SOFTWARE DESIGN AND ANALYSIS
CS 3004

Object Oriented Analysis and Design

Fall 2021
Dr. Muhammad Bilal

Outline

1. Compare and contrast analysis and design.
2. Define object-oriented analysis and design (OOA/D).
3. Illustrate a brief example.

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.

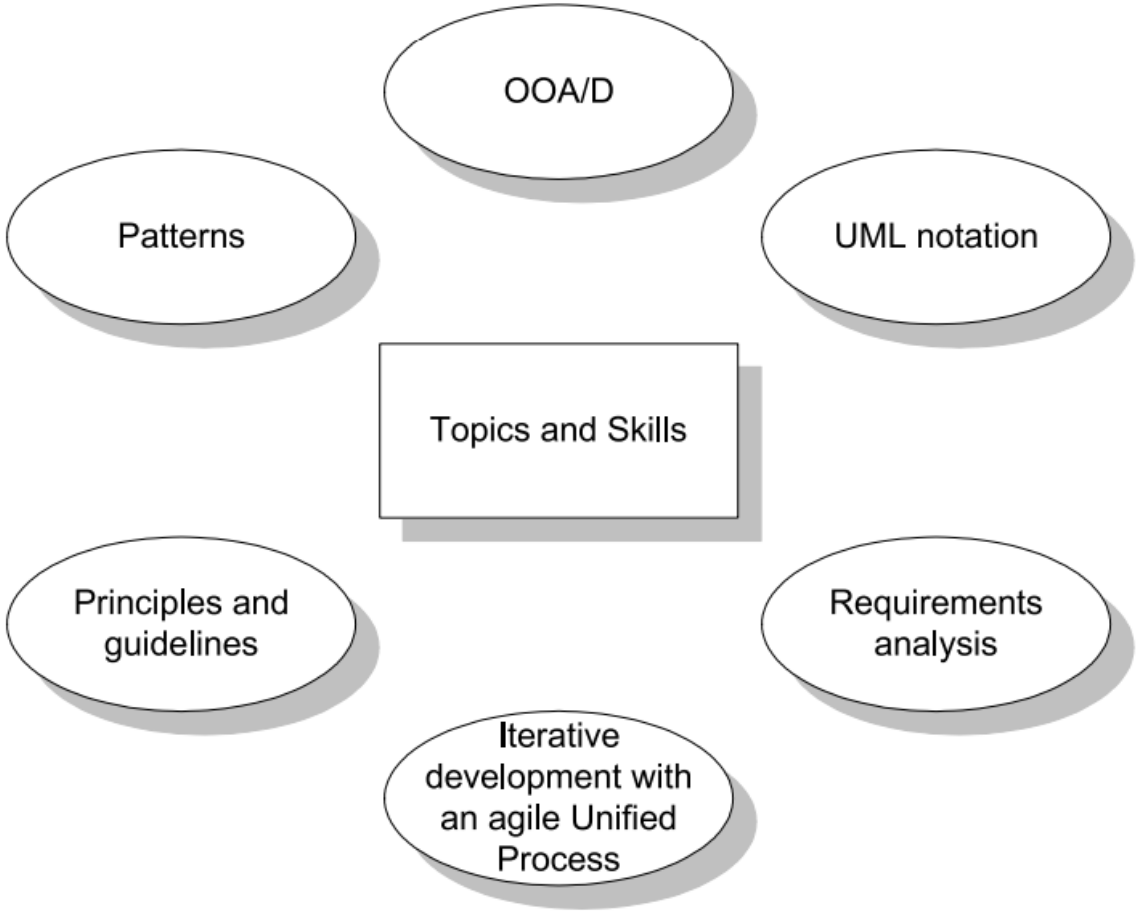
The Unified Process

- A standardized approach to analysis and design helps to ensure that all necessary tasks are understood and completed in software development.
- This text, and the course, will focus on the Unified Process developed at Rational Software by Ivar Jacobsen, Grady Boch, Jim Rumbaugh, and others.

Other Necessary Skills

- Requirements Analysis, Object-Oriented Analysis and Object-Oriented Design are not a complete toolkit for a software developer.
- There are many other skills necessary in Software development, including programming. This course only covers a subset of the necessary skills.

Topics and Skills



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).

- **Design** emphasizes a *conceptual solution* that fulfills the requirements, rather than its implementation.

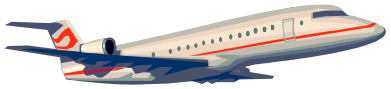
- *do the right thing (analysis), and do the thing right (design).*

What is Object Oriented Analysis (OOA)?

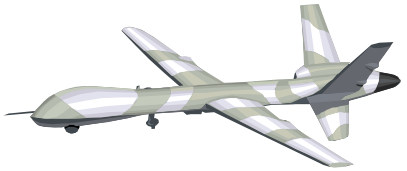
- The emphasis is on finding and describing the objects (or concepts) in the problem domain.
- In a Library Information System, some of the concepts include *Book*, *Library*, and *Patron*.
- In the case of the flight information system, some of the concepts include *Plane*, *Flight*, and *Pilot*.

Exercise: How many classes could you find here?

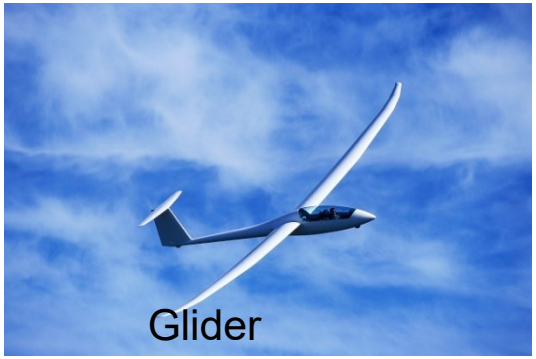
VTOL



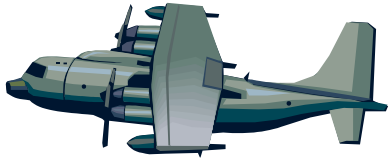
Jet



Drone



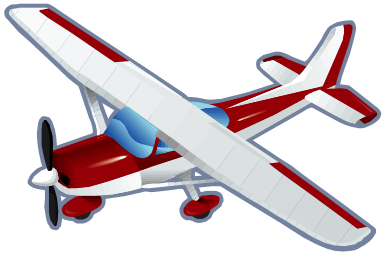
Glider



Military

Space Shuttle

Helicopter



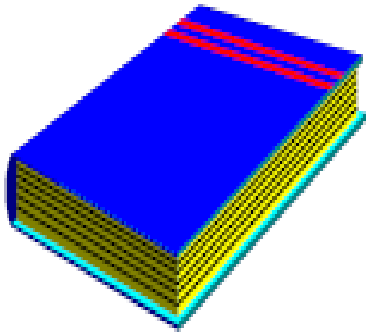
Turboprop

Airship

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*.

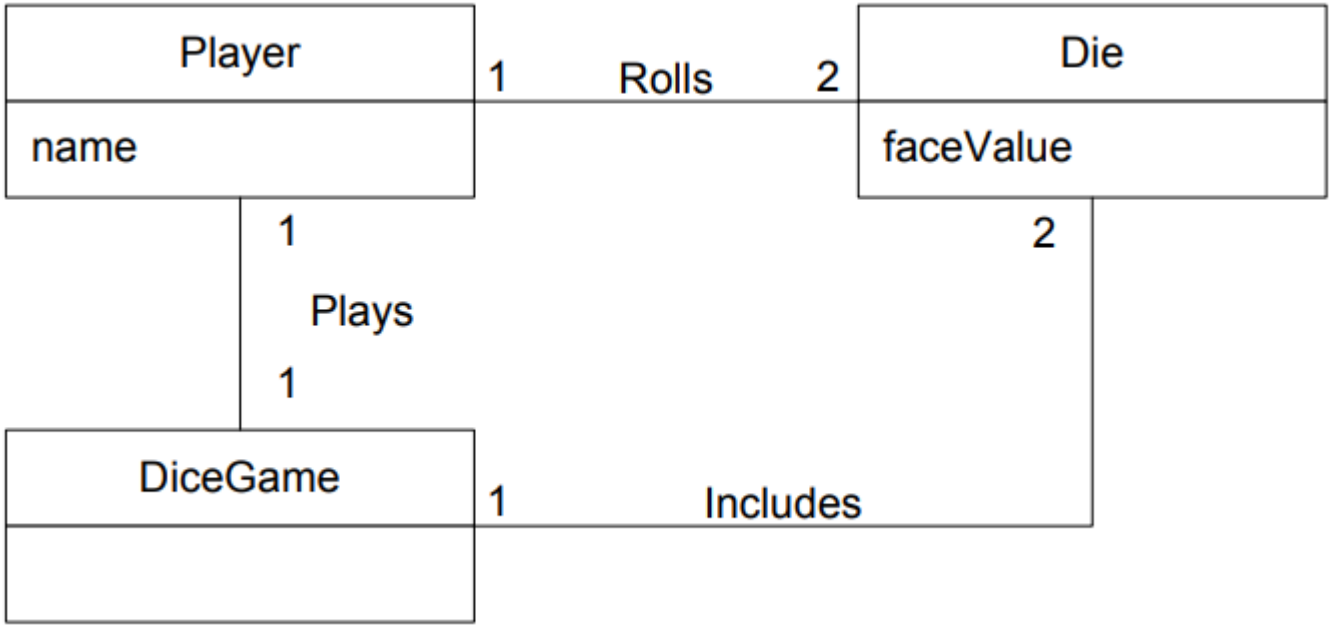
Example Tasks

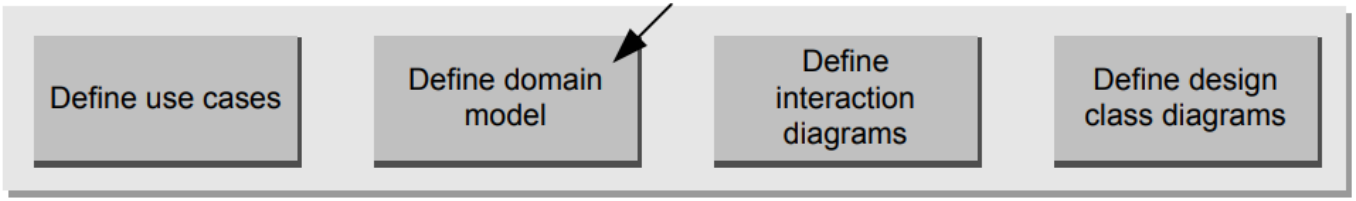
- Define Use Cases
- Define a Domain Model
- Define Interaction Diagrams
- Define Design Class Diagrams

Define Use Cases

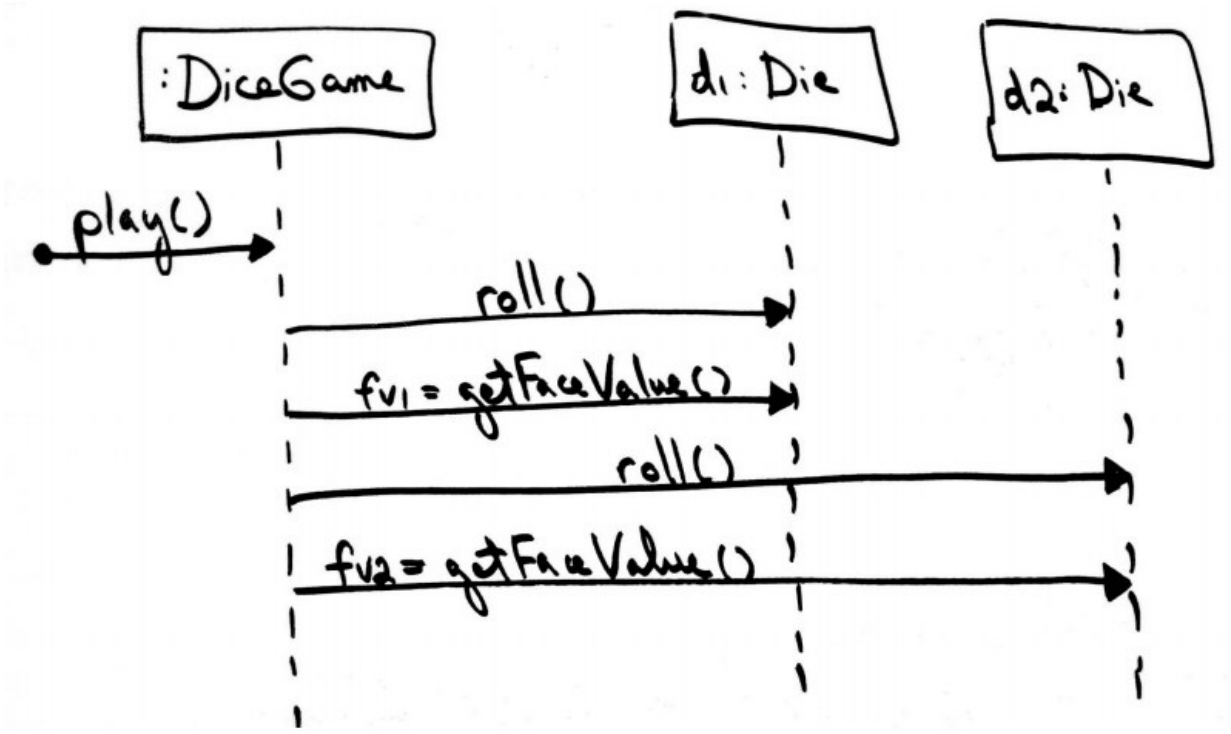
- **Play a Dice Game:** A player picks up and rolls the dice. If the dice face value total seven, they win, otherwise, they lose.

Define a Domain Model

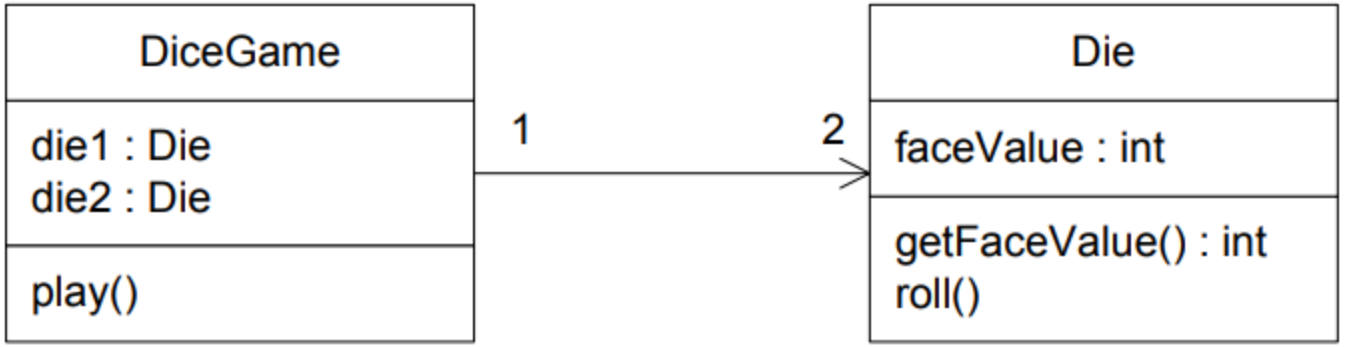


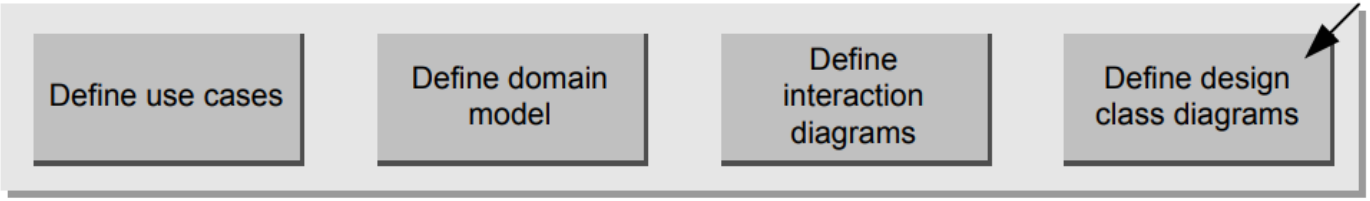


Define Interaction Diagrams



Define Design Class Diagrams

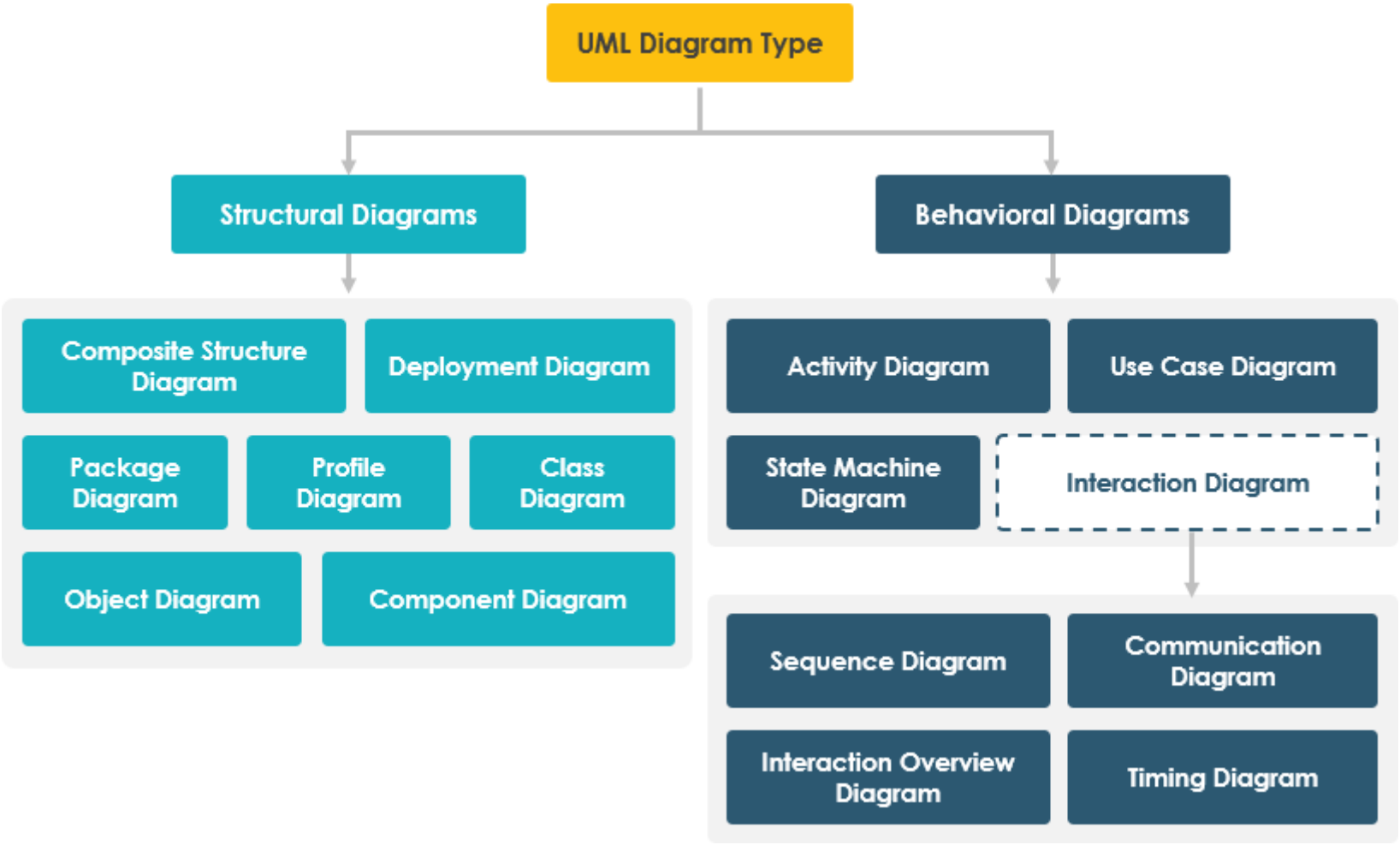




UML

- The Unified Modeling Language (UML) is a language for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems

Diagram Types



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.

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.

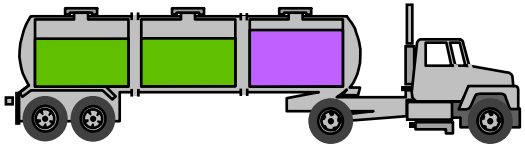
- Person, place, thing, or event
- Employee, customer, instructor, student
- Warehouse, office, building, room
- Product, vehicle, computer, videotape

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

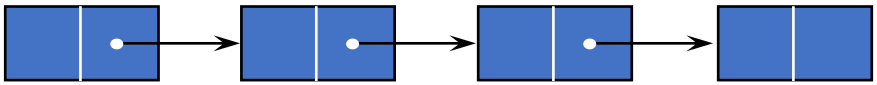


Truck

  - Conceptual entity

Chemical Process

  - Software entity



Linked List

Representing Objects

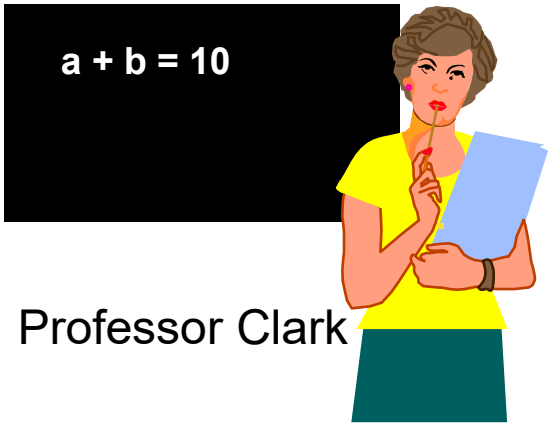
- An object is represented as rectangles with underlined names

: Professor

Class Name Only

ProfessorClark

Object Name Only



ProfessorClark :
Professor

Class and Object Name

What is a Class?

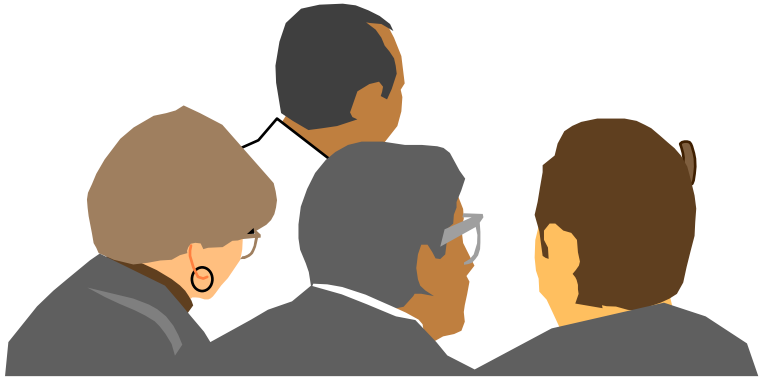
- A class is a description of a group of objects with common properties (attributes), behavior (operations), relationships, and semantics
  - An object is an instance of a class
  - Emphasizes relevant characteristics

Sample Class

Class
Course

Properties
Name
Location
Days offered
Credit hours
Start time
End time

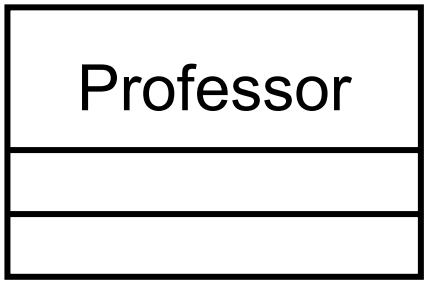
a + b = 10



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

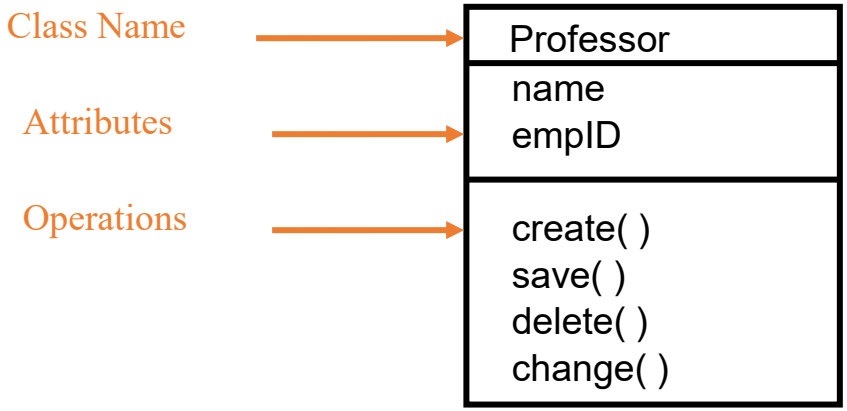
Representing Classes

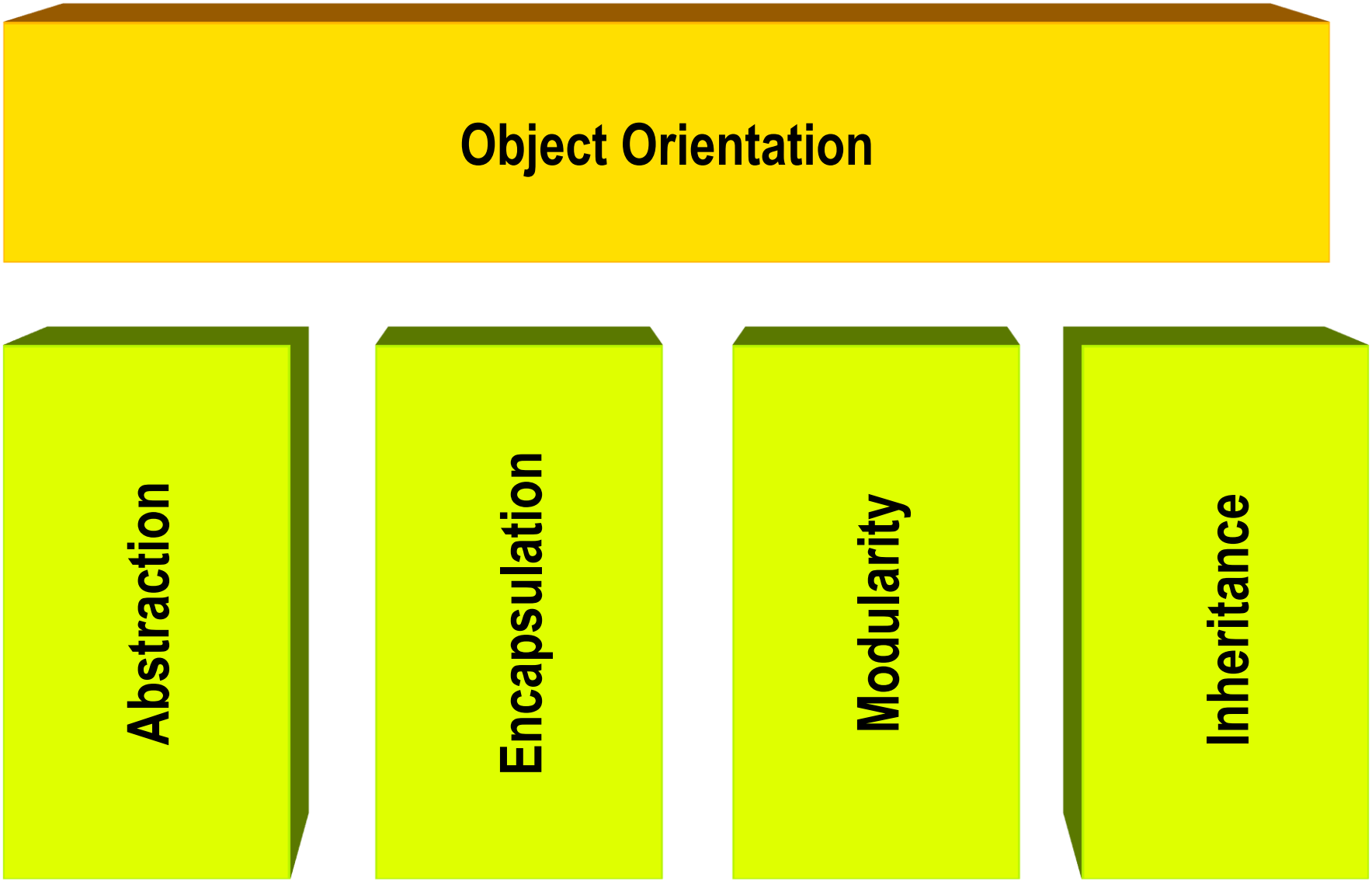
- A class is represented using a compartmented rectangle



Class Compartments

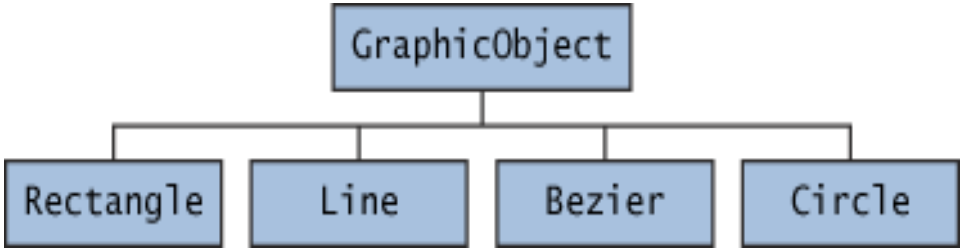
- A class is comprised of three sections
  - The first section contains the class name
  - The second section shows the structure (attributes)
  - The third section shows the behavior (operations)




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.

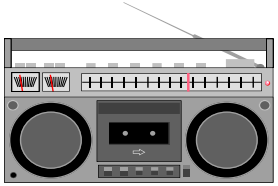


What is Abstraction?

Customer

Salesperson

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

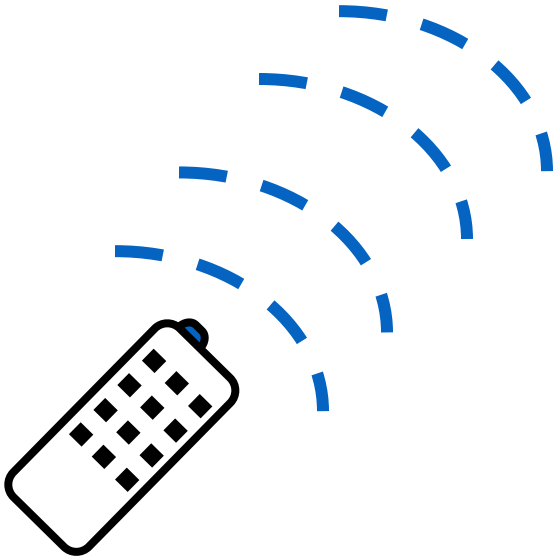


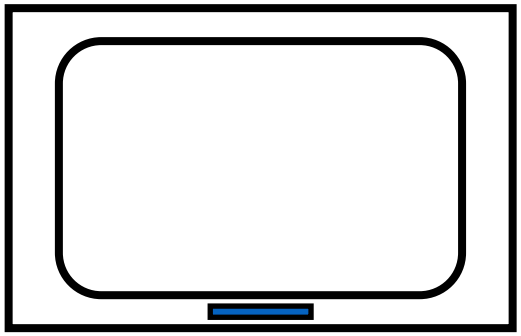
Product

Manages Complexity

What is Encapsulation?

- Hide implementation from clients
  - Clients depend on interface



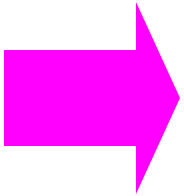


How does an object encapsulate?
What does it encapsulate?

What is Modularity?

- The breaking up of something complex into manageable pieces

Order Processing System



Order Entry

Order Fulfillment

Billing

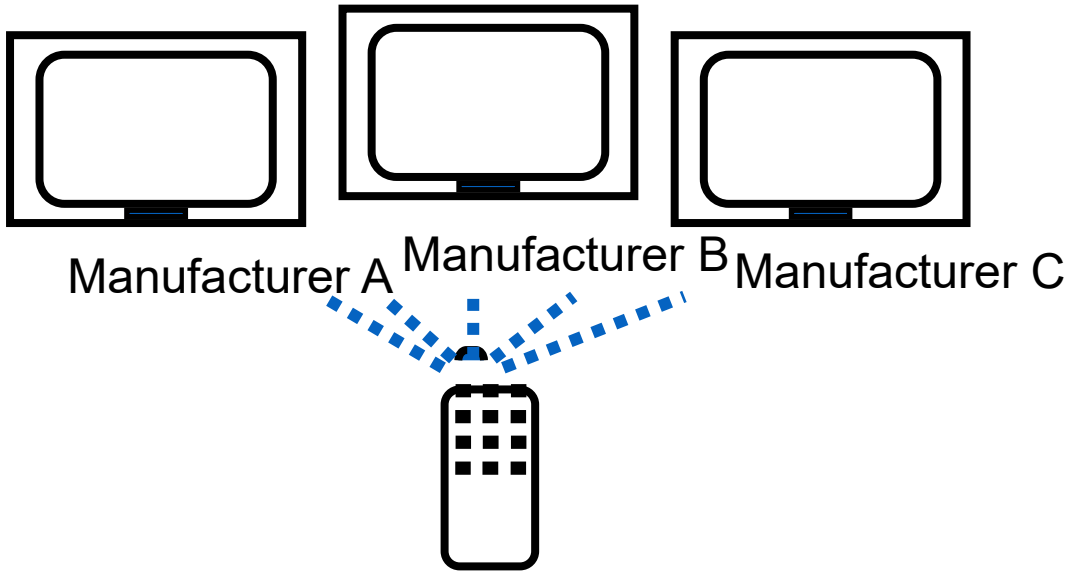
Manages Complexity

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

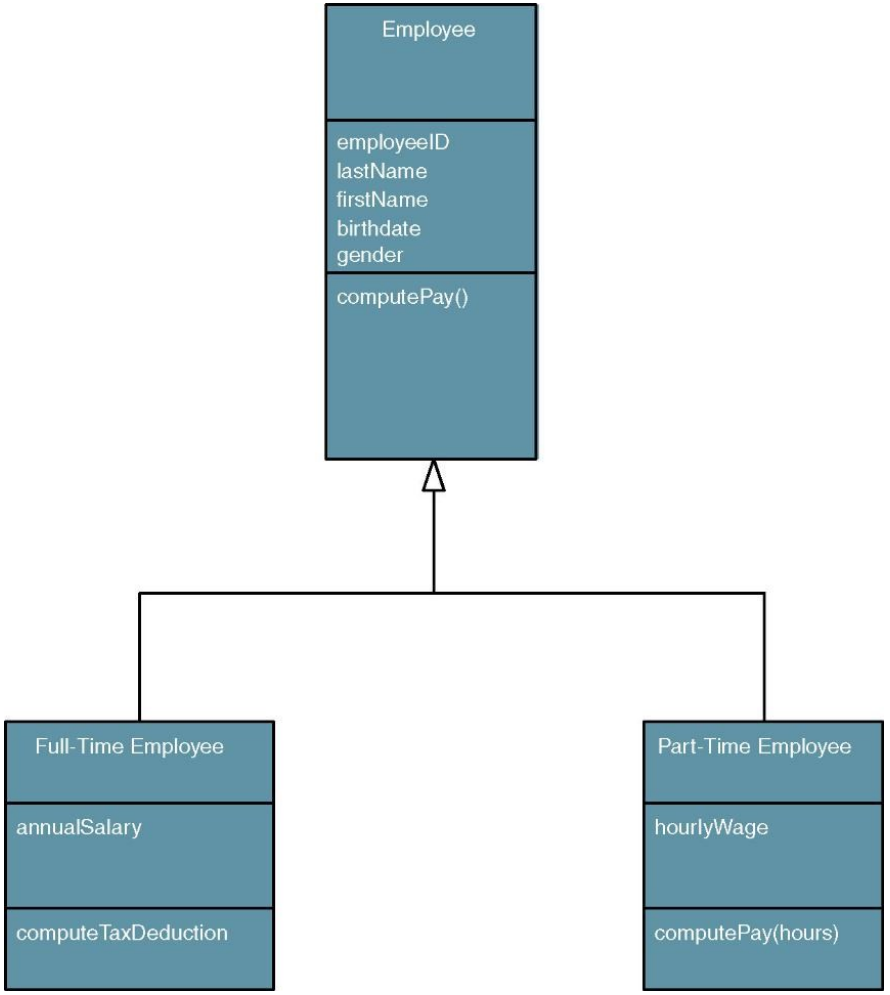


OO Principle: Encapsulation

Polymorphism

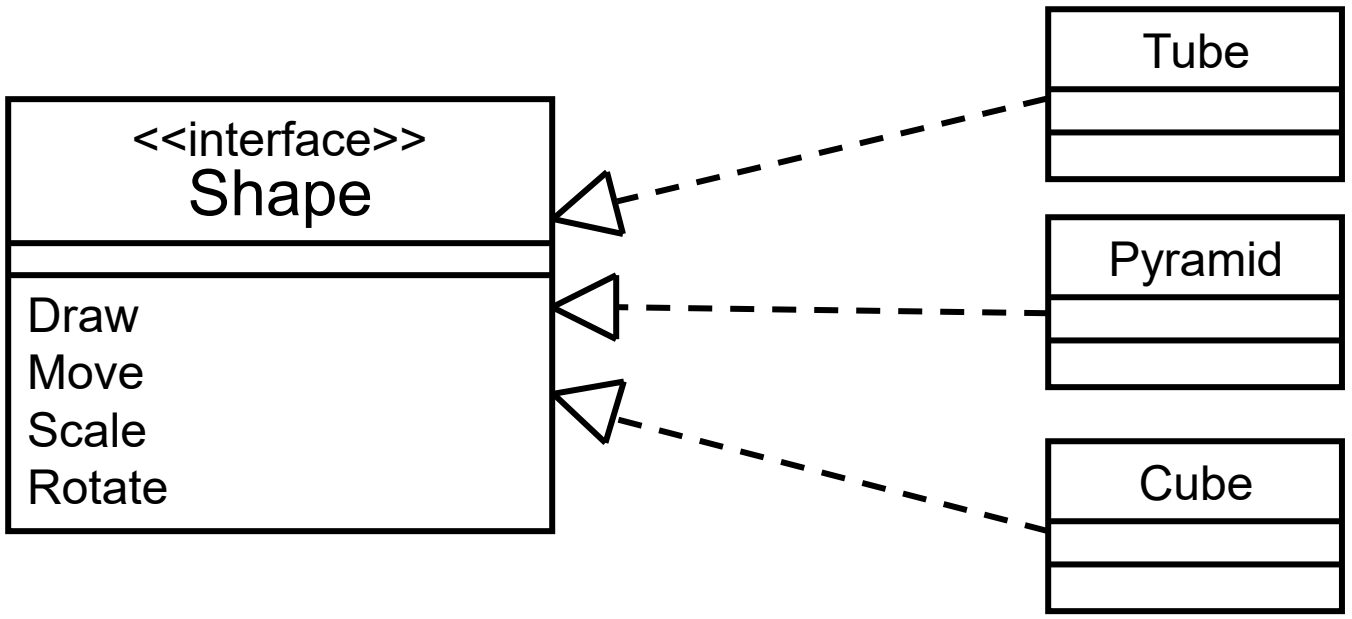
Polymorphism – literally meaning "many forms," the concept that different objects can respond to the same message in different ways.

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.)