

# **SOFTWARE DESIGN**

## **COSC 4353 / 6353**

Dr. Raj Singh



# UML - HISTORY



The Unified Modeling Language (UML) is a general purpose modeling language designed to provide a standard way to visualize the design of a system.



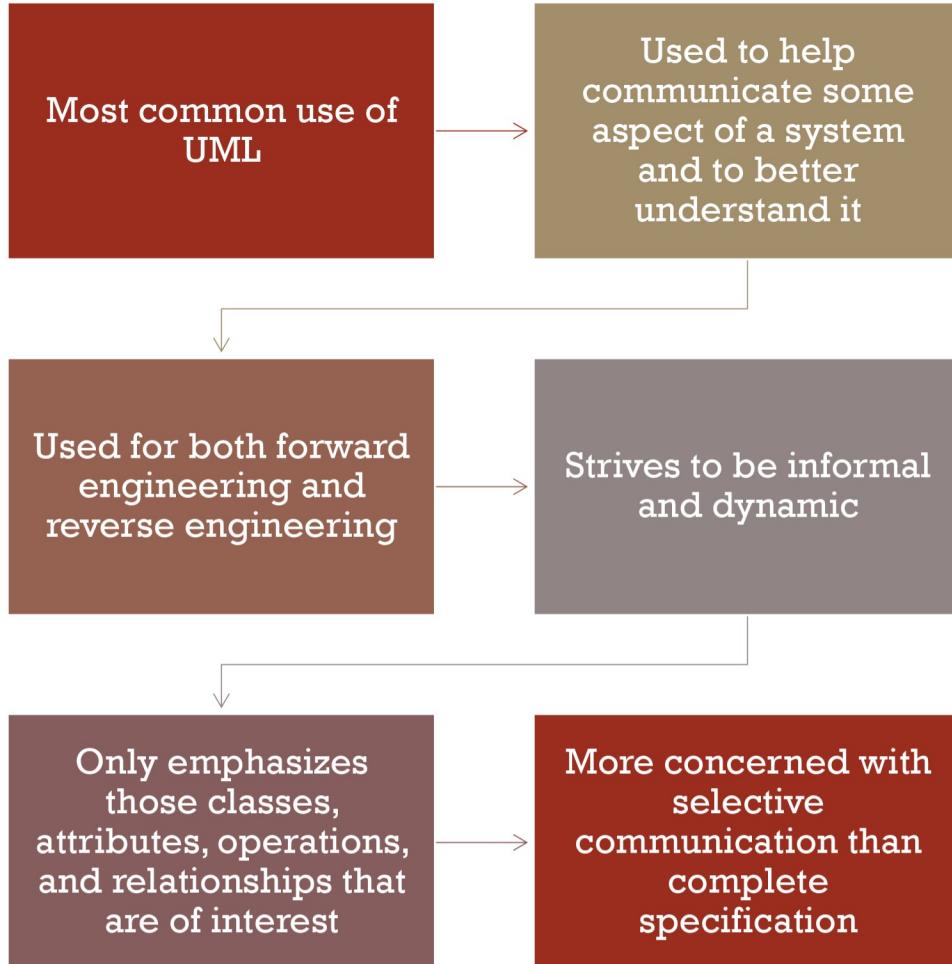
Created and developed by Grady Booch, Ivar Jacobson and James Rumbaugh at Rational Software during 1994–95 with further development led by them through 1996.



In 1997 it was adopted as a standard by the Object Management Group (OMG)



In 2000 UML was also accepted by the International Organization for Standardization(ISO) as an approved ISO standard.



## UML AS A SKETCH

# UML AS A BLUEPRINT



Goal is completeness



It is more definitive, while the sketch approach is more explorative



Used to describe a detailed design for a programmer to follow in writing source code



Notation should be sufficiently complete so that a software engineer can follow it



It can be used by a designer to develop blueprint-level models that show interfaces of subsystems or classes



As a reversed engineered product, diagrams convey detailed information about the source code that is easier for software engineers to understand



Specifies the complete system in UML so that code can be automatically generated



Looks at UML from a software perspective rather than a conceptual perspective



Diagrams are compiled directly into executable code so that the UML becomes the source code



Challenge is making it more productive to use UML rather than some another programming language



Another concern is how to model behavioral logic

Done with interaction diagrams, state diagrams, and activity diagrams

## UML AS A PROGRAMMING LANGUAGE



UML sketches are useful with both forward and reverse engineering and in both conceptual and software perspectives



Detailed forward engineering blueprints are difficult to do well and slow down the development effort

Actual implementation of interfaces will reveal the needs for changes



The value of reversed engineered blueprints depends on the CASE tool

A dynamic browser would be very helpful; a thick document wastes time and resources



UML as a programming language will probably never see significant usage

Graphical forms have not shown to be more productive in writing code than textual code for most programming tasks

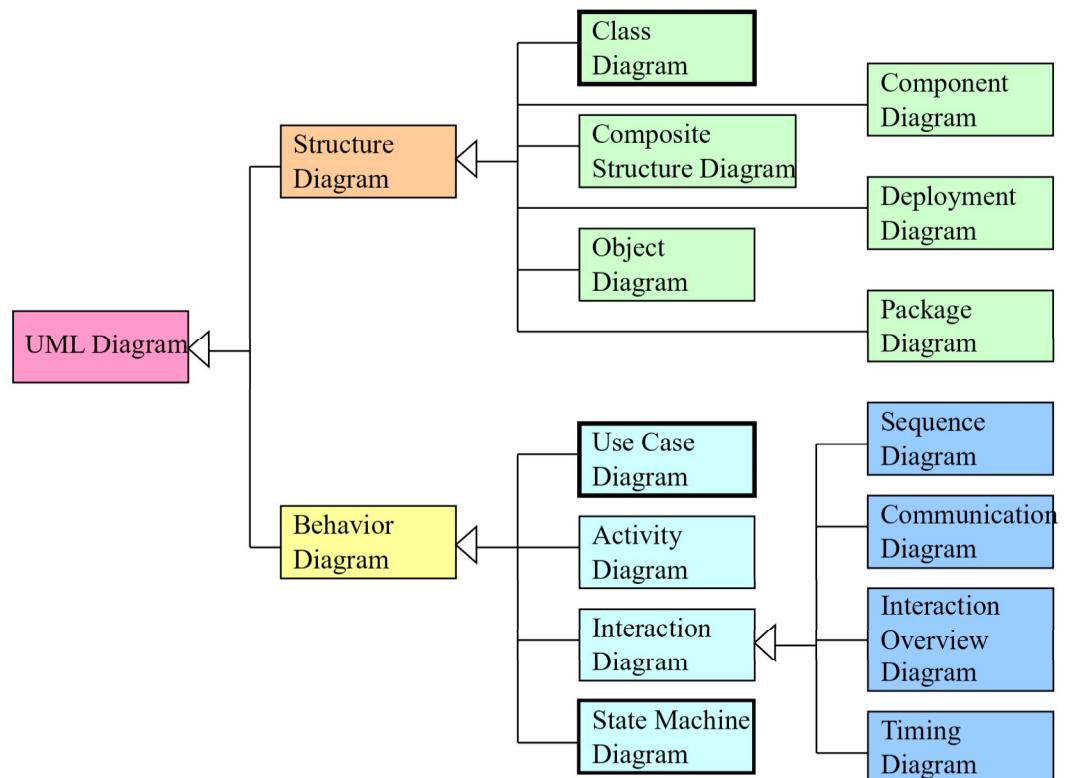
## WAYS OF USING - COMPARISON

# TYPES OF UML DIAGRAMS

DIAGRAM NAME	PURPOSE
Activity	Models procedural and parallel behavior
Class	Models classes, attributes, operations and relationships
Communication	Models interaction between objects
Component	Models structure and connection of components
Composite Structure	Models runtime decomposition of a class
Deployment	Models deployment of artifacts to nodes
Interaction overview	Mixes the sequence and activity diagram

# TYPES OF UML DIAGRAMS (CONTINUED)

DIAGRAM NAME	PURPOSE
Object	Models example configurations of instances
Package	Models compile-time hierarchical structure
Sequence	Models sequence interaction between objects
State Machine	Models how events change an object over its life
Timing	Models timing interaction between objects
Use Case	Models how users interact with a system



# CLASSIFICATION OF DIAGRAM TYPES



Use case / scenario defines how a user uses a system to accomplish a particular goal.



A modeling technique that defines the features to be implemented and the resolution of any errors that may be encountered.



A methodology used in system analysis to identify, clarify, and organize system requirements.



A set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

## SCENARIO-BASED MODELING



What are the main tasks or functions that are performed by the actor?



What system information will the actor acquire, produce or change?



Will the actor have to inform the system about changes in the external environment?

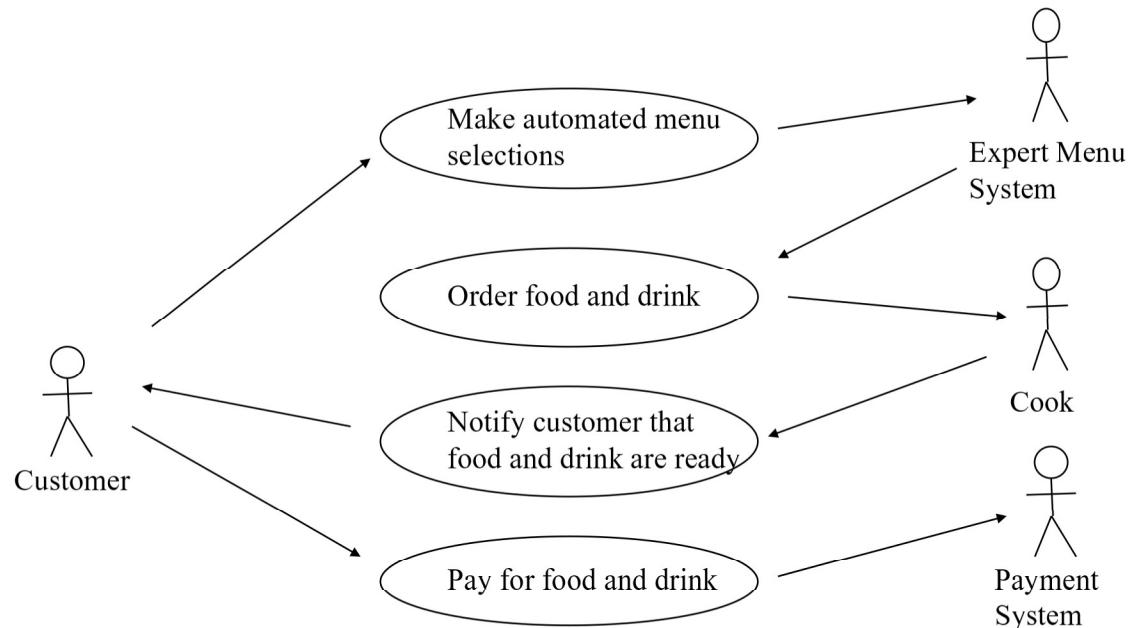


What information does the actor desire from the system?



Does the actor wish to be informed about unexpected changes?

## DEVELOPING A USE-CASE



# USE-CASE DIAGRAM EXAMPLE

# EXCEPTIONS



Describe situations that may cause the system to exhibit unusual behavior



Brainstorm to derive a reasonably complete set of exceptions for each use case



Are there cases where a validation function occurs for the use case?



Handling exceptions may require the creation of additional use cases

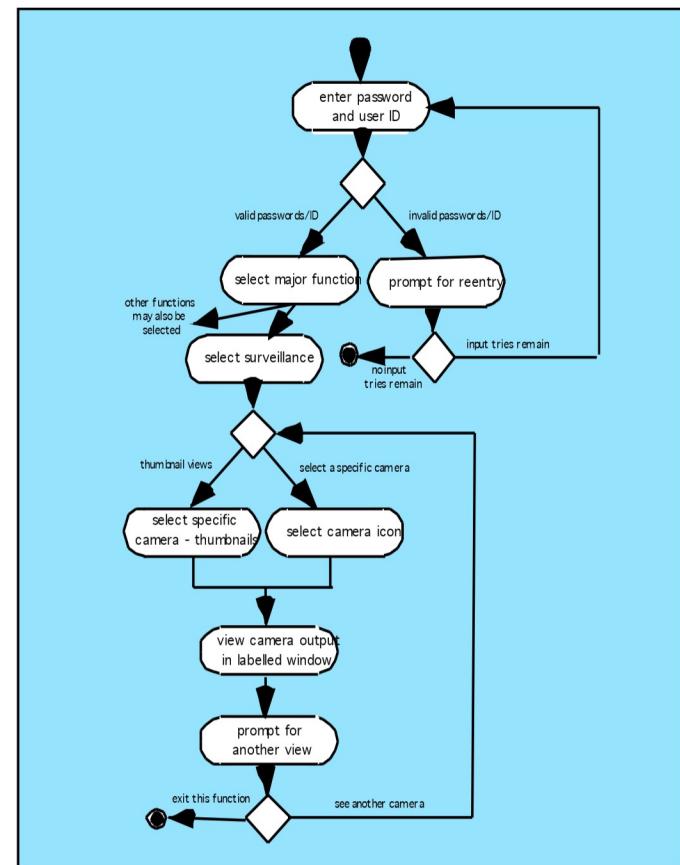
# ACTIVITY DIAGRAM



Supplements the use case by providing a graphical representation



The flow of interaction between actor and the system within a specific scenario



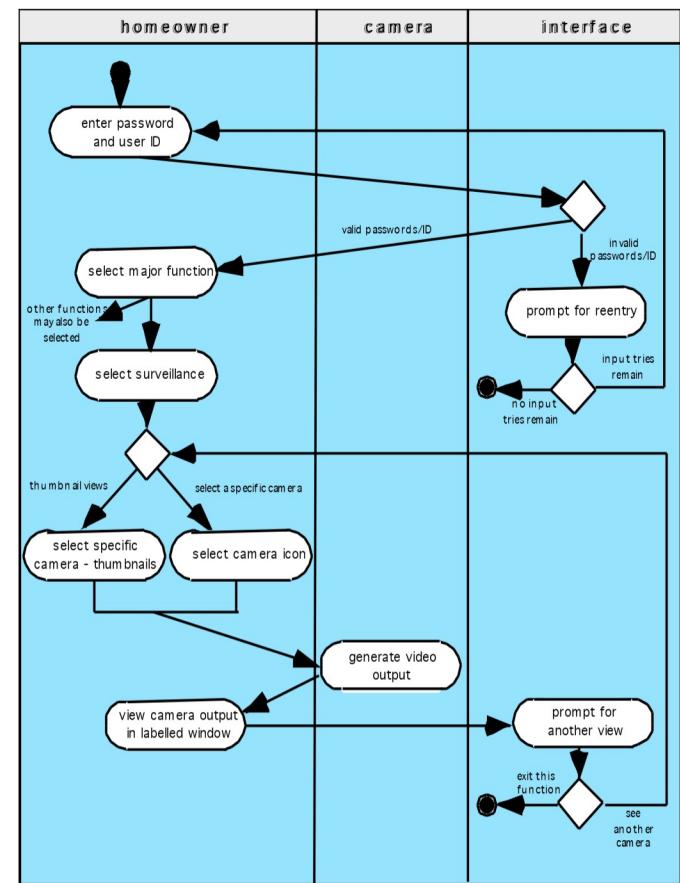
# SWIMLANE DIAGRAM



Allows the modeler to represent the flow of activities described by the use-case



Indicates which actor or analysis class has responsibility for the action described by an activity rectangle

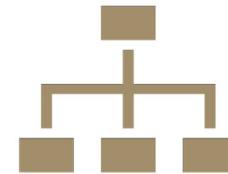


# CLASS-BASED MODELING



## Class-based modeling represents

- objects that the system will manipulate
- operations (also called methods or services)
- relationships between the objects
- collaborations that occur between the classes



## The elements of a class-based model include

- classes and objects
- attributes and operations
- collaboration diagrams and packages

# IDENTIFYING CLASSES



Classes are determined by underlining each noun or noun phrase and entering it into a simple table



If the class is required to implement a solution, then it is part of the solution space;



If a class is necessary only to describe a solution, it is part of the problem space.

## External entities

- (e.g. other systems, devices, people) that produce or consume information

## Things

- (e.g. reports, displays, letters, signals) that are part of the information domain for the problem

## Occurrences or events

- (e.g. completion of actions) that occur within the context of system operation

## Roles

- (e.g. manager, engineer, salesperson) played by people who interact with the system

## Organizational units

- (e.g. division, group, team) that are relevant to an application

## Places

- (e.g. manufacturing floor) that establish the context of the problem and the overall function

## Structures

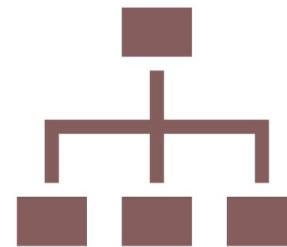
- (e.g. sensors, computers) that define a class of objects or related classes of objects

# MANIFESTATIONS OF CLASSES

# DEFINING ATTRIBUTES



Attributes describe a class that has been selected for inclusion in the analysis model.



Attributes describe the structure and value of an instance of a class.

# DEFINING OPERATIONS



An operation is a method or function that can be performed by a class.



Operations define the behavior of a class, what a class can do.



Operations can perform computation, take an action, call another method, etc.

# CLASS TYPES



## Entity classes

also called model or business classes, are extracted directly from the statement of the problem



## Boundary classes

are used to create the interface that the user sees and interacts with the software



## Controller classes

manage a “unit of work” from start to finish



System intelligence should be distributed across classes to best address the needs of the problem



Each responsibility should be stated as generally as possible



Information and the behavior related to it should reside within the same class



Information about one thing should be localized with a single class, not distributed across multiple classes.



Responsibilities should be shared among related classes, when appropriate.

## RESPONSIBILITIES

# COLLABORATIONS



Classes fulfill their responsibilities in one of two ways:

- a class can use its own operations to fulfill a particular responsibility
- a class can collaborate with other classes



Collaborations identify relationships between classes



Collaborations are identified by determining whether a class can fulfill each responsibility itself

# ASSOCIATION, AGGREGATION AND COMPOSITION



**Association is a (\*a\*) relationship between two classes, where one class use another**



**Aggregation, a special type of an association, is the (\*the\*) relationship between two classes.**

Association is non-directional, aggregation insists a direction.



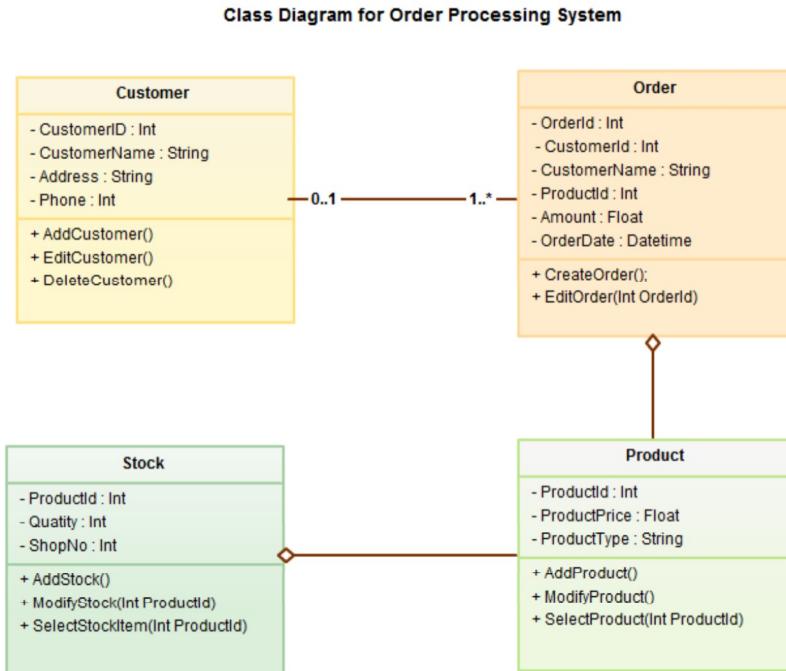
**Composition can be recognized as a special type of an aggregation.**

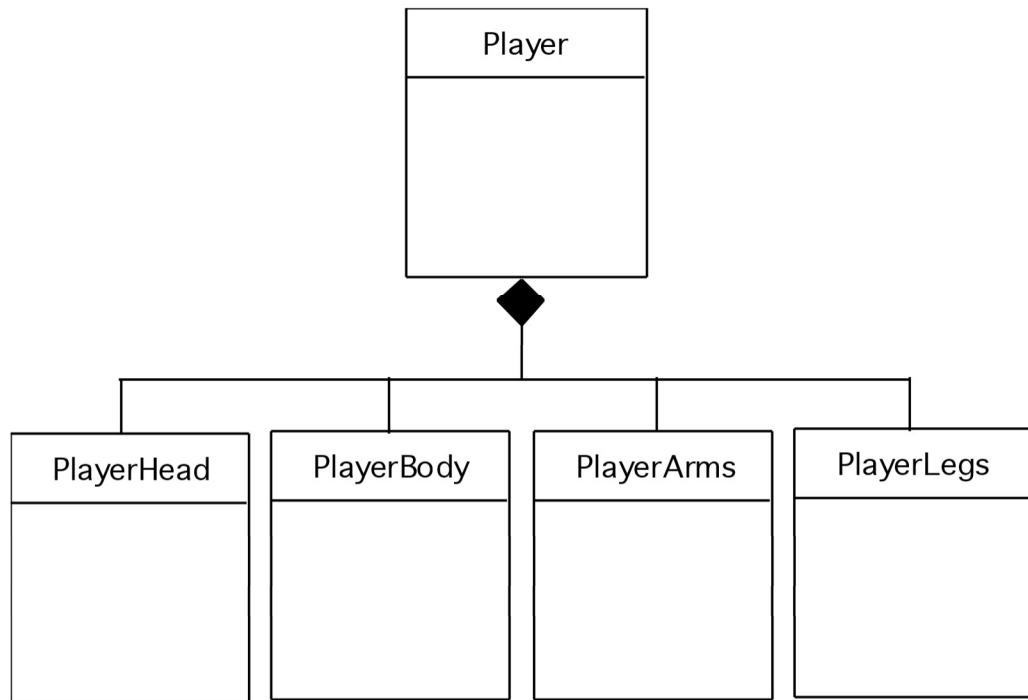


**Aggregation is a special kind of an association and composition is a special kind of an aggregation.**

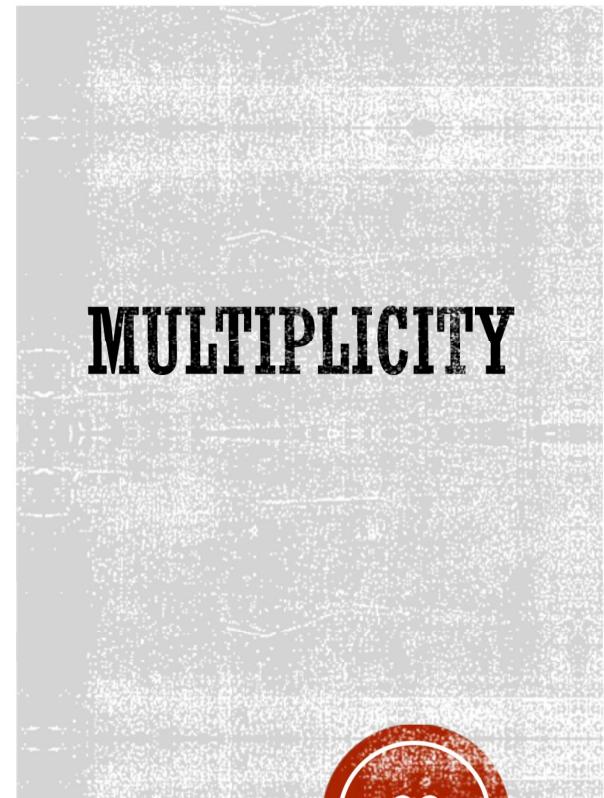
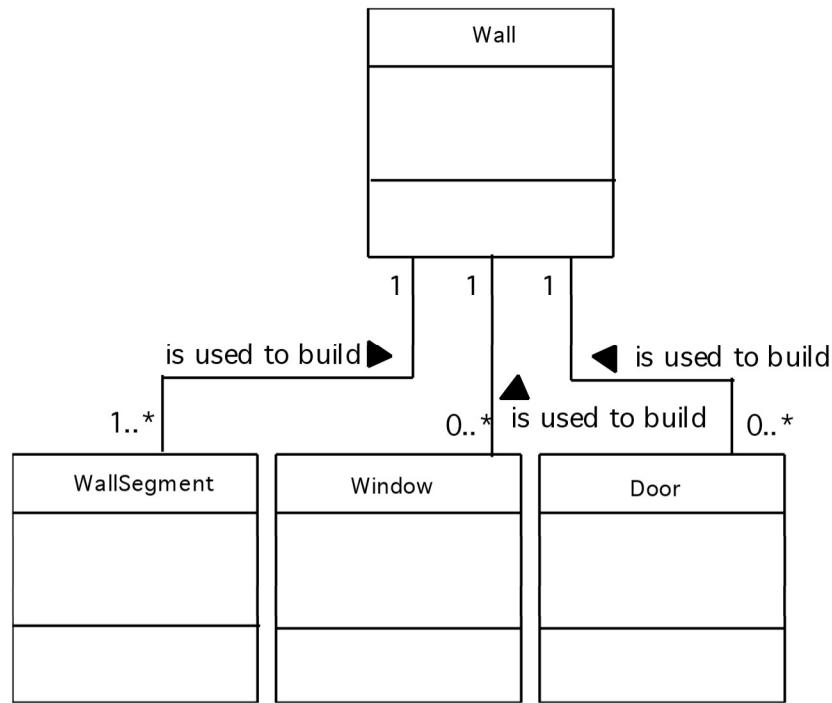
Association → Aggregation  
→ Composition

# EXAMPLE CLASS DIAGRAM



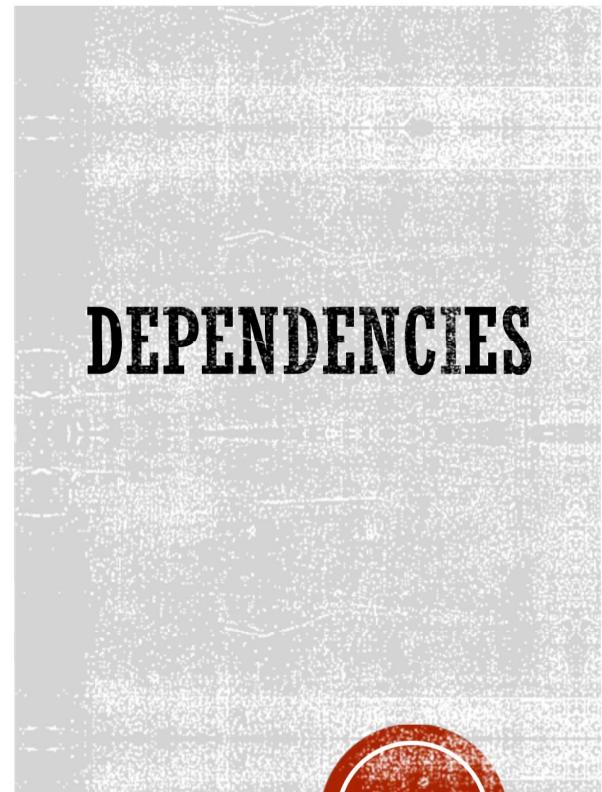
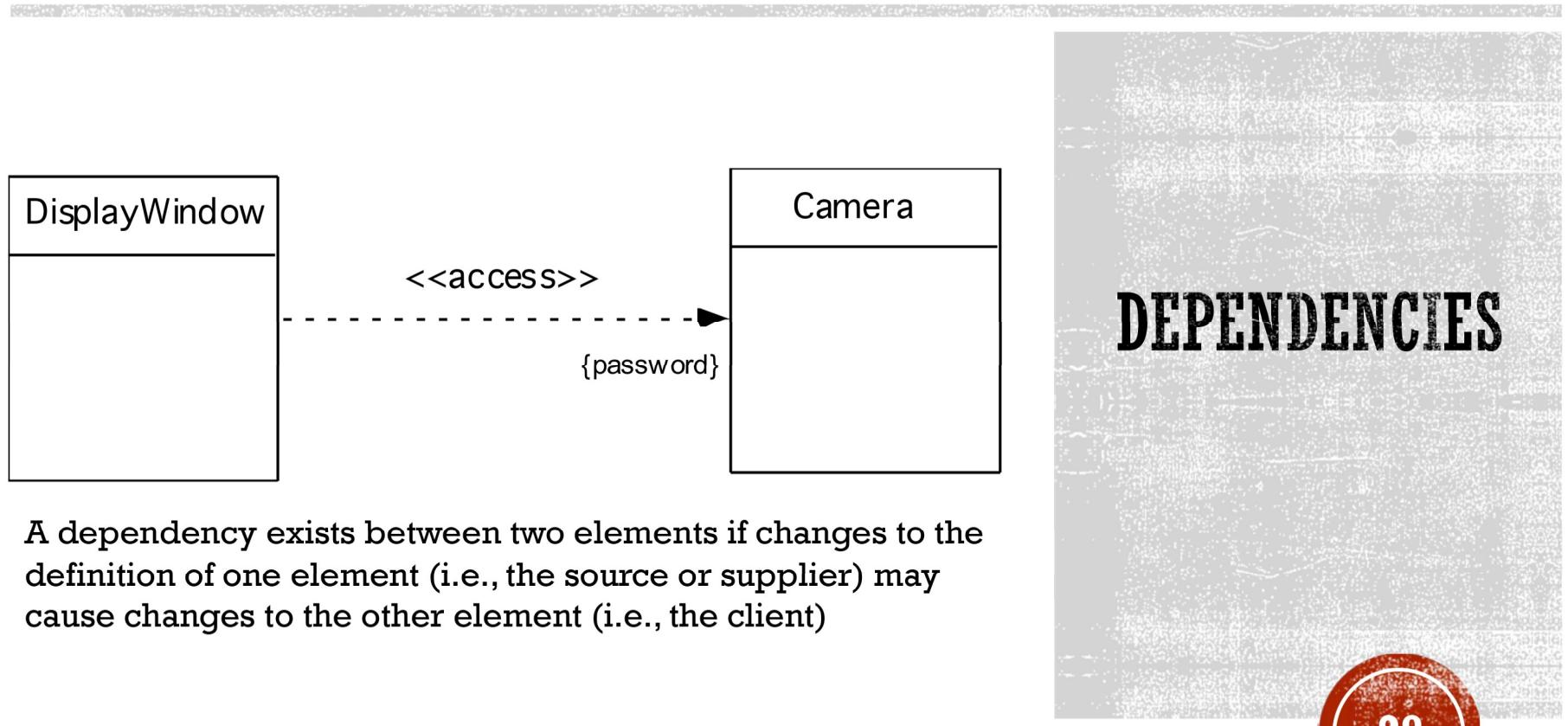


# COMPOSITE AGGREGATE CLASS



## MULTIPLICITY

28





The behavioral model indicates how software will respond to external events.



Evaluate all use-cases to fully understand the sequence of interaction within the system.



Identify events that drive the interaction sequence and understand how these events relate to specific objects.



Create a sequence for each use-case.



Build a state diagram for the system.



Review the behavioral model to verify accuracy and consistency.

# BEHAVIORAL MODELING

# STATE DIAGRAMS

In the context of behavioral modeling, two different characterizations of states must be considered:

- the state of each class as the system performs its function and
- the state of the system as observed from the outside as the system performs its function

The state of a class takes on both passive and active characteristics:

- A passive state is simply the current status of all of an object's attributes.
- The active state of an object indicates the current status of the object as it undergoes a continuing transformation or processing.

# THE STATES OF A SYSTEM



## State

a set of observable circum-stances that characterizes the behavior of a system at a given time



## State transition

the movement from one state to another



## Event

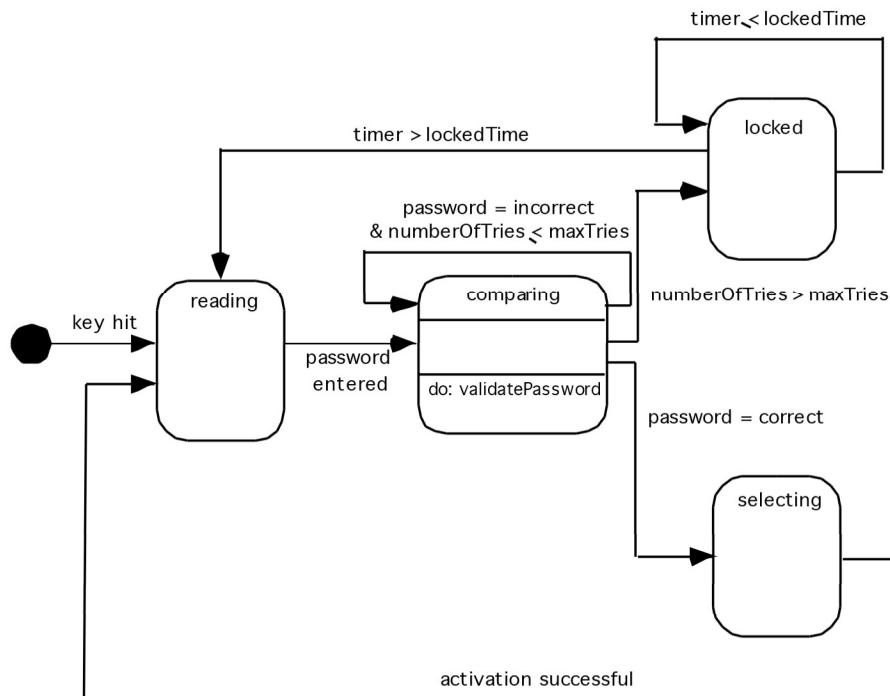
an occurrence that causes the system to exhibit some predictable form of behavior



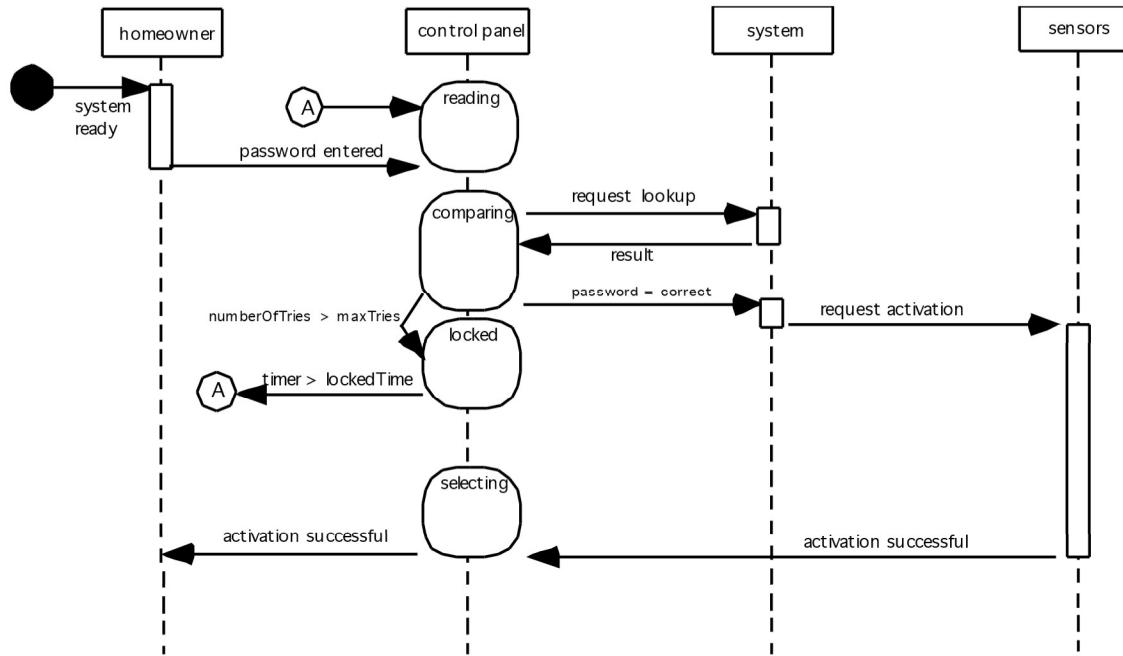
## Action

process that occurs as a consequence of making a transition

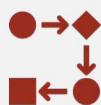
# STATE DIAGRAM EXAMPLE



# SEQUENCE DIAGRAM



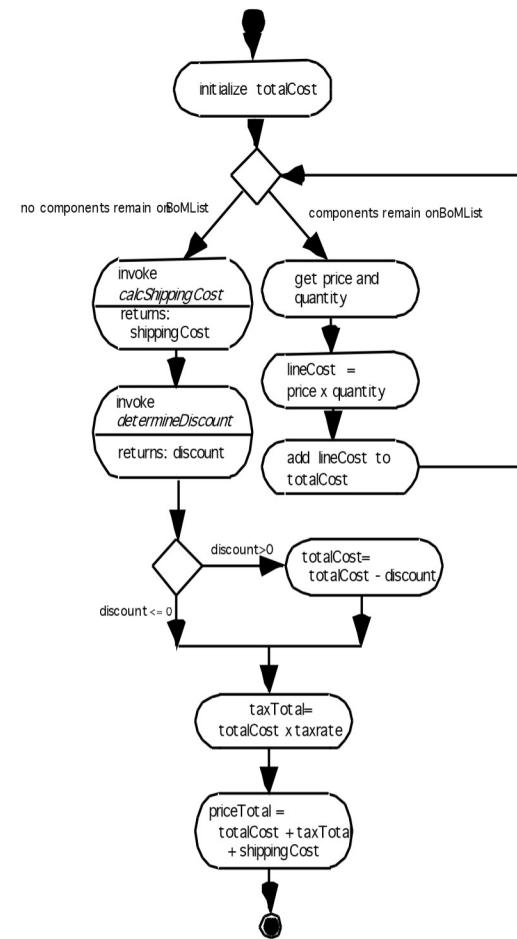
# ACTIVITY DIAGRAM



A flowchart to represent the flow from one activity to another activity.



The activity can be described as an operation of the system.



# DATA FLOW DIAGRAM



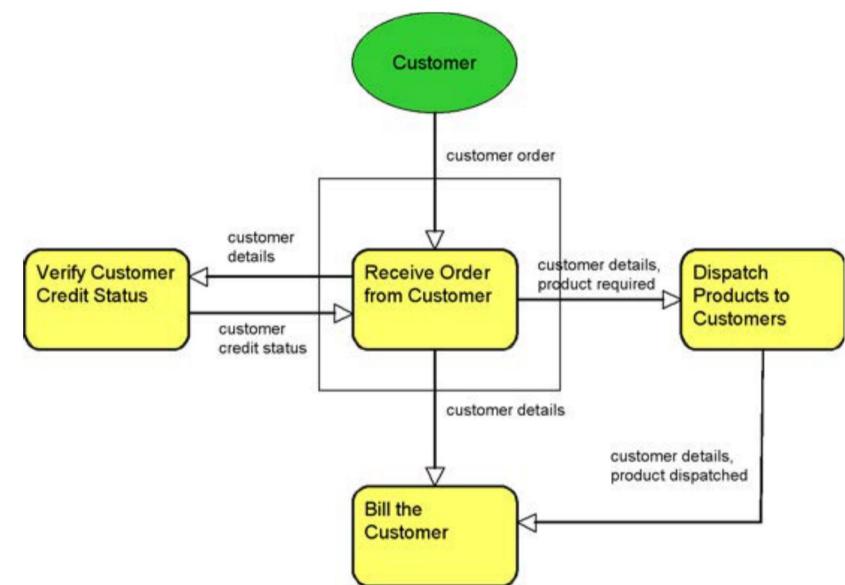
A data-flow diagram is a way of representing a flow of a data of a process or a system.



The DFD also provides information about the output and input of each entity and the process itself.



A data-flow diagram has no control flow, there are no decision rules and no loops.



# OBJECT DIAGRAM



Represents a snapshot of the objects in a system at a point in time



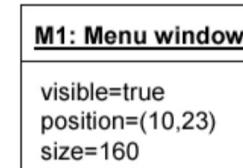
Shows instances rather than classes, therefore it is sometimes called an instance diagram



When to use object diagrams

To show examples of objects connected together based on a specific multiplicity number

To show instances with values for their attributes



ID label : Class Name

Attribute values

# PACKAGE DIAGRAM

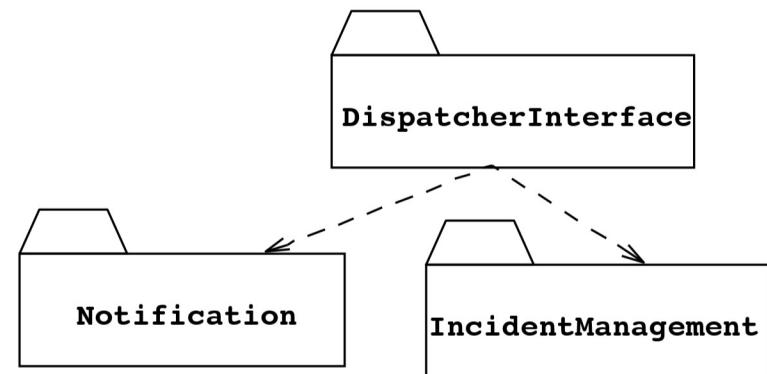
Used to take any construct in UML and group its elements together into higher-level units

Used most often to group classes

Corresponds to the package concept in Java

Represented by a tabbed folder, where the tab contains the package name

Can show dependencies between packages



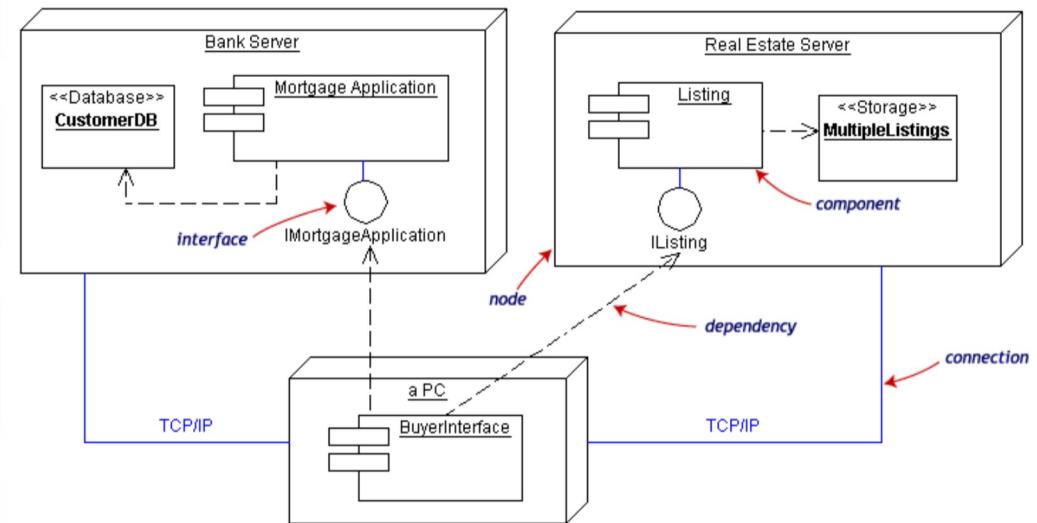
# DEPLOYMENT DIAGRAM

Shows the physical architecture of the hardware and software of the deployed system

Typically contain components or packages

Physical relationships among software and hardware in a delivered systems

Explains how a system interacts with the external environment





A use case diagram helps describe how people interact with the system



An activity diagram shows the context for use cases and also the details of how a complicated use case works



A class diagram drawn from the conceptual perspective is a good way of building up a rigorous vocabulary of the domain

It also shows the attributes and operations of interest in domain classes and the relationships among the classes



A state diagram shows the various states of a domain class and events that change that state

# FITTING UML INTO SOFTWARE REQUIREMENTS ANALYSIS



A class diagram drawn from the software perspective can show design classes, their attributes and operations, and their relationships with the domain classes



A sequence diagram helps to combine use cases in order to see what happens in the software



A package diagram shows the large-scale organization of the software



A state diagram shows the various states of a design object and events that change that state



A deployment diagram shows the physical layout of the software

# FITTING UML INTO SOFTWARE DESIGN



Complements the written documentation and in some instances can replace it



Captures the outcome of the requirements analysis and design activities in a graphical format



Supplies a software maintainer with an overall understanding of a system



Provides a good logical roadmap of the system layout



Describes the various states in which a system may exist



Details complex algorithms in a more understandable form



Shows how multiple objects collaborate in the system

## FITTING UML INTO SOFTWARE DOCUMENTATION

# HOMEWORK

---



Review class notes.



Additional reading:  
Examples of UML diagrams



Start a discussion on Google Groups to clarify your doubts.