Goals and Objectives

- Chapters 2 and 4 / Hour 2
- Objects and Classes
- Abstraction and classification (generalization)
- Kinds of objects and classes
- Class relationships
  - Generalization (inheritance)
  - Realization (interfaces)
- Association
- Aggregation
- Composition
- Dependency
- Messages and methods
- Defining features of the object-oriented model
- Encapsulation
- Inheritance (generalization/specialization)
- Polymorphism

What Is An Object?

Compare with Object-Oriented Approach, p. 16

- An entity that corresponds to something in the real world
- An entity that has responsibilities
  - Data it "knows things"
  - Operations it "knows how to do things"
  - It responds to messages
  - It provides services
  - Can cooperate or collaborate with other objects (send them messages)
- Boundaries
  - Has an interface (public operations and data)
  - Limits interior access, implement data hiding (private)
- Identifiable existence or lifetime (instance)
- Identity (name)

Classes vs Objects

An object is an instance of a class

- A class is a data type (an abstract data type or ADT)
- An object is a variable
- An instance of a class
- It usually has a name (but may be anonymous)

Naming Conventions

See Hour 2, p. 21

- The UML uses “Camel Notation”
  - Class names begin with capital letters
  - Attribute and behavior names begin with lower case letters
  - Second and subsequent words in compound names are capitalized
  - Spaces may appear in UML names but not in program code
  - Java style also specifies Camel Notation

Abstraction

Controlling complexity by identifying germane responsibilities

- Abstraction
  - Identifying the essential features in the problem domain
  - Separating or filtering essential features from extraneous detail
  - What is "essential" and what is "extraneous"
  - Depends on the problem
  - Contrast this with a "shopping list" of features
  - Best approach depends on the object’s “position” in the program
- The Stealth Fighter model
  - About one-tenth the size of the final aircraft
  - Did not include engines, avionics, armament
  - Did represent the true airframe shape
  - Did have radar absorbent material (RAM) surface coating
Classification
Finding classes: from observation to generalization

- The "real world" contains objects not classes
- We observe objects and classify them based on attributes, behaviors, and relationships
- Classes are artificial and arbitrary

Object Roles
The "star" vs the "extra"
- "Knowledge" of program is high at top, low at bottom
- Features of application classes are problem specific
  - Often complex
  - Get the job done
- Features of general or library classes are broad
  - Often small and simple
  - "Shopping list" of features
  - What might someone want?

Common Class Categories
Grouping classes into components
- Coad and Yourdon
  - Problem domain (application, may require subdivision)
  - Human interaction (interface/controls/GUI)
  - Task management (concurrency: threads/processes)
  - Data Management (database/DBMS)
- Satzinger and Ørvik
  - User interface (controls/GUI)
  - Operating environment (OS/Computers)
  - Task-Related
    - Document
    - Multimedia
    - Problem domain
    - Implementation (implicit; not in the real world but needed by the program)
- Object-Oriented Approach, pp. 19-24

Class Relationships
Programs are built on the skeleton of classes and relationships
- Relationships between classes
  - Promotes reuse (instructions and data)
  - Allows objects to cooperate or collaborate
  - Messages are sent along relationship lines
- Two basic relationships
  - OO features (generalization / specialization, also known as inheritance)
  - Structural (whole / part)
    - Association
    - Aggregation
    - Composition
  - Dependency, also known as using and delegation

Generalization / Specialization
Inheritance
- Subclasses inherit all super class features
- Super classes are general
- Subclasses are more specific
- Hierarchy may be arbitrarily high
  - "Is A" relationship

Gen/Spec Hierarchies
Reading inheritance diagrams
- A Square is a (special) Rectangle
  - Has all of the features of a Shape, a Polygon, and a Rectangle
  - Has features the others do not
- A Rectangle is a (special) Polygon
  - Has all of the features of a Shape and a Polygon
  - Has features that Shape and Polygon don't
- A Polygon is a (special Shape)
  - Has all of the features that a Shape does
  - Has features that a Shape doesn't
  - Read the inheritance relationship from the subclass to the super class
An interface is a special class
- Defines function/method signatures and return value types
- It does not have attributes (data)
- It does not define bodies for any functions or method
- Java has interfaces (i.e., interface is a Java keyword)
- Realization is like inheritance except that it is done with interfaces
- In Java, you implement an interface (i.e., implements is a Java keyword)

Realization
Java: `implements`
Classifying Methods
Satzinger & Ørvik, The Object-Oriented Approach, pp. 43-44

- **Standard methods**
  - Add a new object
  - Show information about an object
  - Delete an object
  - Change the values of attributes of an object
  - Database synonyms: add, query, delete, and update

- **Custom methods**
  - Tailored for a specific class
  - Reflect the responsibilities of an object
  - Are the special services provided by an object
  - Focus on custom methods during analysis; add standard methods during design and implementation

Classifying Services
Coad and Yourdon, Object-Oriented Analysis, pp. 147-148

- **Algorithmically-Simple services**
  - Create instantiate and initialize a new object
  - Connect establishes / breaks a relationship between objects
  - Access gets (accessor) or sets (mutator) an attribute
  - Release disconnects and deletes an object
  - Algorithmically-simple services may not always be shown in a class diagram

- **Algorithmically-Complex services**
  - Calculate calculates a value from one or more attributes
  - Monitor monitors an external system or device; manages inputs and outputs
  - Access methods or functions set or get attributes
  - **Accessor** methods get attributes
  - **Mutator** methods set attributes

Defining Object-Orientation
Summary: The three “Crown Jewels”

- **Encapsulation**
  - Packaging data and operations together
  - Synonymous with an object
  - Implements *data hiding* (data defined in a class are not visible or accessible from outside the class)

- **Inheritance**
  - Reusing in subclasses gain features defined in a super class
  - *Object-based* systems support objects but not inheritance

- **Polymorphism**
  - Determining at run-time which function / method to call
  - Letting an object determine which function / method is appropriate