## CS 494

## Object-Oriented Analysis & Design

# Course Introduction

**Dr. Tom Horton**
**Email: horton@virginia.edu**
**Phone: 982-2217**
**Office Hours: Mon. & Wed., 3:30-5 p.m. and Thur. 2-3:30 (or by appointment)**
**Office: Olsson 228B**





## Course Overview

- **Object-oriented SW Engineering, Development**
  - Low-overhead requirements analysis methods
  - OO domain model of requirement-level objects, relationships.
  - Translating requirements into OO designs
  - Evaluating designs for quality
- **Other Design Issues**
  - Design patterns, refactoring, testing, architecture, persistence,…



## Course Overview (cont'd)

- **Java development**
  - Core Java competency
  - Translate designs into Java
  - (Maybe) Explore more advance Java topic (GUI development, database connectivity, unit-testing)



## Technologies

- **Requirements**
  - CRC cards, use cases, XP stories
- **Modeling requirements and design**
  - UML: class diagrams, sequence diagrams,…
  - UML tools: MS Visio (free to you!), Together ControlCenter, others
- **Implementation**
  - Java
- **Development and Testing**
  - Some IDE with debugger, GUI support
  - JUnit



## Course Overview

- **Balanced approach: notation vs. principles, front-end vs. implementation**



## Outcomes: What you should learn

- **Object-oriented analysis and design**
  - Goals, what to do, what not to do
  - What to model and how to evaluation it
- **UML**
  - Use cases, class diagrams for requirements specification
  - Class diagrams, sequence diagrams, state diagrams, packages for design
- **Design patterns**
  - What they are, how they're described, a few common patterns

## Outcomes (cont'd)

- **Modeling and the SW lifecycle**
  - Clear understanding of the role of UML models throughout lifecycle
  - How requirements models are transformed to design
  - How design models transform to code
- **Evaluation**
  - Assessing design quality
  - On your own and using *formal technical reviews*



## Resources

- **Textbook: *The Object Primer. 2nd edn.* Scott W. Ambler (Cambridge Univ Press, February 15, 2001).**
- **Java book:**
  - Got one? OK (probably)
  - *Just Java 2. 5th edn* (Prentice Hall, 2001). By Peter van der Linden.
  - Or, Eckel's *Thinking in Java*. Printed or on-line!
- **Web site:**
  **http://www.cs.virginia.edu/~horton/cs494**
  - Course news, slides, Java and UML links, etc.
- **Student on-line survey**
  - Fill out ASAP, please!



## Pre-requisites

- **For programming needs: must have CS216**
- **This course was planned as a successor to CS340**
  - CS494 is about OOP and about software engineering
- **Students in this course must:**
  - Know what you do in requirement specification
  - Know how that differs from design
  - Know how to do a formal technical review
- **Review CS340 slides or Jalote's textbook:**
  - Pages 73-87, 98-107, 273-294



## Grading

- **Mid-term Exam. 20%.**
- **Final Exam. 25%. Friday, May 9. 2-5 pm.**
  - Partly comprehensive
- **Homework assignments, including Java programming. 20%.**
- **Project work. 35%**
- **Question: Tell me about your Sr. Thesis deadlines…**



## Project Work

- **Create OO models and documentation for proposed system(s)**
  - Requirements models, then design models
  - Coding from design? Some….
- **Multiple parts or in stages.**
- **Work done by small teams (2-3)**
- **You'll evaluate others' projects.**
  - Formal technical reviews (as learned in CS340)
  - Evaluation checksheets and a process



## Programming and Homework

- **Problem: lots of you, less of me and grader…**
- **Let's talk!**

## Computing Needs

- **Course goal: Learn a UML/OOA&D CASE tool**
- **Microsoft Visio**
  - In CS and ITC labs
  - We can give you a copy!
- **Rational Rose? Ugh!**
  - Instead, Together Control Center http://www.togethersoft.com/
    - Needs Java VM (UNIX or Windows)
    - Download and talk to me about a license





## Idioms, Patterns, Frameworks

- **Idiom: a small language-specific pattern or technique**
  - A more primitive building block
- **Design pattern: a description of a problem that reoccurs and an outline of an approach to solving that problem**
  - Generally domain, language independent
  - Also, analysis patterns
- **Framework:**
  - A partially completed design that can be extended to solve a problem in a domain
    - Horizontal vs. vertical
  - Example: Microsoft's MFC for Windows apps using C++



## Examples of C++ Idioms

- **Use of an Init() function in constructors**
  - If there are many constructors, make each one call a private function Init()
    - Init() guarantees all possible attributes are initialized
    - Initialization code in one place despite multiple constructors
- **Don't do real work in a constructor**
  - Define an Open() member function
    - Constructors just do initialization
    - Open() called immediately after construction
  - Constructors can't return errors
    - They can throw exceptions



## Design Patterns: Essential Elements

- **Pattern name**
  - A vocabulary of patterns is beneficial
- **Problem**
  - When to apply the pattern, what context.
  - How to represent, organize components
  - Conditions to be met before using
- **Solution**
  - Design elements: relationships, responsibilities, collaborations
  - A template for a solution that you implement
- **Consequences**
  - Results and trade-offs that result from using the pattern
  - Needed to evaluate design alternatives



## Patterns Are (and Aren't)

- **Name and description of a proven solution to a problem**
- **Documentation of a design decision**
- **They're not:**
  - Reusable code, class libraries, etc. (At a higher level)
  - Do not require complex implementations
  - Always the best solution to a given situation
  - Simply "a good thing to do"

## Example 1: Singleton Pattern

- **Context: Only one instance of a class is created. Everything in the system that needs this class interacts with that one object.**
- **Controlling access: Make this instance accessible to all clients**
- **Solution:**
  - The class has a static variable called *theInstance* (etc)
  - The constructor is made private (or protected)
  - Clients call a public operation *getInstance()* that returns the one instance
    - This may construct the instance the very first time or be given an initializer



## Singleton: Java implementation

```
public class MySingleton {
  private static theInstance =
        new MySingleton();
  private MySingleton() { // constructor
    …
  }
  public static MySingleton getInstance() {
      return theInstance;
  }
}
```



## Static Factory Methods

- **Singleton patterns uses a *static factory method***
  - Factory: something that creates an instance
- **Advantages over a public constructor**
  - They have names. Example:
    **BigInteger(int, int, random)** vs.
    **BigInteger.probablePrime()**
  - Might need more than one constructor with same/similar signatures
  - Can return objects of a subtype (if needed)
- **Wrapper class example:**
  **Double d1 = Double .valueOf("3.14");**
  **Double d2 = new Double ("3.14");**
- **More info: Bloch's *Effective Java***

In the system analysis or object-oriented analysis phase of software development, the system requirements are determined, the classes are identified and the relationships among classes are identified. The three analysis techniques that are used in conjunction with each other for object-oriented analysis are object modelling, dynamic modelling, and functional modelling. Object Modelling. Object modelling develops the static structure of the software system in terms of objects.Â  Structured Analysis vs. Object Oriented Analysis. The Structured Analysis/Structured Design (SASD) approach is the traditional approach of software development based upon the waterfall model. The phases of development of a system using SASD are â^'. Feasibility Study. Object-Oriented Analysis and Design module teaches students on how to effectively use object-oriented technologies and software modeling as applied to software development process with the help of Unified Modeling Language (UML). UML is the standard language for object-oriented analysis and design. UML is used throughout the software development life cycle to capture and communicate analysis and design artifacts. In this course you will use graphical modeling language, to communicate concepts, decisions, understand the problem, propose the solution and manage complexity of artifacts.