Object-Oriented Design

Lecturer: Raman Ramsin

Lecture 9:
Generalization/Specialization
Analysis Workflow: *Analyze a Use Case*

- The *analysis workflow* consists of the following activities:
  - Architectural analysis
  - **Analyze a use case**
    - **Outputs:**
      - analysis classes
      - use case realizations
  - Analyze a class
  - Analyze a package
Generalization

- Generalization is a relationship between a more general thing and a more specific thing:
  - the more specific thing is consistent in every way with the more general thing.
  - the *substitutability principle* states that you can substitute the more specific thing anywhere the more general thing is expected.
  - generalization applies to all classifiers and some other modeling elements.
Generalization/Specialization

- Generalization hierarchies may be created by generalizing from specific things or by specializing from general things.
Inheritance

- Class inheritance is implicit in a generalization relationship between classes.

- The subclass inherits the following features from its parents - attributes, operations, relationships, and constraints.
Inheritance: Overriding

Subclasses may:
- add new features;
- override inherited operations:
  - the subclass provides a new operation with exactly the same signature as the parent operation it wishes to override; the operation signature consists of an operation name, types of all parameters in order, and return type.
Abstract Operations and Classes

- Abstract operations are designed to have no implementation:
  - they serve as placeholders;
  - all concrete subclasses must implement all inherited abstract operations.

- An abstract class has one or more abstract operations:
  - abstract classes can't be instantiated;
  - abstract classes define a contract as a set of abstract operations that concrete subclasses must implement.
Abstraction Level

- all things at the same level in a generalization hierarchy should be at the same level of abstraction.
Polymorphism

- Polymorphism means "many forms". It allows you to design systems using an abstract class, then substitute concrete subclasses at runtime - such systems are very flexible and easy to extend; just add more subclasses.
- Encapsulation, inheritance, and polymorphism are the "three pillars" of OO.
Polymorphism: Example

Class diagram showing the `Shape` class with methods `draw`, `getArea`, and `getBoundingArea`. Subclasses `Square` and `Circle` inherit these methods. Diagram illustrates method dispatching on a `Canvas` object with different instances of `Circle` and `Square` classes.
Polymorphism: Concrete Operations

- Polymorphic operations have more than one implementation:
  - different classes may implement the same polymorphic (abstract/concrete) operation differently;
  - polymorphism allows instances of different classes to respond to the same message in different ways.

Diagram: UML class diagram showing relationships between classes and operations.
Advanced Generalization: Generalization Sets

- Generalization set - a set of subclasses organized according to a particular rule; constraints:
  - \{complete\} - generalization set contains all possible members;
  - \{incomplete\} - generalization set does not contain all possible members;
  - \{disjoint\} - an object may be an instance of no more than one of the members of the generalization set;
  - \{overlapping\} - an object may be an instance of more than one of the members of the generalization set;
  - \{incomplete, disjoint\} - the default.
Generalization Sets: Implementation

Diagram:

- **Shape**
  - **Cube**
  - **Sphere**
  - **Pyramid**
  - **threeDShape** {disjoint, incomplete}

- **twoDShape** {disjoint, incomplete}
  - **Square**
  - **Circle**
  - **Triangle**

- **ThreeDShape**
  - **Cube**
  - **Sphere**
  - **Pyramid**
  - {disjoint, incomplete}

- **Shape**
  - **TwoDShape**
  - {disjoint, incomplete}
  - **Square**
  - **Circle**
  - **Triangle**
Generalization Sets: Example
Reference