SOFTWARE DESIGN AND ARCHITECTURE
SE2002

Software Design Method and Object-Oriented Analysis and Design

Spring 2023
Dr. Muhammad Bilal

Outline

1. Software Design Methods
2. Structural Vs Object Oriented Paradigms
3. Design goals
4. UML Diagrams

Software Design Methods

Software Design Methods

- In a Software development process, the Software Design Methodology (SDM) refers to specific set of procedures used to create a conceptual design for fulfilling the set of requirements.

Software Design Methods

- The choice of the SDM primarily depends upon several factors, namely,
  - the *type of the software* (such as standalone or distributed and networked; Strategic or operational etc.)
  - the *scope of the development project* (such as revamp of the existing system or new system, the number of modules involved, underlying complexity of the coding, system testing and implementation etc),
  - the *resources constraints* (such as time, money, expertise)

Software Design Methods

Systematic approaches to developing a software design.

- Structured (Function-Oriented)
- Object-Oriented
- Data-Oriented (Data-structure-centered)
- Component-based
- Formal Methods

Software Design Methods

- **Structured Methods**
  - Process functions are identified
  - Process intensive tasks
- **Object-Oriented**
  - develop an object model of a system
  - To understand real-world entities and their relationship
- **Data-Oriented**
  - Entities are determined for each sub-system, then entity inter-relationships are examined to develop the additional entities needed to support the relationships.
  - Database and banking applications

Software Design Methods

- **Component-based**
  - Divide the system into components
  - For large systems that can be modularized.

- **Formal Methods**
  - Requirements and programs are translated into mathematical notation
  - For safety and security systems
  - Expensive to implement

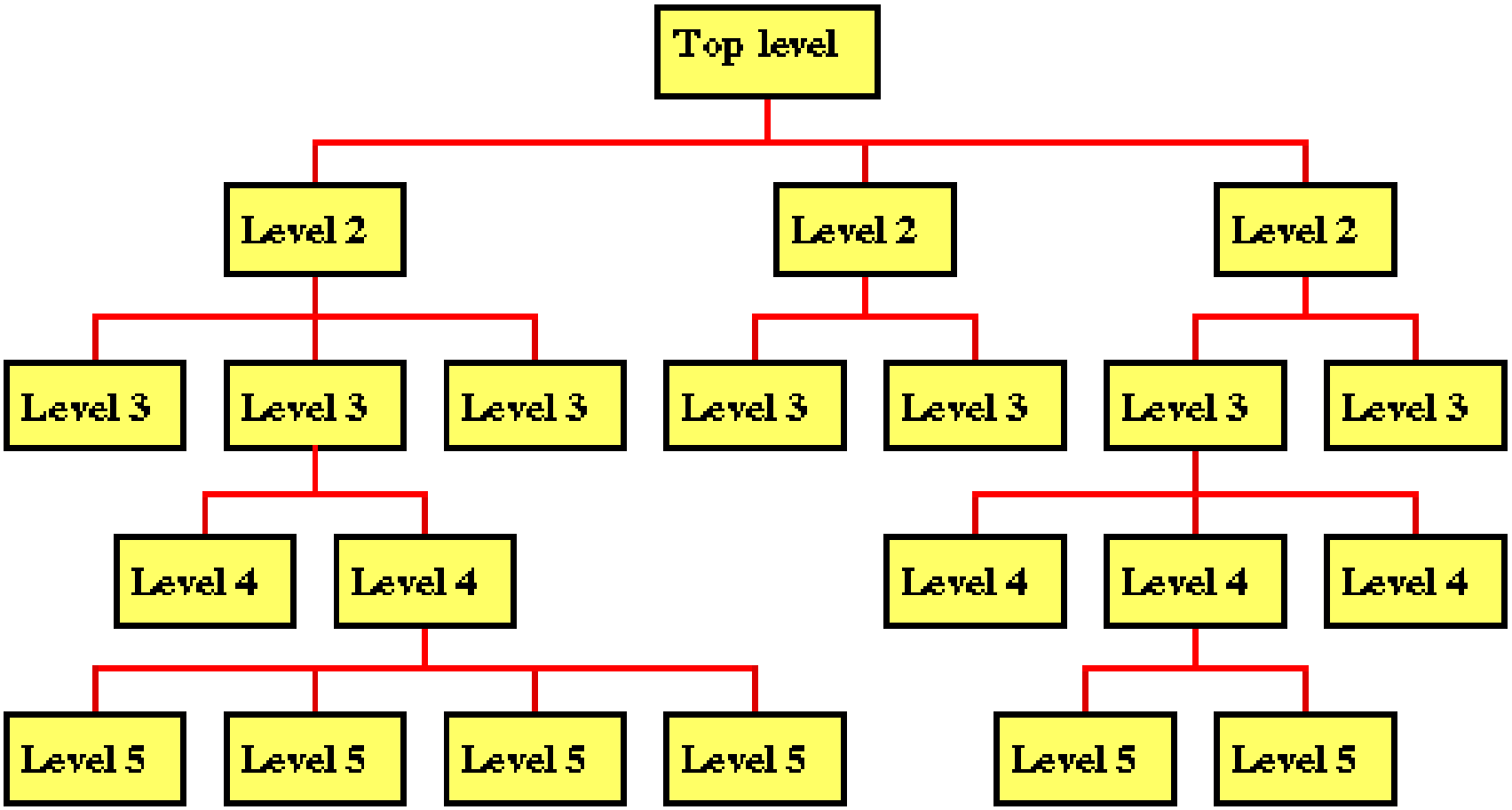
Design Paradigms

Software Design Paradigms

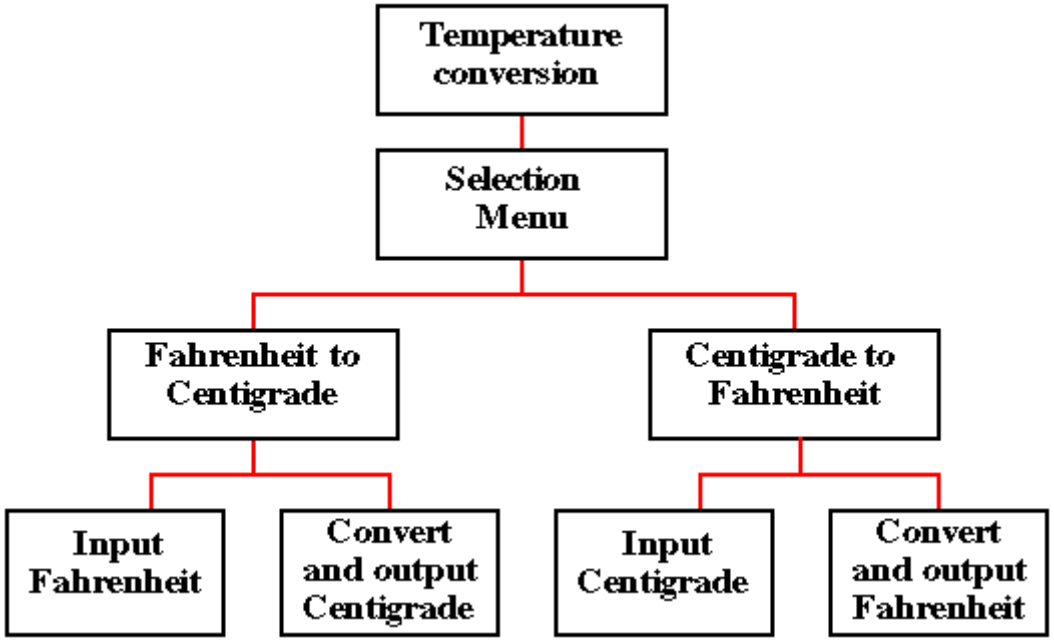
- Structured Design/Function Oriented Design
- Object-Oriented Design

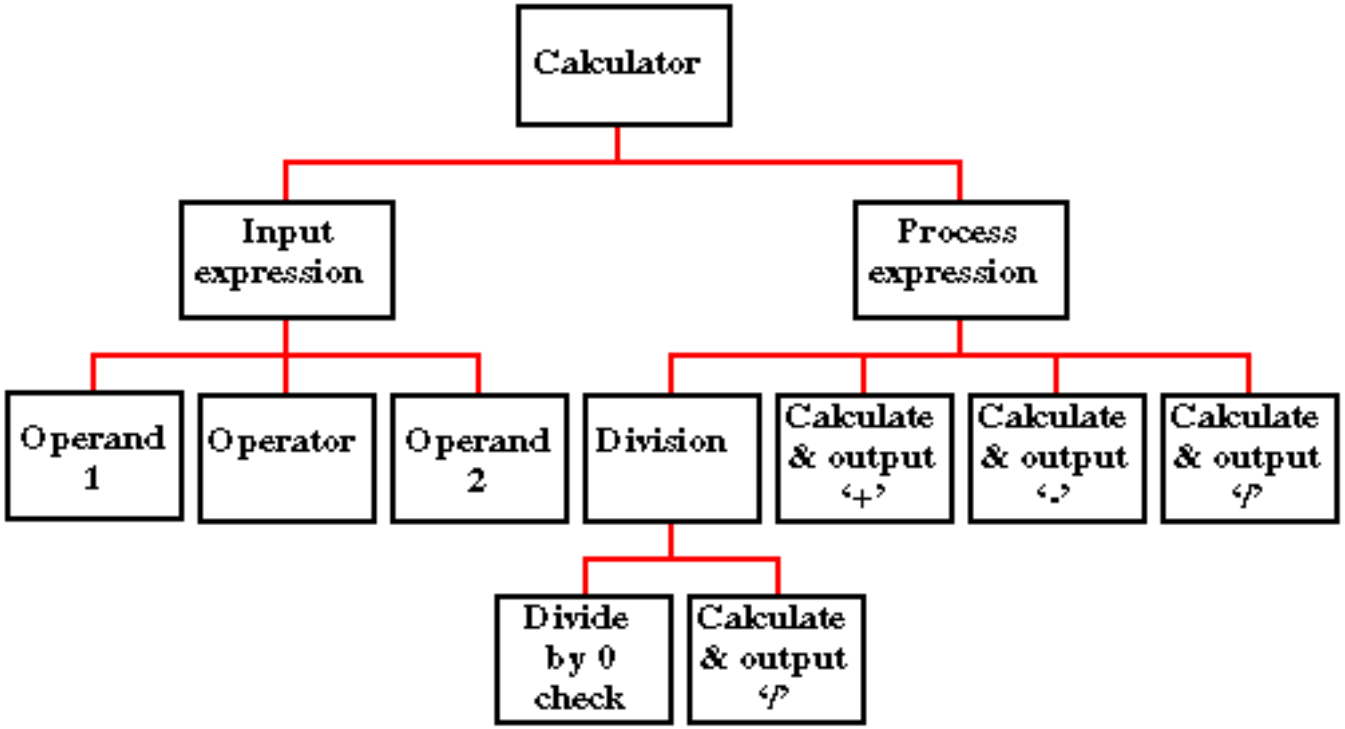
Structured/Procedural Paradigm

- Focus on procedures and functions.
- Design the system by decomposing it based on the processes and functions.
- Top-down algorithmic decomposition.
- This approach separates data from procedures.


Top level
Level 2
Level 2
Level 2
Level 3
Level 3
Level 3
Level 3
Level 3
Level 3
Level 3
Level 4
Level 4
Level 4
Level 4
Level 4
Level 5
Level 5
Level 5
Level 5
Level 5
Level 5

Example:


Temperature conversion
Selection Menu
Fahrenheit to Centigrade
Centigrade to Fahrenheit
Input Fahrenheit
Convert and output Centigrade
Input Centigrade
Convert and output Fahrenheit


Calculator
Input expression
Process expression
Operand 1
Operator
Operand 2
Division
Calculate & output '+'
Calculate & output '-'
Calculate & output '*'
Divide by 0 check
Calculate & output '/'

Drawbacks

- Interdependencies between various functions and processes.
- Cannot be reused easily.
- Data related to each function is not attached.

Object-Oriented Paradigm

- Describing the software solution in terms of collaborating objects, with responsibilities.
  - Objects,
  - classes,
  - encapsulation,
  - States,
  - inheritance,
  - composition,
  - polymorphism

Object-Oriented Design

- Bottom-up
  - Encapsulate data and procedures in objects and classes.
  - Refinement in classes lead to a composed larger system.

Benefits

- Enjoys all the benefits of Modular approach
- Dependencies can be handled by finding the commonalities through inheritance and polymorphism.

- Naturalness
  - OO paradigm models the real world better because everything in real world is an object.
- Reusability
  - Using the existing classes or components in future design, without much effort.

Case Study : Fire Alarm

- The owner of a large multi-stored building wants to have a computerized fire alarm system for his building.
- Smoke detectors and fire alarms would be placed in each room of the building.
- The fire alarm system would monitor the status of these smoke detectors.

- Whenever a fire condition is reported by any of the smoke detectors, the fire alarm system should determine the location at which the fire condition is reported by any of the smoke detectors, the fire alarm system should determine the location at which the fire condition has occurred and then sound the alarms only in the neighbouring locations.

- The fire alarm system should also flash an alarm message on the computer console. Fire fighting personnel man the console round the clock.

- After a fire condition has been successfully handled, the fire alarm system should support resetting the alarms by the fire fighting personnel.

Function-Oriented Approach

- **/* Global data (system state) accessible by various functions */**
- BOOL detector_status[MAX_ROOMS];
- int detector_locs[MAX_ROOMS];
- BOOL alarm_status[MAX_ROOMS];
  - /* alarm activated when status is set */
- int alarm_locs[MAX_ROOMS];
  - /* room number where alarm is located */
- int neighbor-alarm[MAX_ROOMS][10];
  - /* each detector has atmost 10 neighboring locations */

Function-Oriented Approach

The functions which operate on the system state are:

- interrogate_detectors();
- get_detector_location();
- report_fire_location();
- determine_neighbor();
- ring_alarm();
- reset_alarm();

Object-Oriented Approach

- class detector
  - attributes
    - status, location, neighbors
  - operations
    - create, sense-status, get-location, find-neighbors
- class alarm
  - attributes
    - location, status
  - operations
    - create, ring-alarm, get_location, reset-alarm

Structured vs. OO Paradigm

- In **structured design**, data and functions are kept separately.
- Usually all of the data are placed before any of the functions are written.
- Sometimes, it is not intuitively known which data works with which function.

Structured vs. OO Paradigm

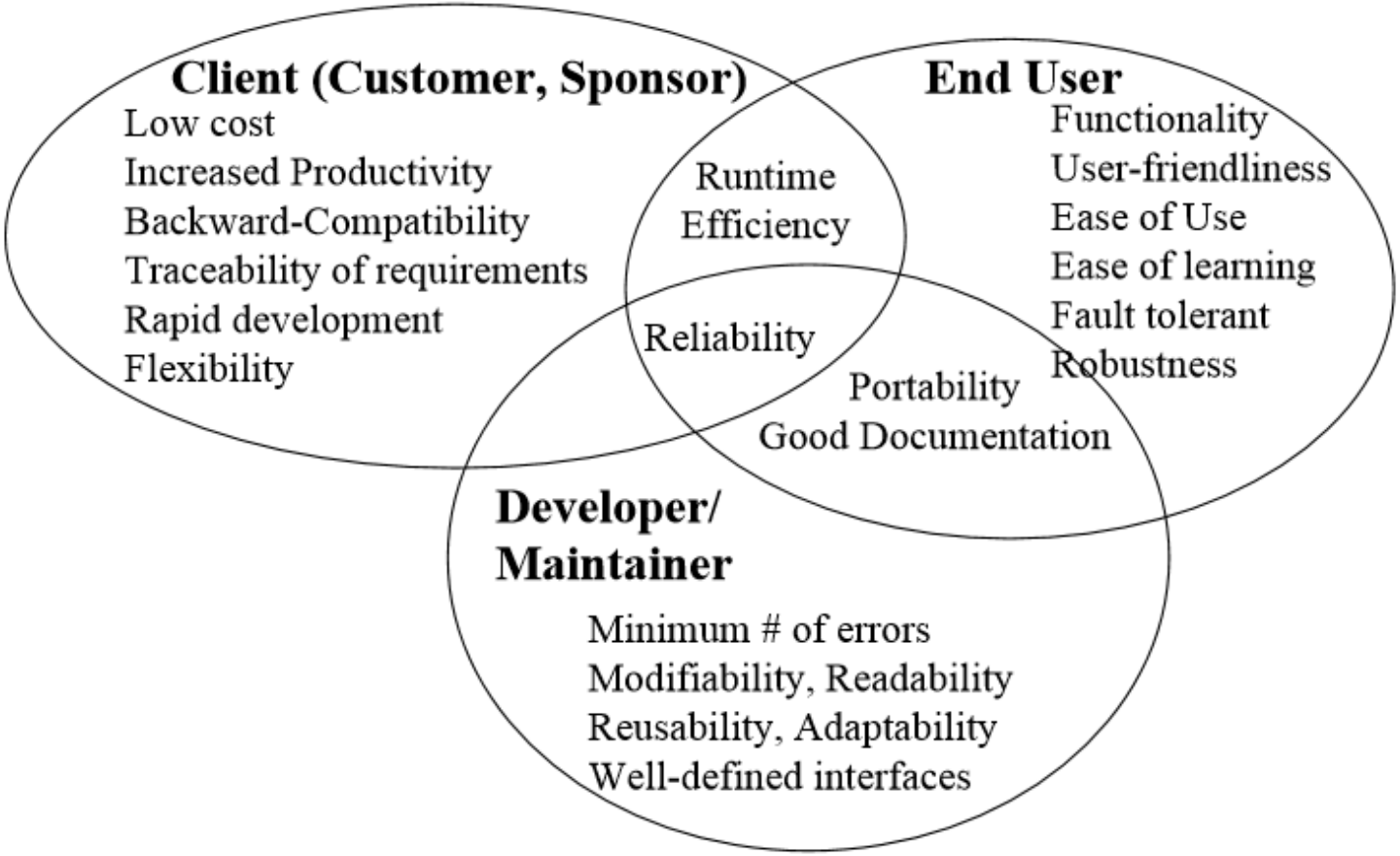
- In **OO design**, the related data and functions of an object are placed together within one unit.
- In Structured approach abstraction exists in functions unlike OO approach where abstraction is in terms of objects

Design Goals and trade-offs

List of Design Goals

- Reliability
- Modifiability
- Maintainability
- Understandability
- Adaptability
- Reusability
- Efficiency
- Portability
- Traceability of requirements
- Fault tolerance
- Backward-compatibility
- Cost-effectiveness
- Robustness
- High-performance
- Good documentation
- Well-defined interfaces
- User-friendliness
- Reuse of components
- Rapid development
- Minimum # of errors
- Readability
- Ease of learning
- Ease of remembering
- Ease of use
- Increased productivity
- Low-cost
- Flexibility

Relationship Between Design Goals



Typical Design Trade-offs

- Functionality vs. Usability
- Cost vs. Robustness
- Efficiency vs. Portability
- Rapid development vs. Functionality
- Cost vs. Reusability

Object-Oriented Analysis and Design

Applying UML

- UML is just a standard diagramming notation.
- It is just a tool, not a skill that is valuable in itself.
- Knowing UML helps you communicate with others in creating software, but the real work in this course is learning Object-Oriented Analysis and Design, not how to draw diagrams.

Assigning Responsibilities

- The most important skill in Object-Oriented Analysis and Design is assigning responsibilities to objects.
- That determines how objects interact and what classes should perform what operations.

Requirements Analysis

- All Software Analysis and Design is preceded by the analysis of requirements.
- One of the basic principles of good design is to defer decisions as long as possible. The more you know before you make a design decision, the more likely it will be that the decision is a good one.
- TFCL: ***Think First, Code Later!***

Use Cases

- Writing Use Cases is not a specifically Object Oriented practice.
- But ***it is a best practice for elaborating and understanding requirements***. So we will study Use Cases.

What Is Analysis and Design?

- **Analysis** emphasizes an *investigation* of the problem and requirements, rather than a solution.
- "Analysis" is a broad term, best qualified, as in *requirements analysis* (an investigation of the requirements) or *object analysis* (an investigation of the domain objects).

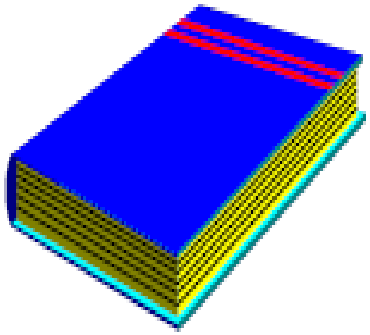
What is Object Oriented Analysis (OOA)?

- The emphasis is on finding and describing the objects (or concepts) in the problem domain.
- In a Library Information System, some of the concepts include *Book*, *Library*, and *Patron*.
- In the case of the flight information system, some of the concepts include *Plane*, *Flight*, and *Pilot*.

What is Object Oriented Design (OOD)?

- The emphasis is defining software objects and how they collaborate to fulfill the requirements.
- In a Library Information System, a *Book* software object may have a *title* attribute and a *get Chapter* method.
- In the flight information system, a *Plane* software object may have a *tailNumber* attribute and a *getFlightHistory* method.

domain concept



Book

title

visualization of domain concept

representation in an object-oriented programming language

```
public class Book
{
private String title;

public Chapter getChapter(int) {...}
}
```

domain concept

Plane

tailNumber

visualization of domain concept

representation in an object-oriented programming language

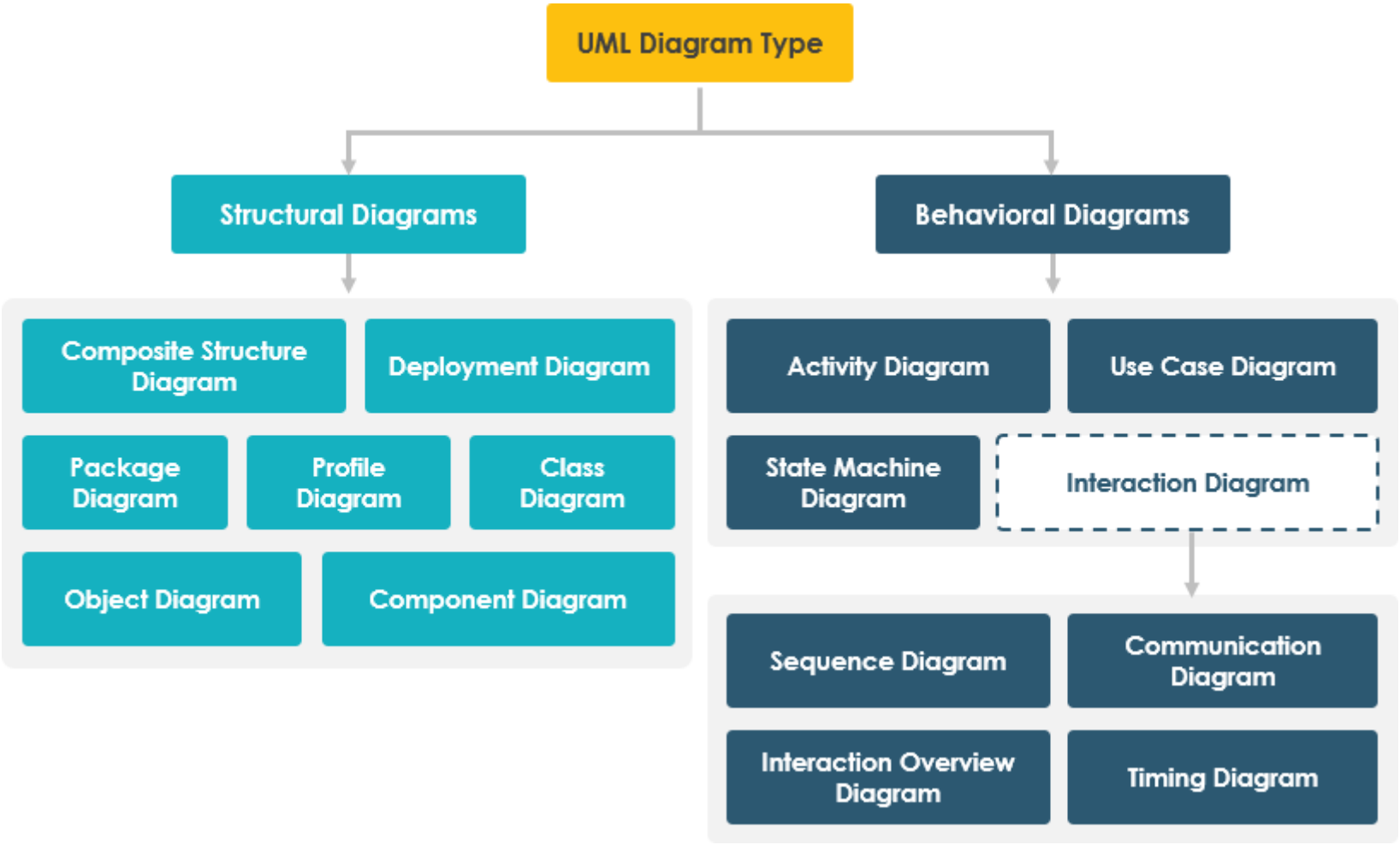
```
public class Plane
{
private String tailNumber;

public List getFlightHistory() {...}
}
```

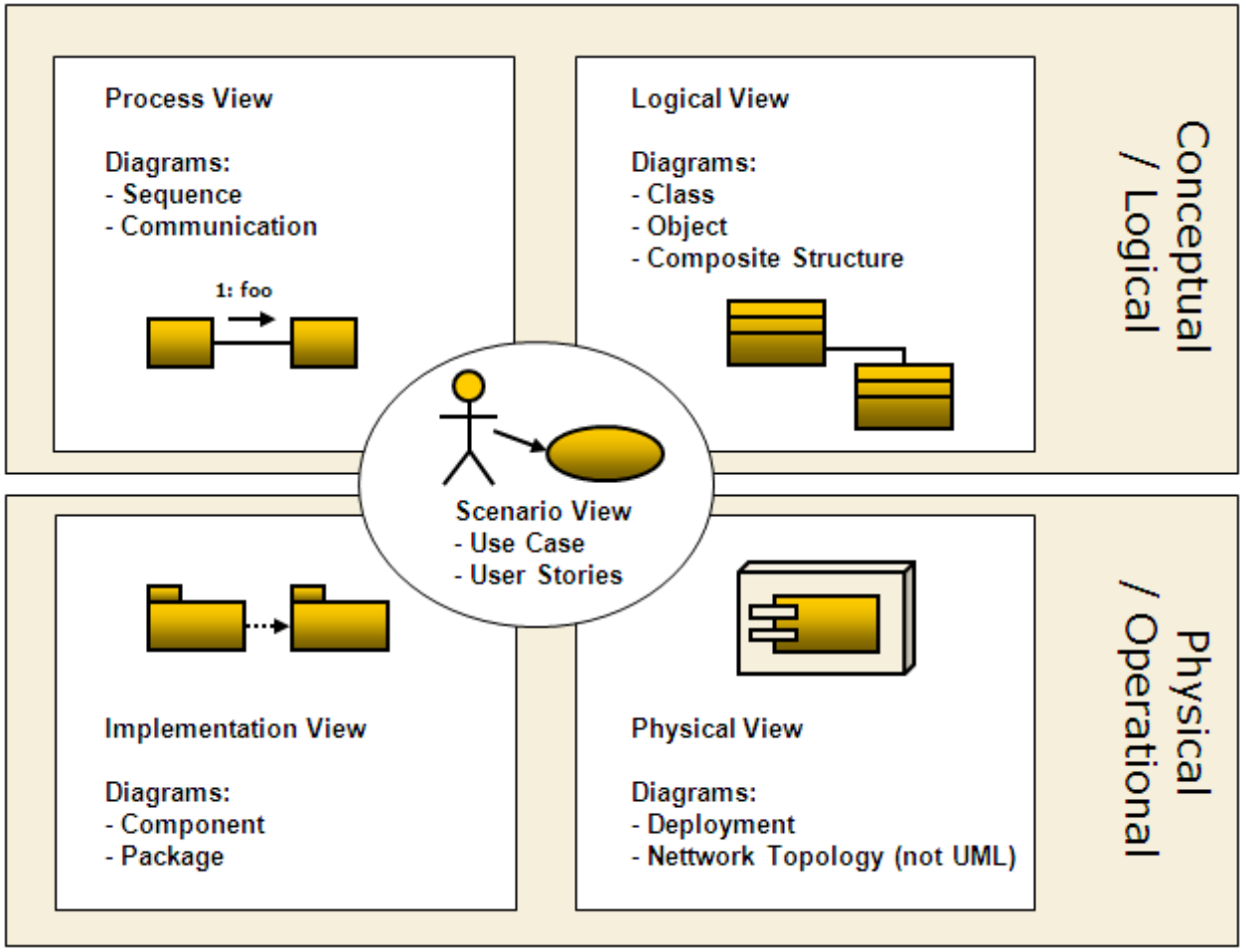
Implementation

- During *Implementation*, or *Object-Oriented Programming*, design objects are implemented, such as a book class in Java.
- Implementation is also known as *Coding* or *Construction*.

Diagram Types



4+ 1 View Model



The Case Study

- The text uses the development of a Point of Sale (POS) System as a case study to demonstrate object-oriented software development.
- A POS system is a replacement for a cash register that adds many computer services to the traditional cash register. Most retail stores now use POS systems.

Concepts of OOP

Terminology & concepts

- **Object**
- **Attributes**
- **Instantiations**
- **Classes**

Objects, Attributes, & Instances

Object – something that is or is capable of being seen, touched, or otherwise sensed, and about which users store data and associate behavior.

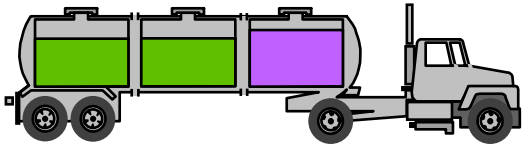
- Person, place, thing, or event
- Employee, customer, instructor, student
- Warehouse, office, building, room
- Product, vehicle, computer, videotape

Attribute – the data that represent characteristics of interest about an object.

Object instance – each specific person, place, thing, or event, as well as the values for the attributes of that object.

What is an Object?

- Informally, an object represents an entity, either physical, conceptual, or software
  - Physical entity

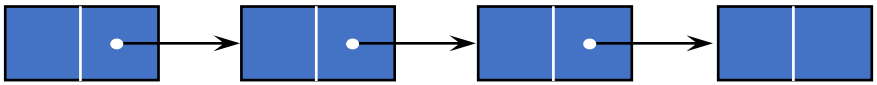


Truck

  - Conceptual entity

Chemical Process

  - Software entity



Linked List

Representing Objects

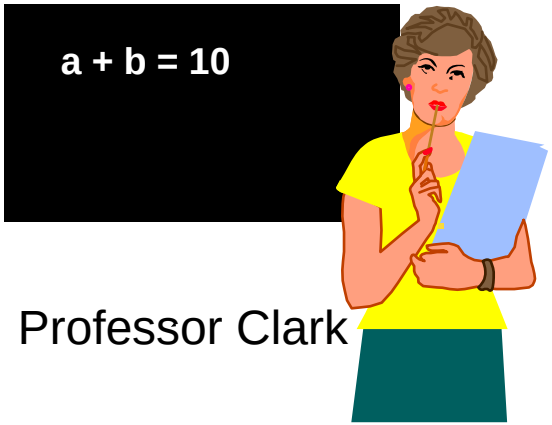
- An object is represented as rectangles with underlined names

: Professor

Class Name Only

ProfessorClark

Object Name Only



ProfessorClark : Professor

Class and Object Name

What is a Class?

- A class is a description of a group of objects with common properties (attributes), behavior (operations), relationships, and semantics
  - An object is an instance of a class
  - Emphasizes relevant characteristics

Sample Class

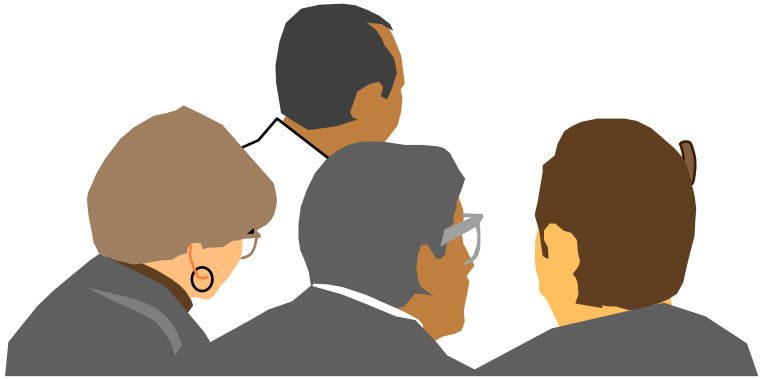
Class

Course

Properties

Name
Location
Days offered
Credit hours
Start time
End time

a + b = 10

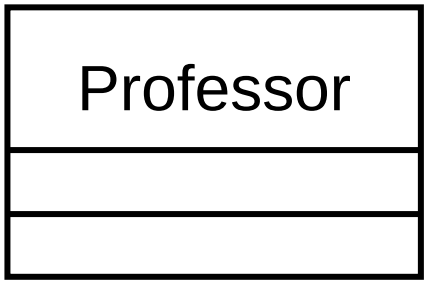


Behavior

Add a student
Delete a student
Get course roster
Determine if it is full

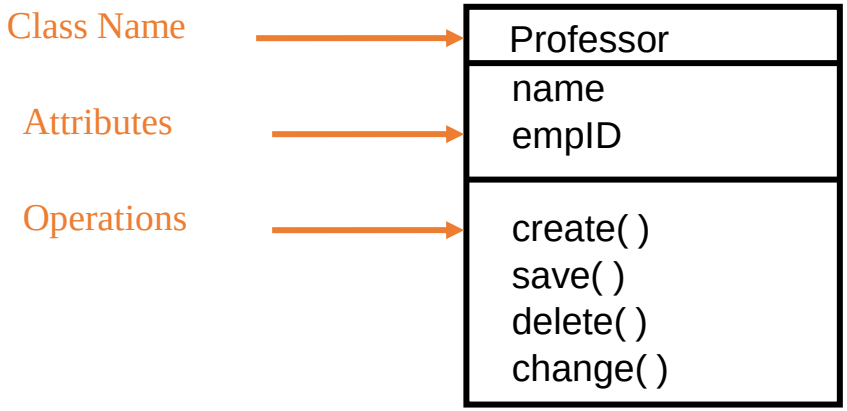
Representing Classes

- A class is represented using a compartmented rectangle



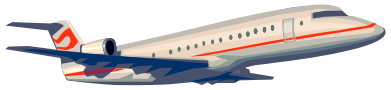
Class Compartments

- A class is comprised of three sections
  - The first section contains the class name
  - The second section shows the structure (attributes)
  - The third section shows the behavior (operations)

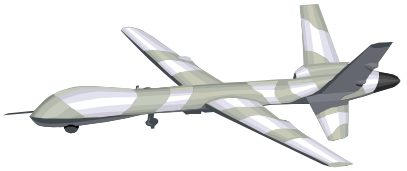


Exercise: How many classes could you find here?

VTOL

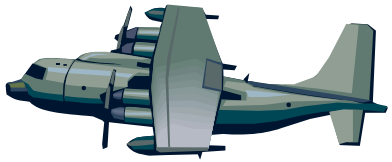


Jet



Drone

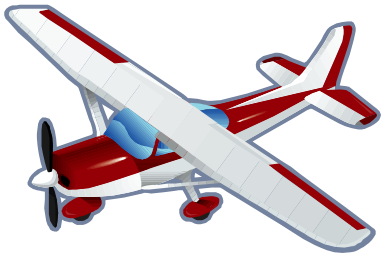
Glider



Military

Space Shuttle

Helicopter



Turboprop

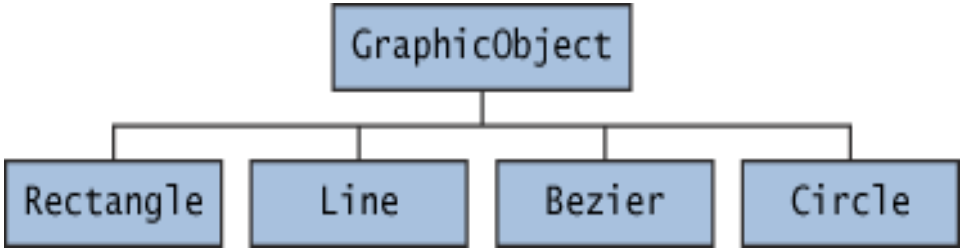
Airship

Object Orientation

- Abstraction
- Encapsulation
- Modularity
- Inheritance

What is Abstraction?

- Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency.
- Abstract classes may not be instantiated, and require subclasses to provide implementations for the abstract methods.

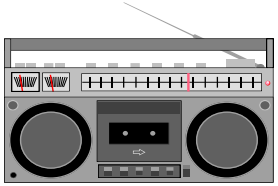


What is Abstraction?

Customer

Salesperson

Not saying Which salesperson – just a salesperson in general!!!

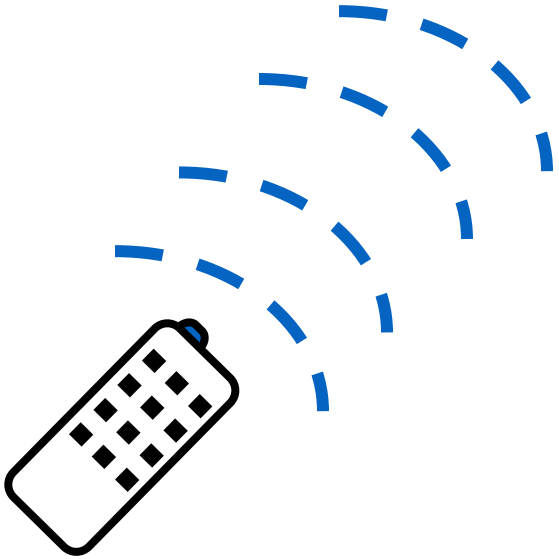


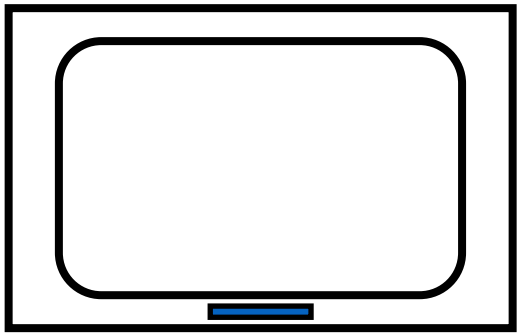
Product

Manages Complexity

What is Encapsulation?

- Hide implementation from clients
  - Clients depend on interface



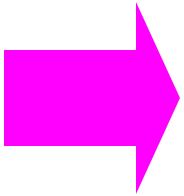


How does an object encapsulate?
What does it encapsulate?

What is Modularity?

- The breaking up of something complex into manageable pieces

Order Processing System



Order Entry

Order Fulfillment

Billing

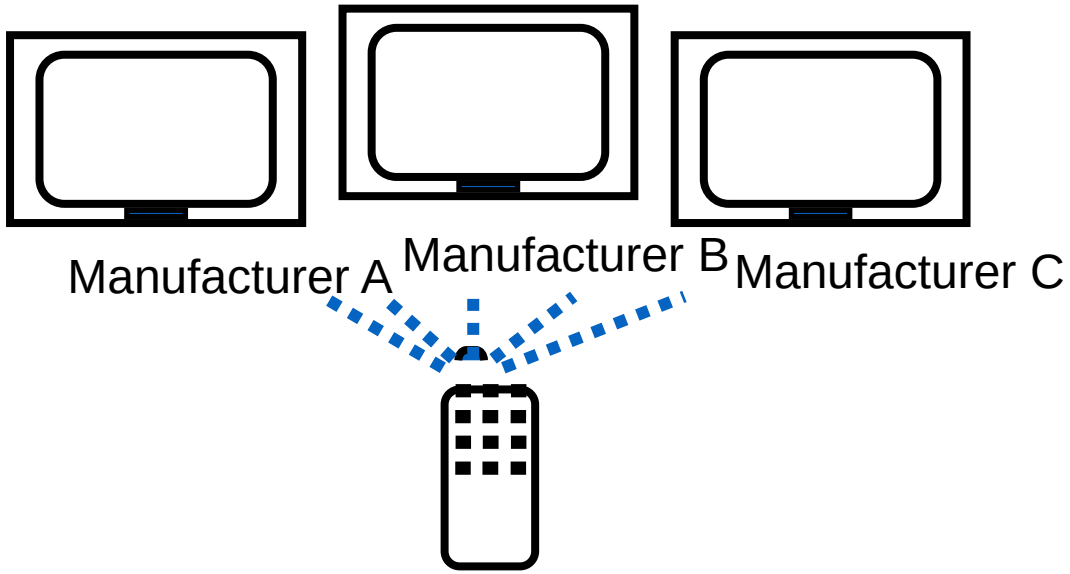
Manages Complexity

Polymorphism

- Polymorphism means “Many Form”
- Two objects are polymorphic if they have the same interface and different behavior.
- This allows clients to use them without knowing their true behavior.

What is Polymorphism?

- The ability to hide many different implementations behind a single interface

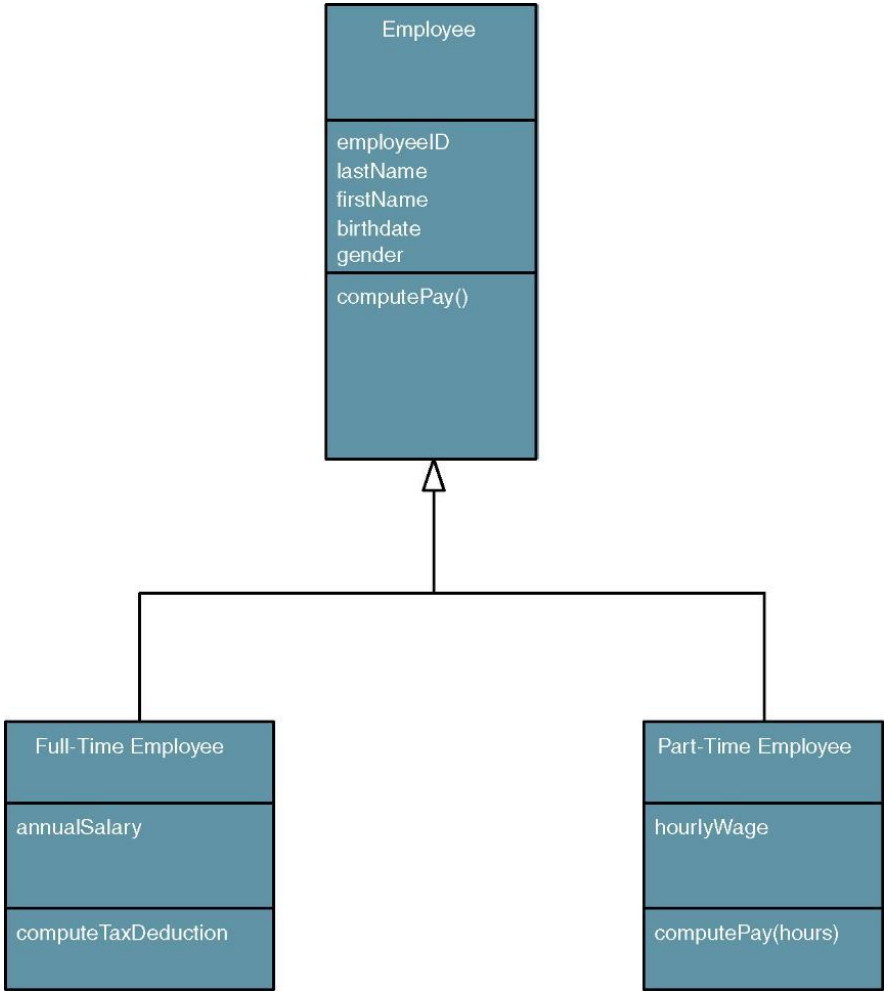


OO Principle: Encapsulation

Polymorphism

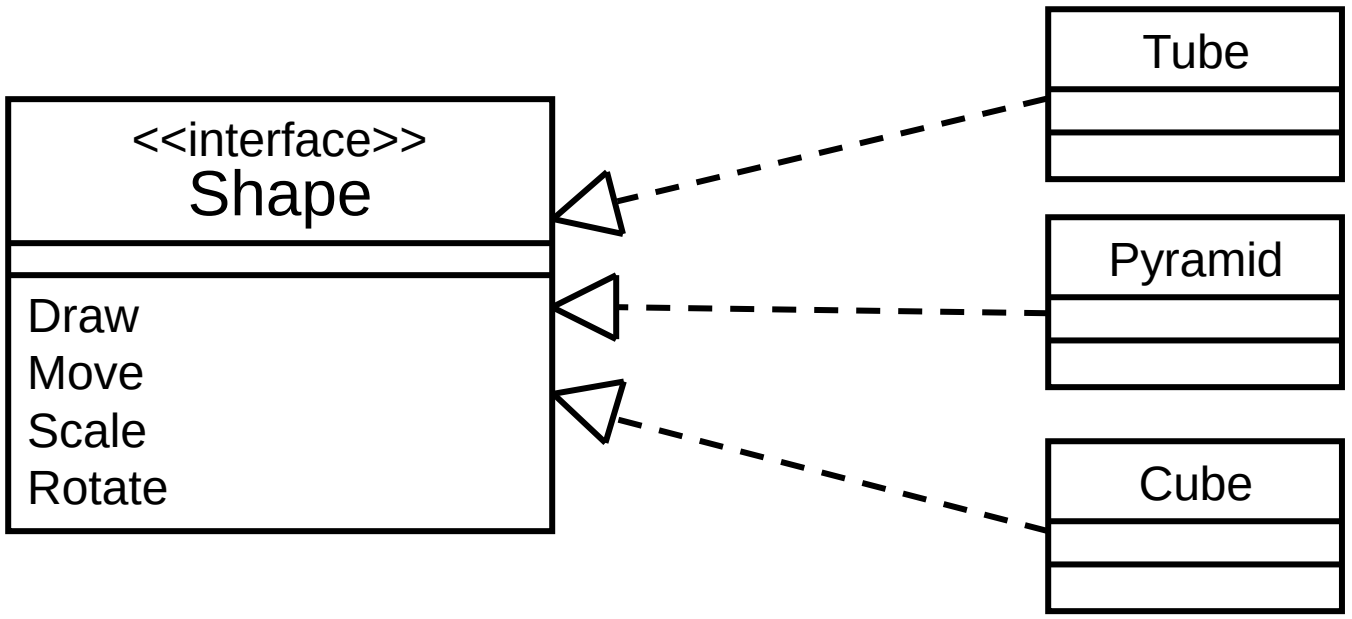
Polymorphism – literally meaning "many forms," the concept that different objects can respond to the same message in different ways.

Override – a technique whereby a subclass (subtype) uses an attribute or behavior of its own instead of an attribute or behavior inherited from the class (supertype).



What is an Interface?

- Interfaces formalize polymorphism
- Interfaces support “plug-and-play” architectures



Supertype, and Subtype

Generalization/specialization – a technique wherein the attributes and behaviors that are common to several types of object classes are grouped (or abstracted) into their own class, called a *supertype*. The attributes and methods of the supertype object class are then inherited by those object classes.

Supertype – an entity that contains attributes and behaviors that are common to one or more class subtypes.

- Also referred to as *abstract* or *parent* class.

Subtype – an object class that inherits attributes and behaviors from a supertype class and then may contain other attributes and behaviors that are unique to it.

- Also referred to as a *child* class and, if it exists at the lowest level of the inheritance hierarchy, as *concrete* class.

Inheritance (cont.)