Object-Oriented Analysis and Design

Themes Underlying Object-Orientation
- Abstraction
  - focus on essential properties, ignore unimportant details
- Encapsulation
  - separate external, visible behavior from internal, hidden behavior
- Combining data and behavior
  - objects, not developers, decide how to carry out operations
- Sharing
  - similar operations and structures are implemented once
- Emphasis on object-structure rather than procedure structure
  - behavior more stable than implementation

Concepts for Implementing Object-Orientation
- Identity
  - program elements are organized into objects
  - object are software bundles of related variables and methods
  - objects interact and communicate with each other using messages
- Classification
  - objects with the same structure and behavior are grouped into a class
  - class is a blueprint defining variables and methods common to a set of objects
- Polymorphism
  - the same operation can perform differently on different classes
- Inheritance
  - sharing of attributes and operations among classes based on a hierarchical relationship
- Role
  - an interface is a contract in the form of method and constant declarations
  - class implementing interface promises to implement all declared methods

Analysis and Design Goals
- Seeing the big picture
- Communicating with domain experts
- Communicating with other developers

Object-Oriented Analysis and Design
- Uses OO concepts to develop models
- Models perspectives can change during development
  - Conceptual - focus on concepts in application domain
  - Specification - focus on class interfaces
  - Implementation - focus on class implementations

The UML
- We will use the UML in our analysis, design and development
  - UML Stands for: Unified Modeling Language
  - UML is a symbolic language for specifying, constructing, and documenting software systems at all levels
UML Diagram Types

- **Requirements**
  - Use case diagrams - provide snapshots of system behavior

- **Class Structural**
  - Class diagrams - describe objects, classes and static relationships between them
  - Interaction diagrams - describe how objects collaborate in some behavior
  - Activity diagrams - describe process flows
  - State diagrams - describe the behavior of a given class

- **Architecture**
  - Package diagrams - show class dependencies
  - Deployment diagrams - show physical relationships among software and hardware components

Overview of Analysis and Design Process

- **Inception**
  - Initial concept development

- **Elaboration**
  - Detailed requirements
  - High-level analysis
  - Baseline architecture
  - Construction plan

- **Construction**
  - Build system interactively

- **Transition**
  - Optimization
  - Beta testing
  - User training

Inception

- **Goals**
  - Establish business rationale
  - Establish scope of the project
  - Not a focus on this class, but clearly important

Elaboration

- **Goals**
  - Get a better understanding of the problem
  - Requirements: What are we actually going to build?
  - Design: How are we going to build it?
  - Construction: What technology are we going to use?
  - Determine risks
  - Requirements
  - Technology
  - Skills
  - Develop baseline architecture
  - Develop construction plan

Requirements Risks

- Identify all potential use cases
  - UML use case diagrams
- Develop domain model
  - UML class diagrams
  - UML activity diagrams

Technological Risks

- Do prototyping experiments
  - Pay attention to interfaces
  - Ex: Building system using C++ and a relational database
- Identify major architectural decisions
  - Pay attention to things that will need to change later
  - Technology failures
  - Interconnection failures
  - Other failures
- UML: Probably just simple sketches at this point
  - Class and interaction diagrams
  - Package diagrams
  - Deployment diagrams
Skills Risks
- Training
- Mentoring
- Reading

Baseline Architecture
- List of use cases
- Domain model
- Technology platform

Construction Plan
- Analyze use cases
  - prioritize use cases
  - consider architectural risk
  - if use case is developed late will rework have to be done?
  - consider schedule risk
  - can you estimate development time for each use case?
- Estimate development time for each use case
- Set iteration length
  - high priority or high risk use cases should be done early
  - don’t put off risk until end
- Add 100% for transition and contingencies

Construction
- System is built in a series of iterations
- Risk appears when difficult issues are left to the end
  - integration and testing should be continuous
  - build in debugging information
  - error messages should be self-evident
  - don’t throw away test code

Construction
- For each use case
  - start with conceptual class diagram
  - as you move to design, change to specification perspective
  - for implementation and documentation use implementation perspective
- UML
  - state diagrams for classes with complex lifecycle behavior
  - draw interaction diagrams for every use case
  - Use text commentary where useful

Transition
- Last steps before delivery
  - beta testing
  - user training
  - optimization
- Resist temptation to do optimization too early