves large upfront download, eg. COD PC

- Thin client: Server handles most of the load. Either for simple applications (webpages) or for security reasons (banking systems)

- TCP/IP: Application $\xrightarrow[\text{http}]{\text{port 80}}$ TCP $\rightarrow$ packets $\rightarrow$ IP: origin $\rightarrow$ destination [re-assemble]
  packets $\rightarrow$
  (TCP: port to port, IP: host to host)

- Sockets: keep all information about client's connection to server
  Socket : Thread = 1:1 (new socket creates new Thread();
  #Threads: in server: #clients + 1, in client: 2
  #Threads in client: 2

Ch 4 (only what is needed for MT 1)

- Requirement: - Defines System, Verb (Shall/~~May~~/Must), End Result, Measurable Quality Criteria
  - Must explain what it does, not how it does it.

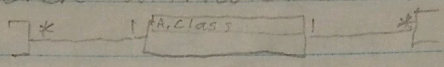
- Types of Requirements:
  - Functional: What system does (inputs, outputs, computations, timing, stored data, etc.)
  - Non-Functional: Constraints on
    - Quality: (RUMER) specified levels of quality
    - Platform: environment & technology of system (Win XP, OS)
    - Process: project plan & dev methods

Ch5:

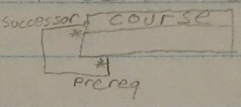
- Attributes: Instance variables

- Associations: - representation of how two classes are related to each other (What kind of instance variables they store of each other)
  - 1-1 should be avoided

- Association Classes: when attribute needs to be in both classes



- Reflexive Association: Association connecting w/ itself (should have labels)



- Generalization: subclass points to its superclass (Check IsA rule)

- Aggregation: Association representing part-whole relationships.
  - ◇ goes next to whole (AKA aggregate): vehicle ◇ 1 — * vehicle Part

- Composition: Special kind of aggregation where if whole is destroyed, parts are as well
  - ◆ goes next to whole (AKA aggregate): Building ◆ 1 — * Room

- Directionality: Association where only one has reference to the other
  - Day 1 ——> * Note => Day { List<Note> Notes; } Note { /nothing }

- Propagation: Occurs when aggregate performs an operation on its parts.
  Must be manually implemented (eg. delete 1 => delete other

- System Model: Models everything from domain of system to User Interface (can generate code)

- System Domain Model: Models domain only, will be used to model code

- Exploratory Domain Model: Models parts of domain to learn about domain, no code is generated from this