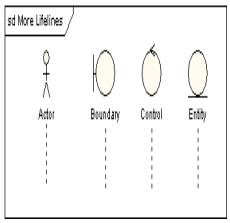
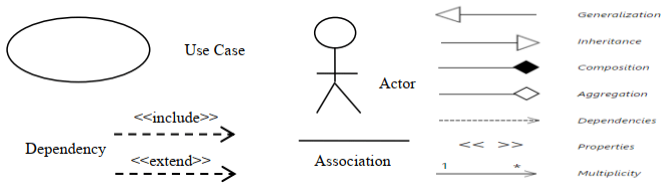
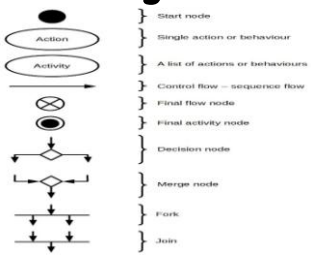
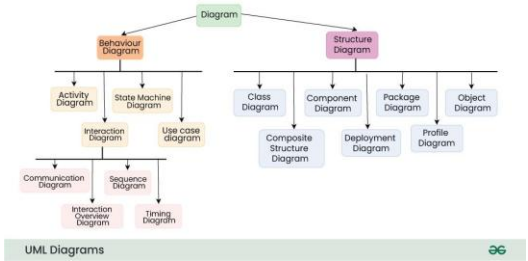



# Software Engineering And Testing

# Object Modeling using UML



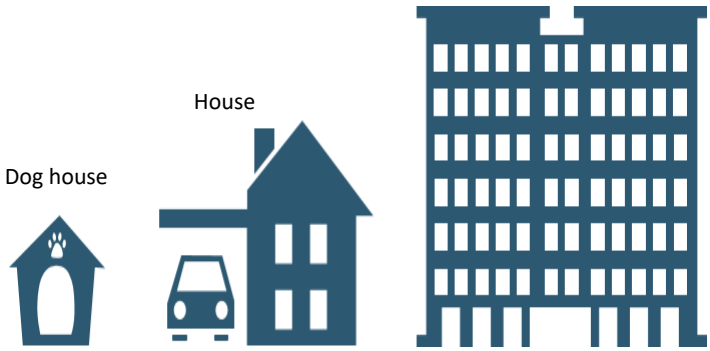




## Outline

- The Importance of Modeling.
- Object-Oriented Modeling and Principles
- An Overview and Conceptual Model of UML
- Object Modelling Using UML - Basic Object-Orientation Concepts
- Unified Modelling Language (UML)
- UML Diagrams, Use Case Model, Class Diagrams, Interaction Diagrams, Activity Diagram, State Chart Diagram, Package, Component, and Deployment Diagrams
- Object-Oriented Software Development - Patterns, Some Common Design Patterns
- Introduction to Object-Oriented Analysis and Design (OOAD) Methodologies
- Applications of the Analysis and Design Process

## The Importance of Modeling

- A key goal of the Object-Oriented approach is **to decrease the "semantic gap" between the system and the real world by using terminology that is the same as the functions that users perform**. Modeling is an essential tool to facilitate achieving this goal .




## The Importance of Modeling

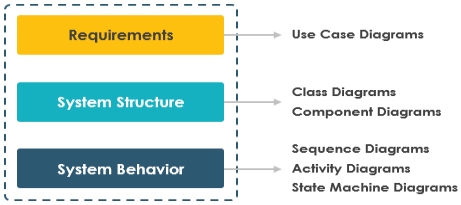
- Modeling is a central part of all the activities that lead up to the deployment of good software.
  - We build models to communicate the desired structure and behavior of our system.
  - We build models to visualize and control the system's architecture.
  - We build models to better understand the system we are building, often exposing opportunities for simplification and reuse.
  - And we build models to manage risk.
- Modeling is a proven & well accepted engineering techniques. In building architecture, we develop architectural models of houses & high rises to help visualize the final products.

## The Importance of Modeling

- A model is a simplification of reality, providing blueprints of a system. UML, in specific:
  - Permits you to specify the structure or behavior of a system.
  - Helps you visualize a system.
  - Provides template that guides you in constructing a system.
  - Helps to understand complex system part by part.
  - Document the decisions that you have made.

## Principles of UML Modeling

- **The choice of model is important**
  - The choice of what models to create has a profound influence on how a problem is attacked and how a solution is shaped. We need to choose your models well.
  - The right models will highlight the most critical development problems.
  - Wrong models will mislead you, causing you to focus on irrelevant issues.
  - For Example: We can use different types of diagrams for different phases in software development.



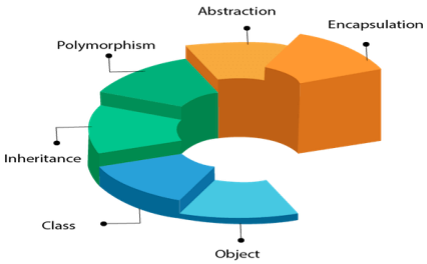
## Principles of UML Modeling

- **Every model may be expressed at different levels of precision**
  - For Example, If you are building a high rise, sometimes you need a 30,000-foot view for instance, to help your investors visualize its look and feel.
  - Other times, you need to get down to the level of the studs for instance, when there's a tricky pipe run or an unusual structural element.
- **The best models are connected to reality**
  - All models simplify reality and a good model reflects important key characteristics.
- **No single model is sufficient**
  - Every non-trivial system is best approached through a small set of nearly independent models. Create models that can be built and studied separately, but are still interrelated.

## Principles of UML Modeling

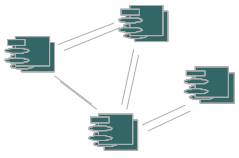
- **No single model is sufficient**
  - In the case of a building:
    - You can study electrical plans in isolation
    - But you can also see their mapping to the floor plan and perhaps even their interaction with the routing of pipes in the plumbing plan.

OOPs (Object-Oriented Programming System)

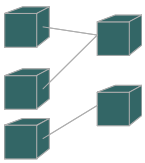


## Principle 1: The Choice of Model Is Important

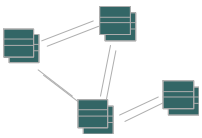
- The models you create profoundly influence how a problem is attacked and how a solution is shaped.
  - In software, the models you choose greatly affect your world view.
  - Each world view leads to a different kind of system.
  - For example, If you build a system through the eyes of a database developer (entity-relationship models) or
  - a structured analyst (algorithmic-centric models) or
  - an object-oriented developer (classes and their collaboration centric models)



Process Model



Deployment Diagram

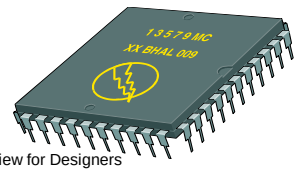


Design Model

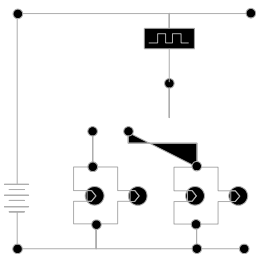
9

## Principle 2: Levels of Precision May Differ

- Every model may be expressed at different levels of precision.
  - The best kinds of models let you choose your degree of detail, depending on:
    - who is viewing the model.
    - why they need to view it.




View for Designers



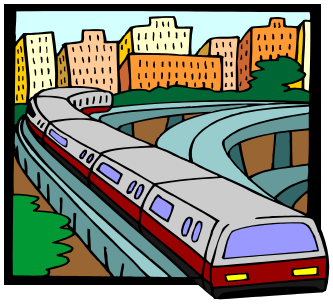
View for Customers

  - An analyst or an end user will want to focus on issues of what; a developer will want to focus on issues of how.
  - Both of these stakeholders will want to visualize a system at different levels of detail at different times.

10

## Principle 3: The Best Models Are Connected to Reality

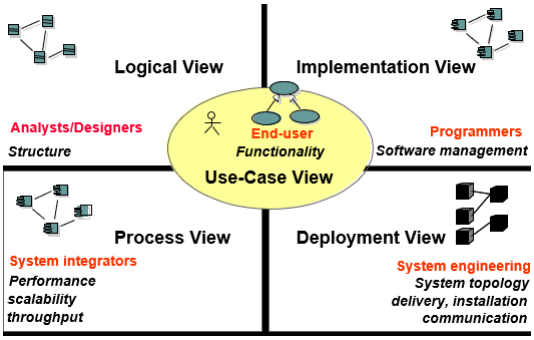
- All models simplify reality.
- A good model reflects potentially fatal characteristics.
- All models simplify reality; the trick is to be sure that your simplifications don't mask any important details.



11

## Principle 4: No Single Model Is Sufficient

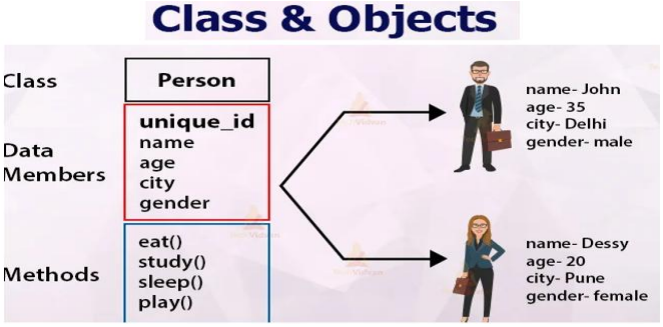
- No single model is sufficient. Every non-trivial system is best approached through a small set of nearly independent models.
  - Create models that can be built and studied separately, but are still interrelated.



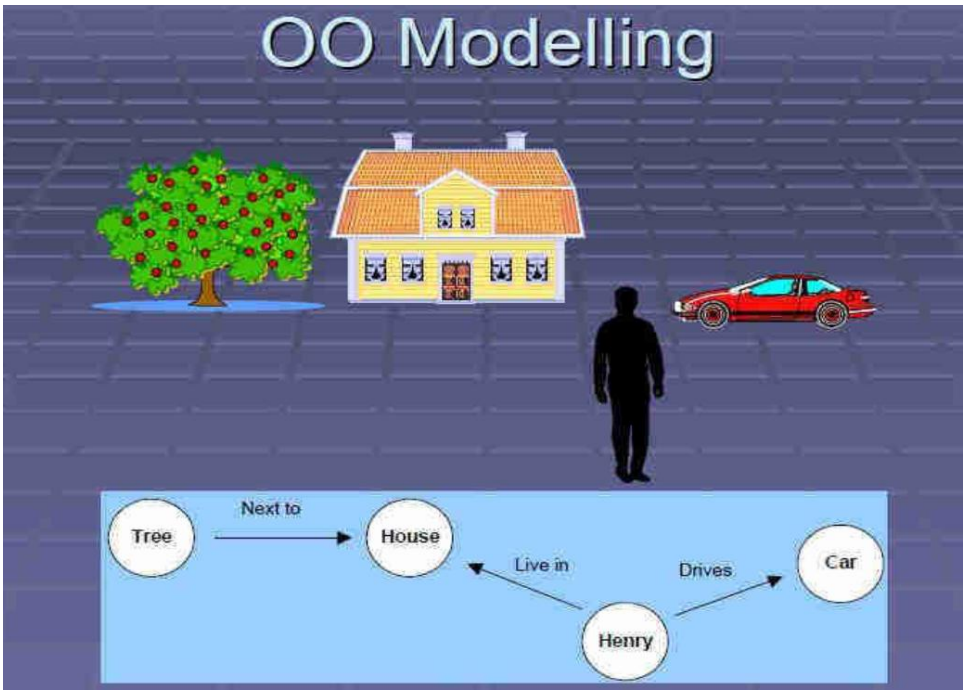

12


## Object Oriented Modeling

- An object is a thing, generally drawn from the vocabulary of the problem space or the solution space; a class is a description of a set of common objects.
- Every object has identity (you can name it or otherwise distinguish it from other objects), state (there's generally some data associated with it), and behaviour (you can do things to the object, and it can do things to other objects, as well).

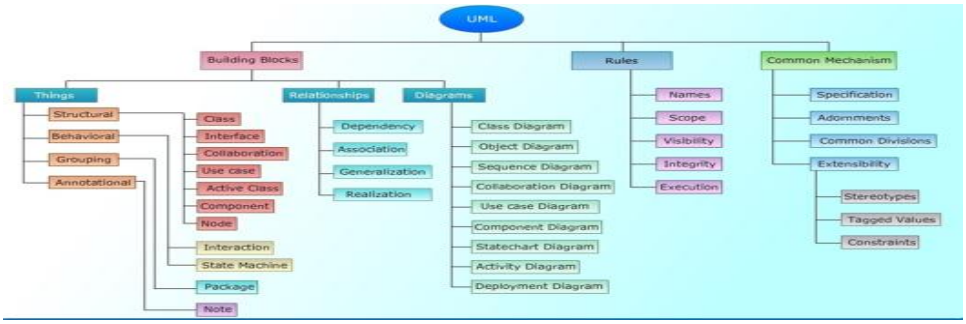



14




## Conceptual Model of UML

- To understand the UML, you need to form a conceptual model of the language, and this requires learning three major elements:
  - the UML's **basic building blocks**,
  - the **rules** that dictate how those building blocks may be put together, and
  - some **common mechanisms** that apply throughout the UML.



## The basic building blocks of the UML

- The vocabulary of the UML encompasses three kinds of building blocks:
  1. Things - important modeling concepts
  2. Relationships - tying individual things
  3. Diagrams - grouping interrelated collections of things and relationships
- Things are the abstractions that are first-class citizens in a model;
- relationships tie these things together;
- diagrams group interesting collections of things.

17

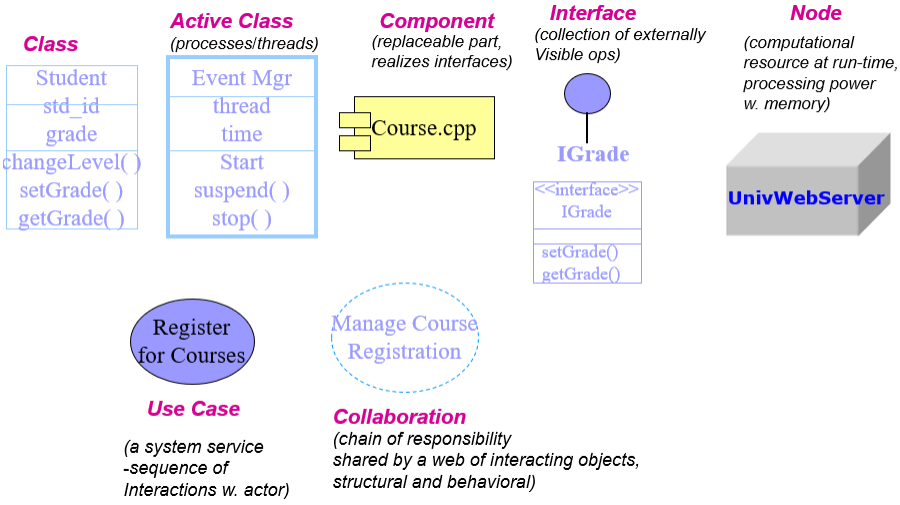
## The basic building blocks of the UML - Things

- UML 1.x
  - Structural — nouns/static of UML models (irrespective of time).
  - Behavioral — verbs/dynamic parts of UML models.
  - Grouping — organizational parts of UML models.
  - Annotational — explanatory parts of UML models.

18

## Structural Things in UML - *7 Kinds (Classifiers)*

- Nouns.
- Conceptual or physical elements.




## Behavioral Things in UML

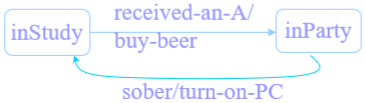
- Verbs.
- Dynamic parts of UML models: "behavior over time"
- Usually connected to structural things.
- Two primary kinds of behavioral things:

❑ ***Interaction***

a set of objects exchanging messages, to accomplish a specific purpose.


❑ ***State Machine***

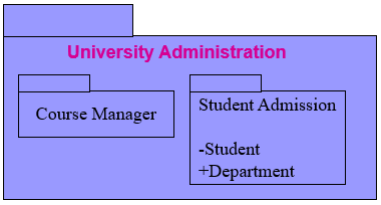
specifies the sequence of states an object or an interaction goes through during its lifetime in response to events.




20

## Grouping Things in UML – *Packages*

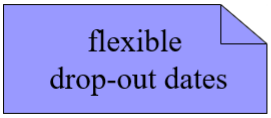
- For organizing elements (structural/behavioral) into groups.
- Purely conceptual; only exists at development time.
- Can be nested.
- Variations of packages are: Frameworks, models, & subsystems.

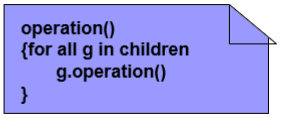




21

## Annotational Things in UML – *Notes*

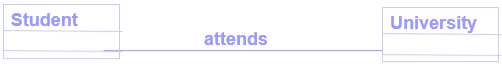
- Explanatory/Comment parts of UML models - usually called adornments
- Expressed in informal or formal text.




22

# Basic building blocks of UML – *Relationship*




***1. Associations***

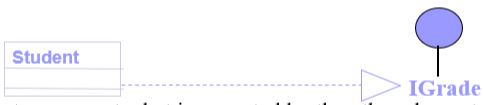
*Structural* relationship that describes a set of links, a link being a connection between objects.

(UML2.0: The semantic relationship between two or more classifiers that involves connections among their instances.)

***variants: aggregation & composition***


***2. Generalization***

a specialized element (the child) is more specific the generalized element.




***3. Realization***

one element guarantees to carry out what is expected by the other element.

*(e.g, interfaces and classes/components; use cases and collaborations)*


***4. Dependency***

a change to one thing (independent) may affect the semantics of the other thing (dependent).

*(direction, label are optional)*

# Basic building blocks of UML – *Diagrams*

A connected graph: Vertices are things; Arcs are relationships/behaviors.

**UML 1.x: 9 diagram types.**

**Structural Diagrams**

*Represent the static aspects of a system.*

- Class; Object
- Component
- Deployment

**Behavioral Diagrams**

*Represent the dynamic aspects.*

- Use case
- Sequence; Collaboration
- Statechart
- Activity

**UML 2.0: 12 diagram types**

**Structural Diagrams**

- Class; Object
- Component
- Deployment
- Composite Structure
- *Package*

**Behavioral Diagrams**

- Use case
- Statechart
- Activity

**Interaction Diagrams**

- Sequence; *Communication*
- Interaction Overview
- Timing

## Types of UML Diagrams

- **Class diagram** – A class diagram is a static diagram that describes the structure of a system by showing its classes and their properties and operations, as well as the relationships between objects.
- **Use Case Diagram** – A use case diagram consists of use cases, roles, and the relationships between them. It shows how users interact with the system and defines the specifications of the use cases.
- **Sequence diagram** – A sequence diagram is a model for communication between objects in a sequential manner. It shows the exact order of objects, classes and roles and information involved in a scenario. It consists of vertical lines belonging to lifelines and horizontal lines of messages.

25

## Types of UML Diagrams

- **Activity diagram** – An activity diagram is a behavior diagram that shows a scenario in terms of the flow of actions. It models a sequence of actions, condition-based decisions, concurrent branches and various loops.
- **Communication diagram** – A communication diagram shows the interaction between objects and parts in the form of messages, which are represented by lifelines. A communication diagram is a modified form of a UML sequential diagram, but differs from it in that its elements do not need to be horizontally ordered and can have any position in the diagram.
- **State Machine Diagram** – A state machine diagram describes the state of an entity (device, process, program, software, module, etc.) and the transitions between states. The conditions specify when a transition from one state to another can be used.

26

## Types of UML Diagrams

- **Object diagram** – An object diagram is a structured UML diagram. It describes a system or its parts at a particular time. It models instances, their values and relationships. It can be used to show examples of data structures.
- **Package diagram** – A package diagram shows the dependencies between packages in a model. It describes the structure and organization of large-scale projects.
- **Component diagram** – A component diagram provides a view of a complex system. It describes the interfaces provided and/or required by the various parts of the system and the relationships between the parts. These parts are represented by components and other artifacts.

27

## Types of UML Diagrams

- **Deployment Diagram** – The deployment diagram describes the deployment of artifacts on a network node. It is used to show the location of artifacts (software, systems, modules, etc.) on physical nodes (hardware, servers, databases, etc.) and the relationships between specific parts of the solution.
- **Composite Structure Diagram** – The composite structure diagram shows the internal structure of a classifier, its parts and ports, through which it communicates with its environment. It models collaboration, where each element has its defined role.
- **Interaction Overview Diagram** – The Interaction Overview Diagram provides a high level view of the interactions in a system or subsystem. It describes processes in a similar way to activity diagrams, but it uses other interaction diagrams and interaction references rather than action nodes.

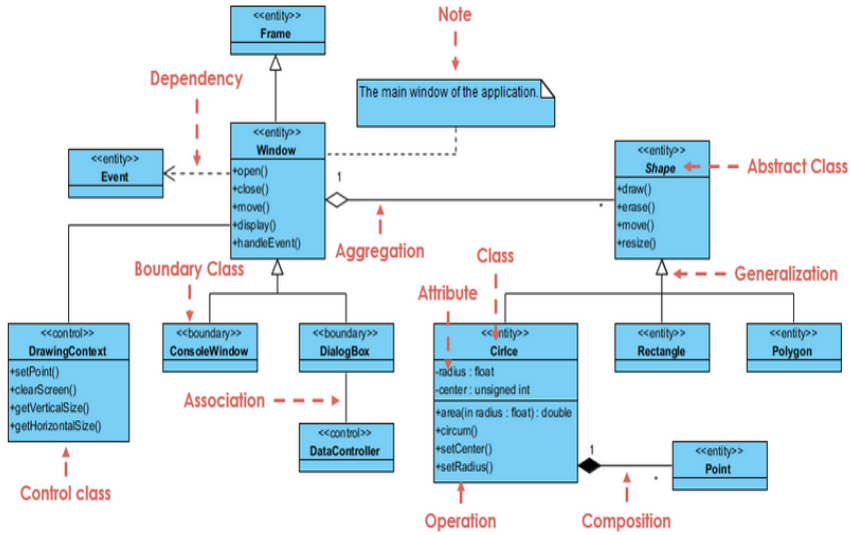
28

## Types of UML Diagrams

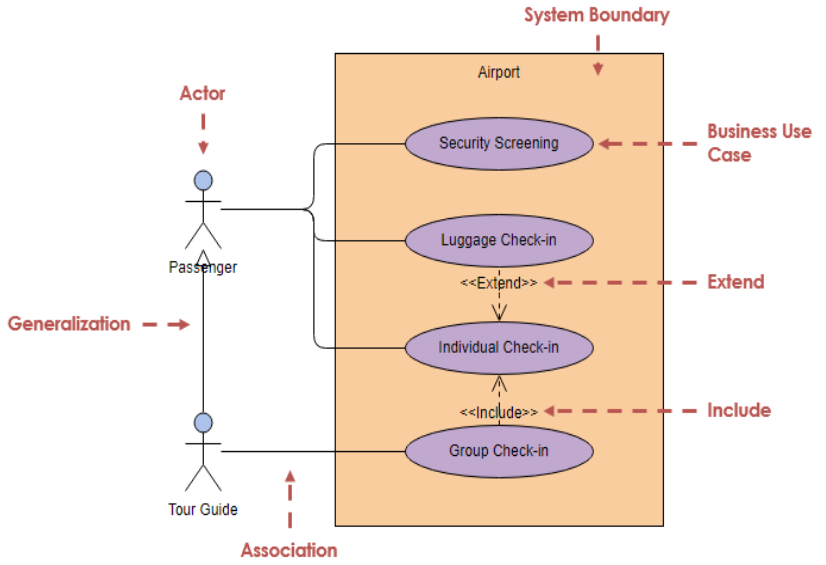
- **Timing Diagrams** – The timing diagram focuses primarily on time, and it describes the changes in the classifier on a timeline. The timelines are stacked vertically, with time increasing from left to right.
- **Profile Diagram** – The Profile Diagram describes and defines extensions to the UML language. The extension mechanism allows you to adapt the language to a specific domain or platform. Extensions are defined by stereotyping.

29

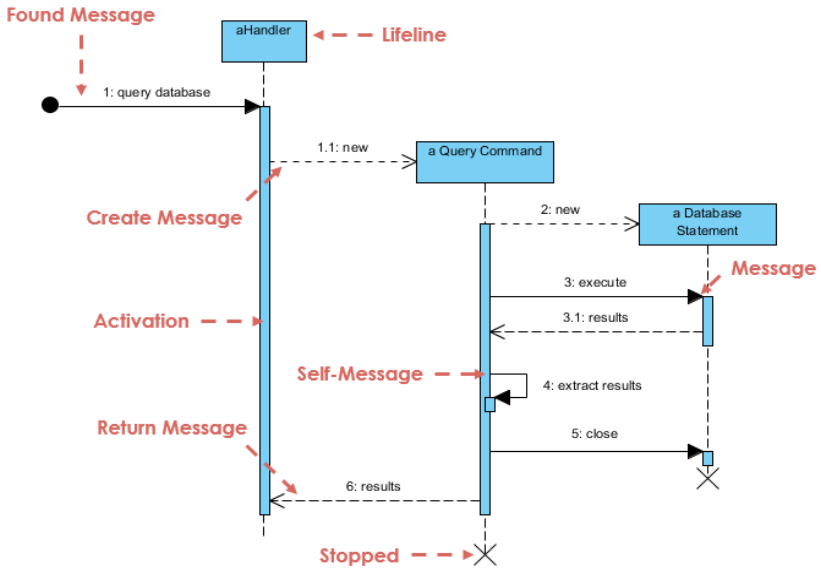
## Class Diagram



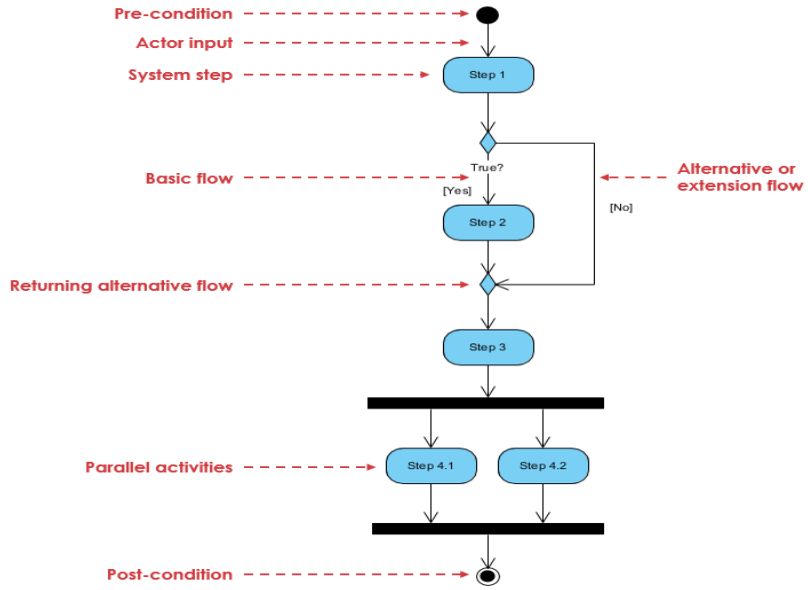
# Use Case Diagram



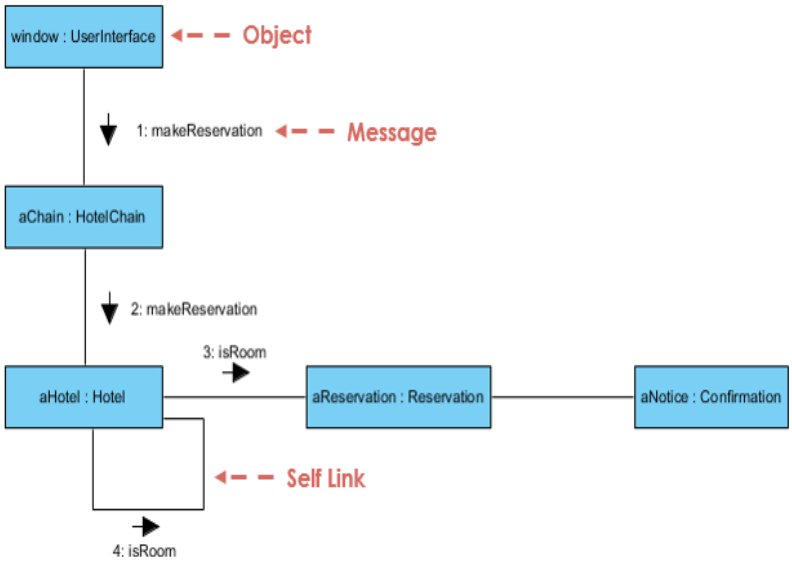

# Sequence Diagram



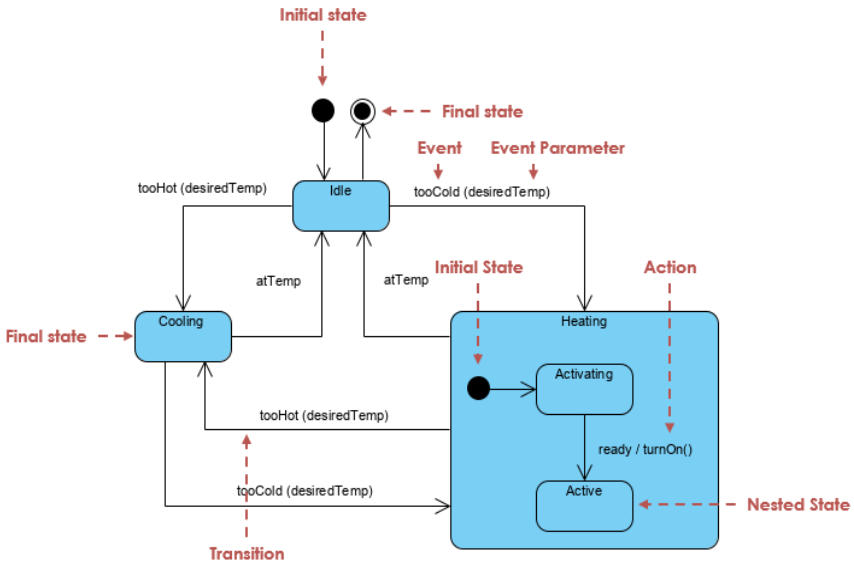
# Activity Diagram



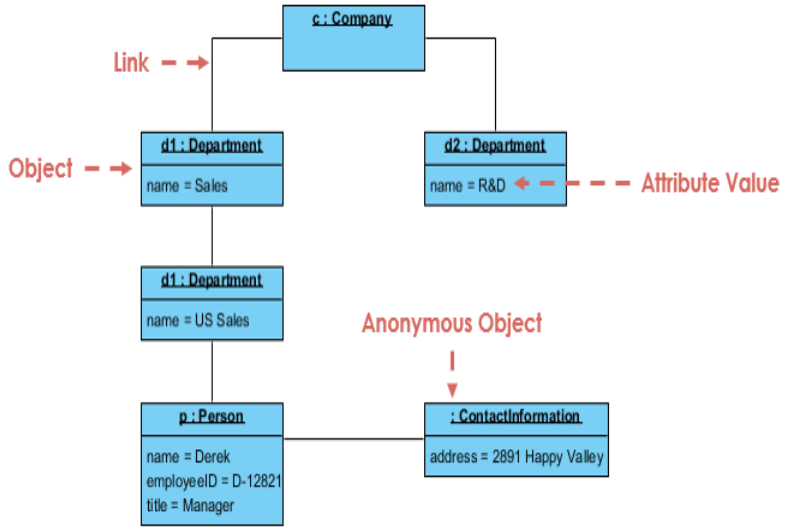

# Communication Diagram



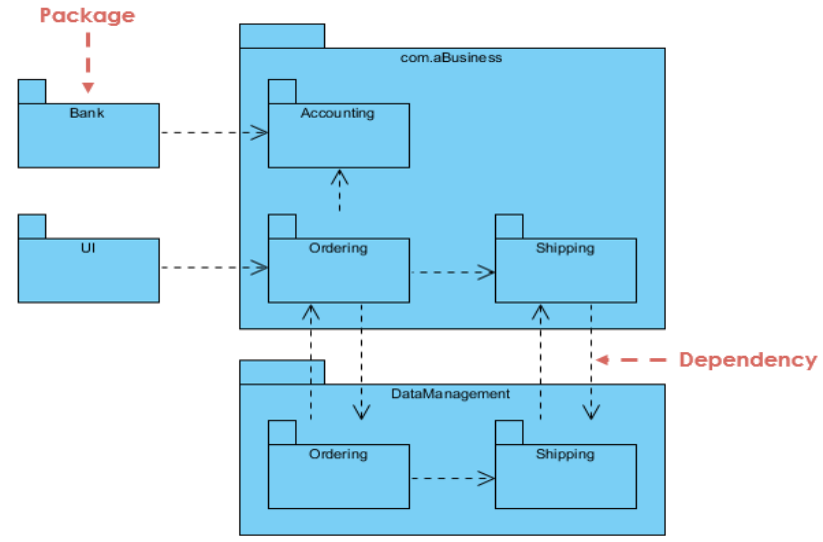
# State Machine Diagram




# Object Diagram

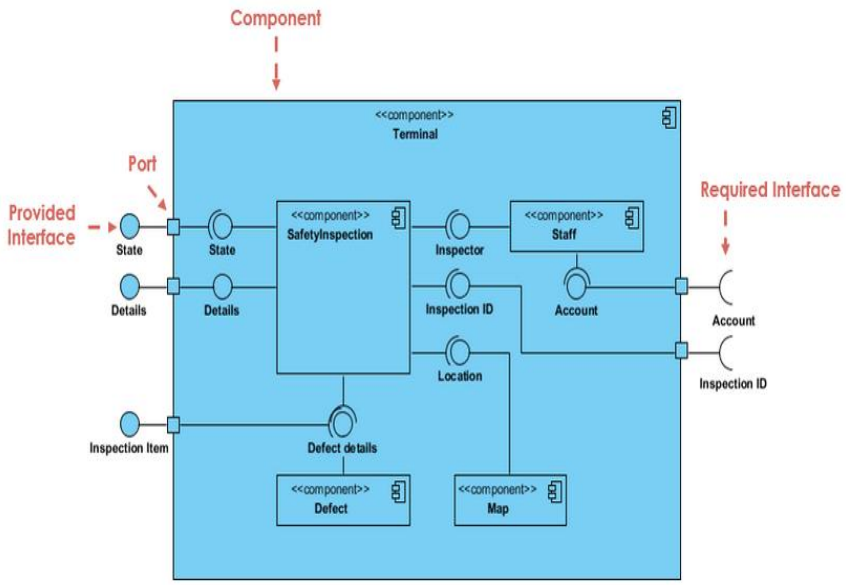


# Package Diagram

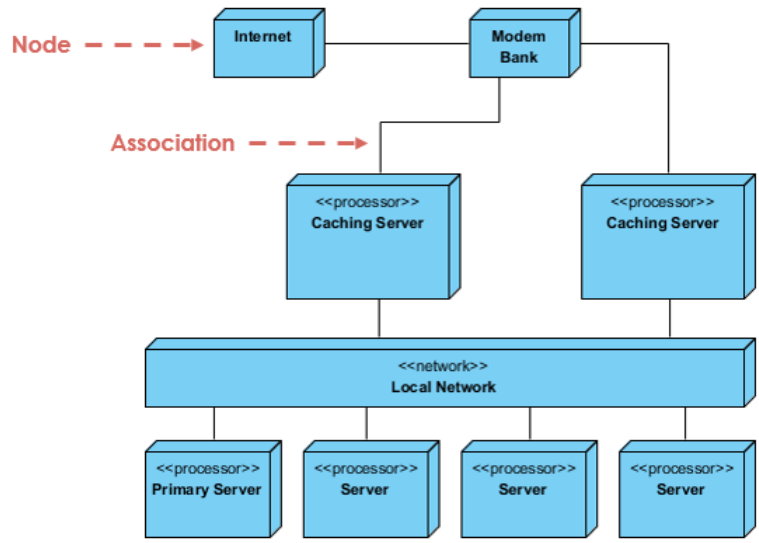



37

# Component Diagram

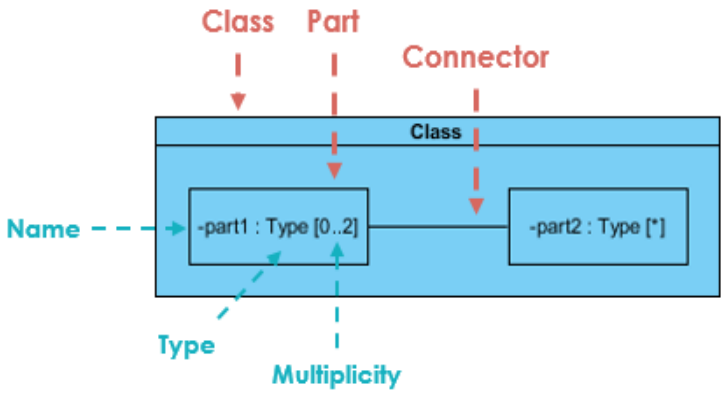


# Deployment Diagram



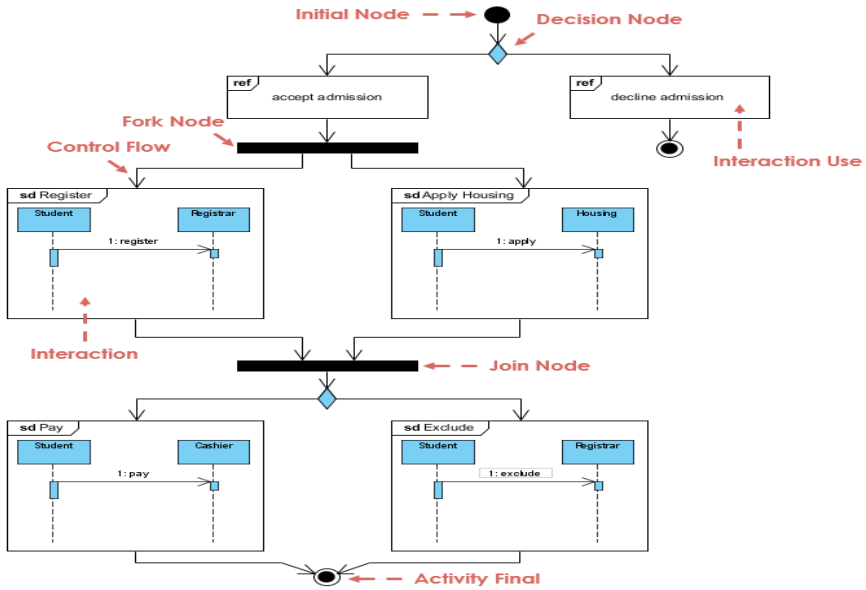

39

# Composite Structure Diagram

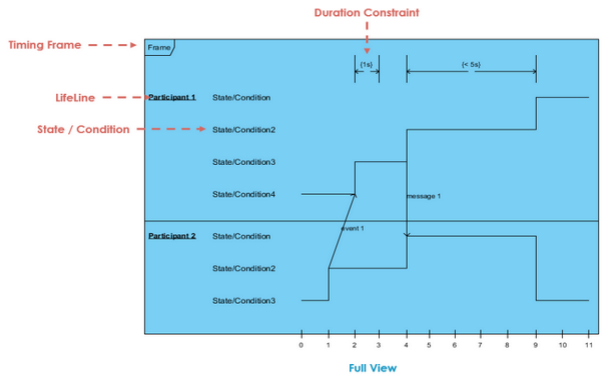


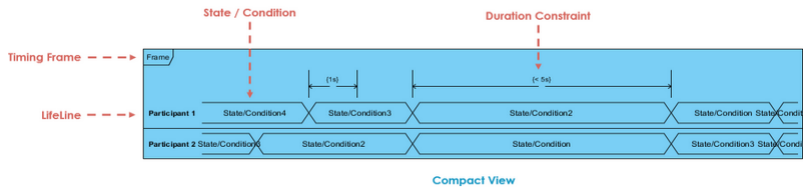

40

# Interaction Overview Diagram

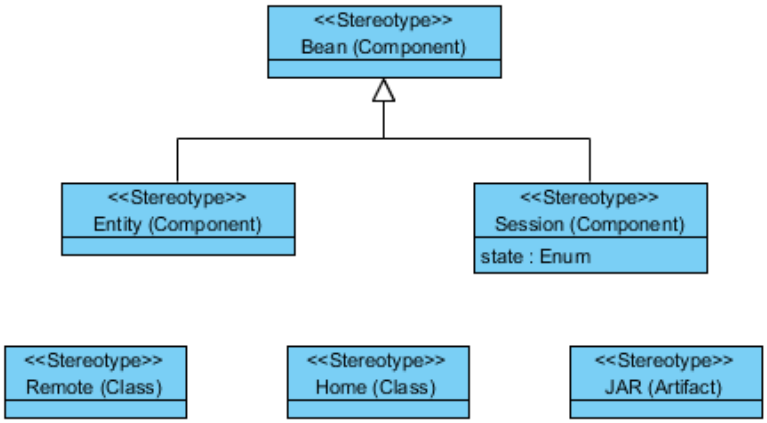



# Timing Diagram



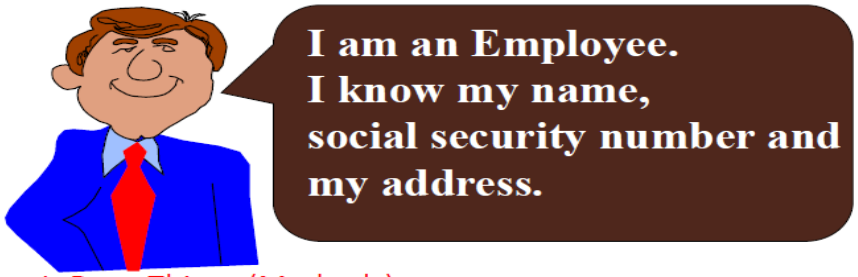
## Profile Diagram



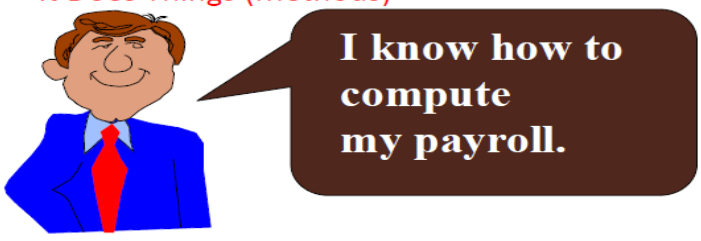

43

## What is an Object?

- It Knows Things (Attributes)



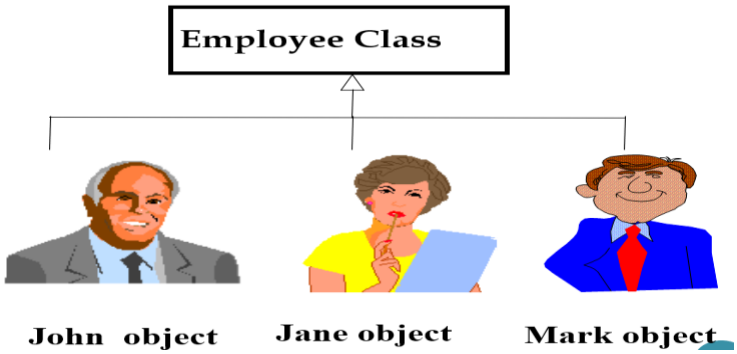

- It Does Things (Methods)




44

## Objects Are Grouped In Classes

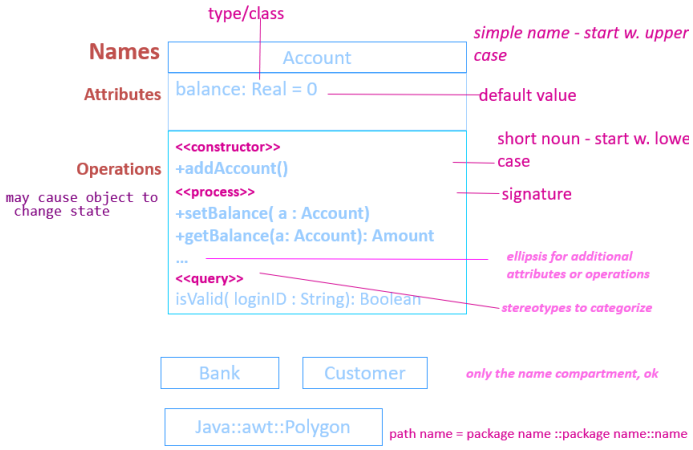
- The role of a class is to define the attributes and methods (the state and behavior) of its instances
- We distinguish classes from instances.(e.g. eagle or airplane)
- The class car, for example, defines the property color
- Each individual car (object) will have a value for this property, such as "maroon," "yellow" or "white"




45

## Classes

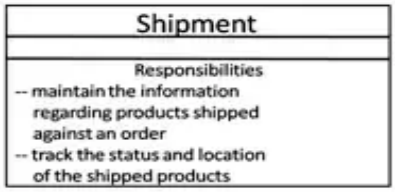
- A class is a description of a set of objects that share the same attributes, operations, relationships, and semantics.
- Graphically, a class is rendered as a rectangle.




46

## Classes

- **Responsibilities**:
  - The responsibilities of a class specify the contract between the class and its objects. The responsibilities of a class represent its features (attributes and operations).
  - For example, the responsibility of a wall class is to represent the height, width and thickness of the wall. Responsibilities are represented as text in a compartment at the bottom of the class.

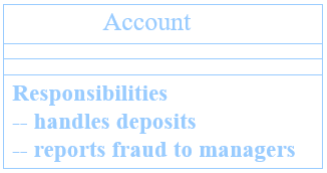



  - When you model a class, specify the things responsibilities.
  - You may use techniques like CRC cards and use-case based analysis.

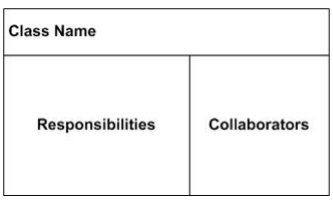
47

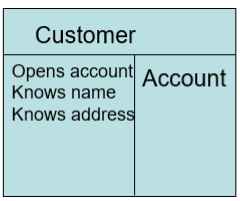
## Classes

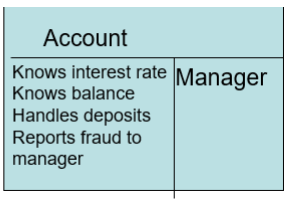
- **Responsibilities**:




  - You may use techniques like CRC cards and use-case based analysis.




48

# CRC

**Blank CRC Card**

| Class Name: | Superclass: | Subclasses: |
|---|---|---|
| Responsibilities | Collaborations | |
| | | |
| | | |
| | | |
| | | |
| | | |

**Address CRC Card**

| Class Name: Address | Superclass: | Subclasses: |
|---|---|---|
| Responsibilities | Collaborations | |
| Create itself (name, city, state, zip code) | None | |
| Know its name | None | |
| Know its city | None | |
| Know its state | None | |
| Know its zip code | None | |

18

**Name CRC Card**

| Class Name: Name | Superclass: | Subclasses: |
|---|---|---|
| Responsibilities | Collaborations | |
| Create itself (First, Middle, Last) | None | |
| Know its first name | None | |
| Know its middle name | None | |
| Know its last name | None | |
| Are two names equal? | String | |
| Compare two names | String | |

19

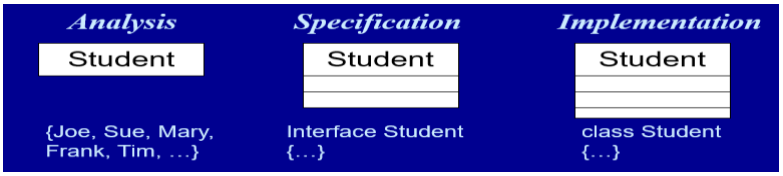
Return Type

| Class Name: Name | Superclass: | Subclasses: |
|---|---|---|
| Responsibilities | Collaborations | |
| Create itself (First, Middle, Last) | None | |
| Know its first name return String | None | |
| Know its middle name return String | None | |
| Know its last name return String | None | |
| Are two names equal? return boolean | String | |
| Compare two names | String | |

49

# Classes

- Three common perspectives:
  - **Analysis** - description of the problem domain
  - **Specification** - logical description of software system
  - **Implementation** - description of software components and their deployment
- Meaning from three perspectives
  - Analysis: sets of objects
  - Specifications: interfaces to encapsulated software representations of objects
  - Implementations: abstract data types

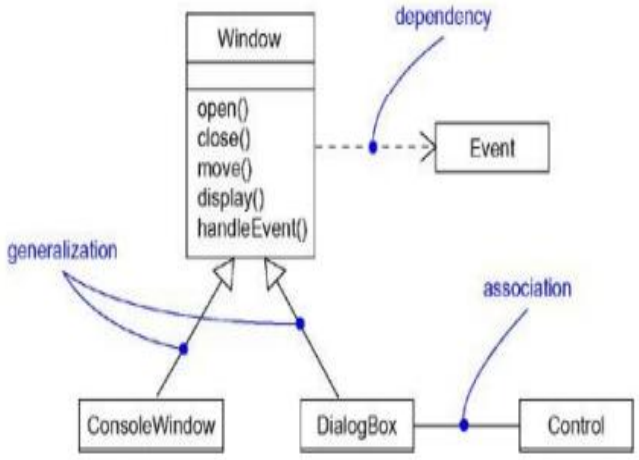



50

## Relationships

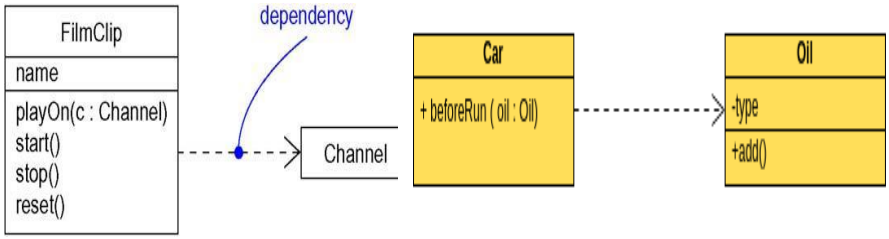
- When you build abstractions, you'll discover that very few of your classes stand alone.
- Instead, most of them collaborate with others in a number of ways.
- In object-oriented modeling, there are three kinds of relationships that are especially important:
  - Dependencies, which represent using relationships among classes (including refinement, trace, and bind relationships);
  - Generalizations, which link generalized classes to their specializations; and
  - Associations, which represent structural relationships among objects. Each of these relationships provides a different way of combining your abstractions.
- Other kinds of relationships are realization and refinement.

51



## Relationships - Dependency

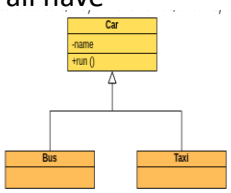
- Dependency : A dependency is a using relationship that states that a change in specification of one thing (for example, class Event) may affect another thing that uses it (for example, class Window), but not necessarily the reverse.
- The dependency relationship is “use” relationship
- Graphically, a dependency is rendered as a dashed directed line, directed to the thing being depended on. Use dependencies when you want to show one thing using another.




53

## Relationships - Generalization

- Generalization/Inheritance : A generalization is a relationship between a general thing (called the superclass or parent)and a more specific kind of that thing (called the subclass or child).
- Generalization is sometimes called an "is-a-kind-of" relationship: one thing (like the class BayWindow) is-a-kind-of a more general thing (for example, the class Window).
- Generalization means that objects of the child may be used anywhere the parent may appear, but not the reverse. In other words, generalization means that the child is substitutable for the parent.
- For example: buses, taxis, and cars are cars, they all have names, and they can all be on the road.



## Relationships- Generalization

- A child inherits the properties of its parents, especially their attributes and operations.
- Often-but not always-the child has attributes and operations in addition to those found in its parents.
- An operation of a child that has the same signature as an operation in a parent overrides the operation of the parent; this is known as polymorphism.
- Graphically, generalization is rendered as a solid directed line with a large open arrowhead, pointing to the parent.
- A class may have zero, one, or more parents.
- A class that has no parents and one or more children is called a root class or a base class.
- A class that has no children is called a leaf class.

55

## Relationships- Generalization

- A class that has exactly one parent is said to use single inheritance; a class with more than one parent is said to use multiple inheritance.

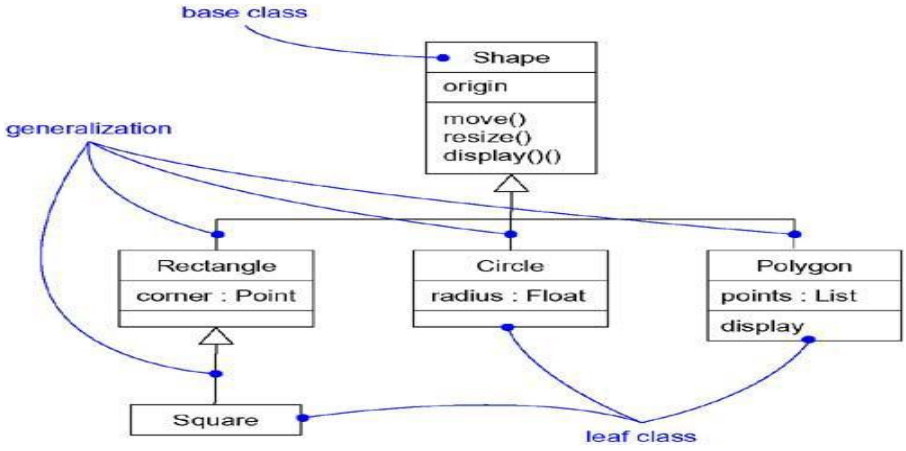
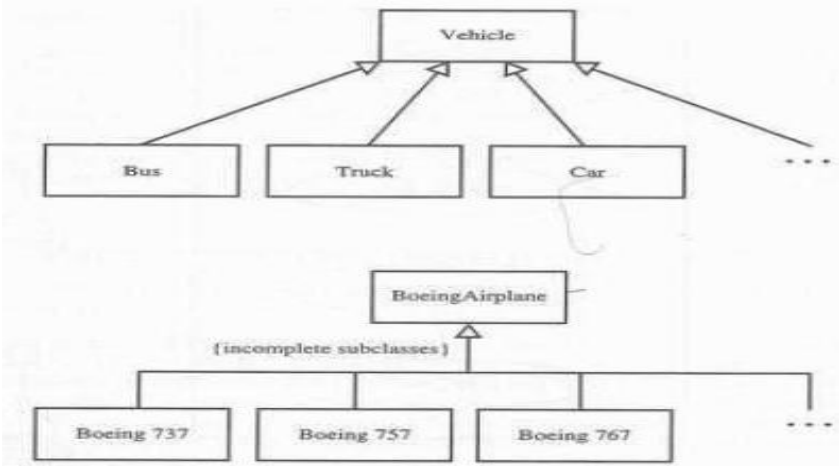



Figure: Generalization

56

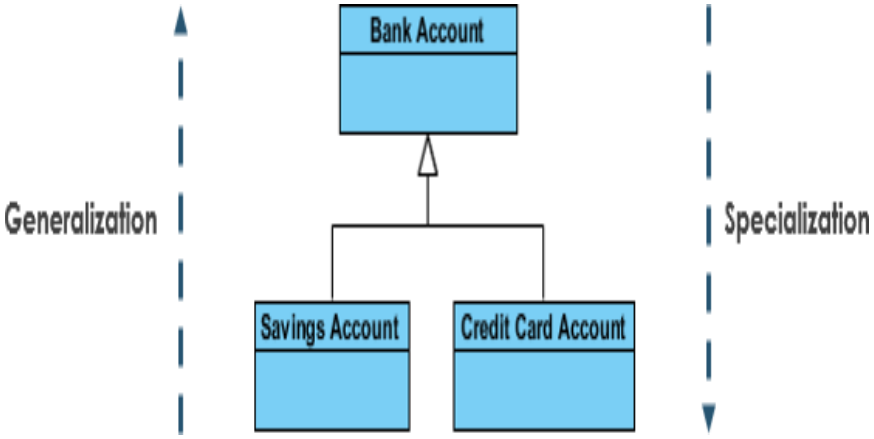
## Relationships- Generalization

- Ellipses indicate that the generalization is incomplete and more subclasses exist that are not shown.
- If the text label is placed on hollow triangle, it applies to all the paths.




## Relationships - Specialization

- Specialization is the reverse process of Generalization means creating new sub-classes from an existing class.




58

## Relationships – Generalization vs Specialization

Generalization and specialization both refer to inheritance but the approach in which they are implemented are different.

If many similar existing classes are combined to form a super class to do common job of its subclass, then it is known as **Generalization**.

If some new subclasses are created from an existing super class to do specific job of the super class, then it is known as **Specialization**.

59

## Relationships- Association

- An association is a structural relationship that specifies that objects of one thing are connected to objects of another.
- Given an association connecting two classes, you can navigate from an object of one class to an object of the other class, and vice versa.
- It's quite legal to have both ends of an association circle back to the same class. This means that, given an object of the class, you can link to other objects of the same class.
- An association that connects exactly two classes is called a binary association.
- Although it's not as common, you can have associations that connect more than two classes; these are called n-ary associations.

60

## Relationships- Association

- Graphically, an association is rendered as a solid line connecting the same or different classes. Use associations when you want to show structural relationships.
- An association can have a name, and you use that name to describe the nature of the relationship.
- So that there is no ambiguity about its meaning, you can give a direction to the name by providing a direction triangle that points in the direction you intend to read the name employs

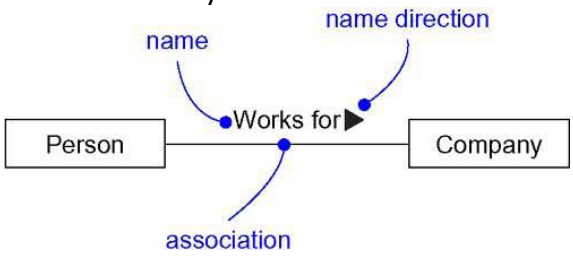


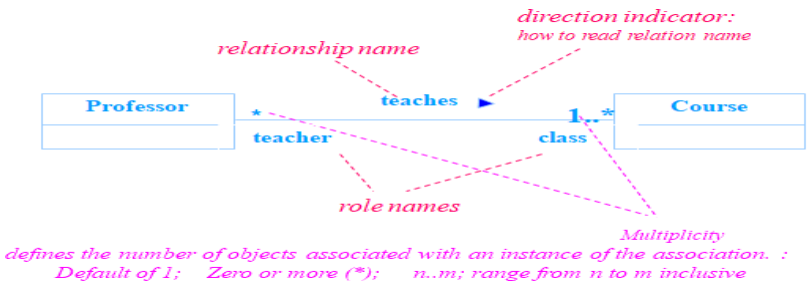

Figure: Association Names

61

## Relationships- Association

- UML uses term association navigation or navigability to specify role.

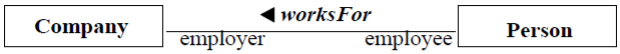

- Person class can't know anything about BankAccount class, but BankAccount class can know about Person class

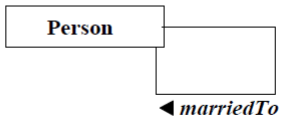



62

## Relationships- Association (Binary)

- A binary association is drawn as a solid path connecting two classes or both ends may be connected to the same class.

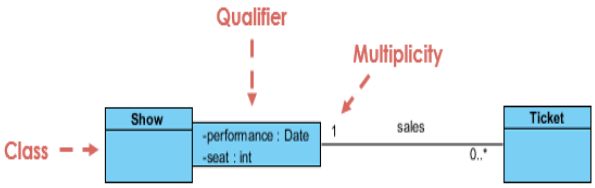


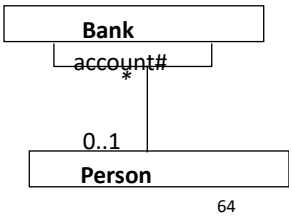




63

## Relationships- Association

- Role : The end of an association, where it connects to a class, shows the association role
  - Role is a part of association, not class
  - Each association has two or more roles to which is connected
- Qualifier : A qualifier is an association attribute. For example, a person object may be associated to a Bank object
  - An attribute of this association is the account#
  - The account# is the qualifier of this association

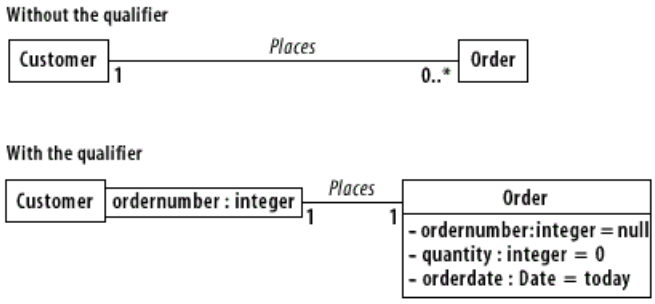



64

## Relationships- Association

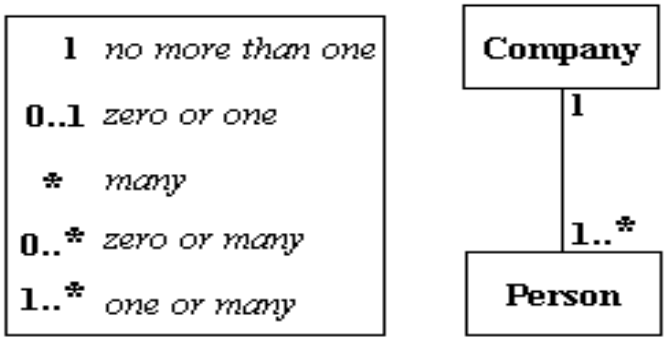
- Qualifier :




65

## Relationships- Association

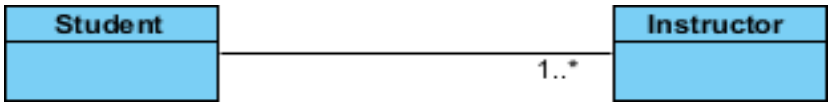
- Multiplicity : Multiplicity specifies the range of allowable associated classes.
- It is given for roles within associations, parts within compositions, repetitions, and other purposes.
- lower bound .. upper bound



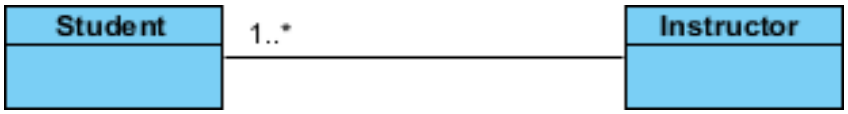

66

## Relationships- Association

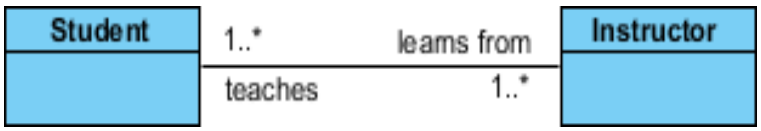
- A single student can associate with multiple teachers:




- The example indicates that every Instructor has one or more Students:



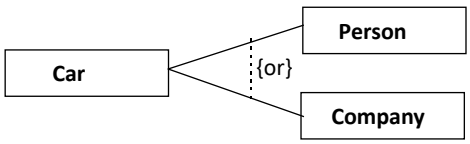

- We can also indicate the behavior of an object in an association (i.e., the role of an object) using role names.



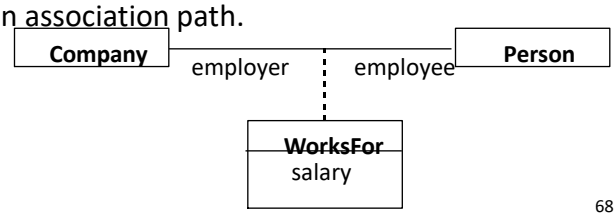

67

## Relationships- Association

- An OR association indicates a situation in which only one of several potential associations may be substantiated at one time for any single object.



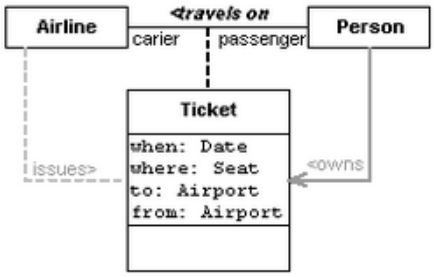

- An association class is an association that also has class properties.
- An association class is shown as a class symbol attached by a dashed line to an association path.




68

## Relationships- Association

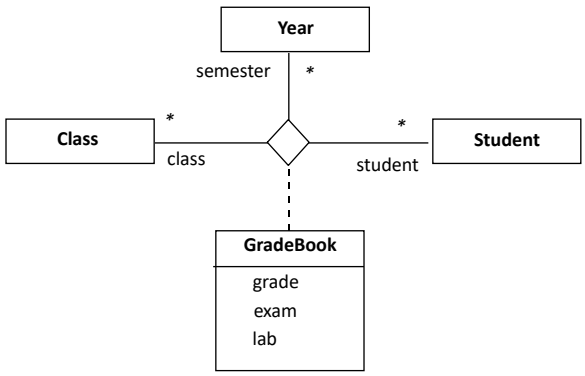
- Association class




69

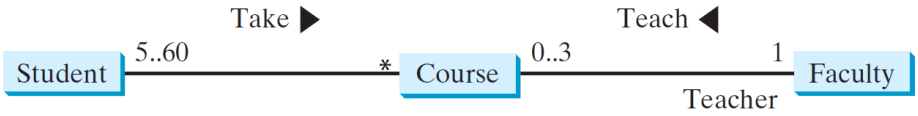
## Relationships- Association

- An n-ary association is an association among more than two classes.
- Since n-ary association is more difficult to understand, it is better to convert an n-ary association to binary association




70

## Relationships- Association

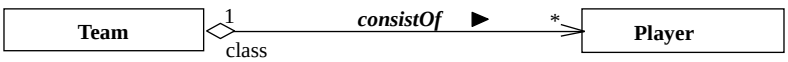



- Each student may take any number of courses, and each course must have at least 5 and at most 60 students. Each course is taught by only 1 faculty member, and a faculty member may teach from 0 to 3 courses per semester.
- **Note** :The character * means an unlimited number of objects, and the interval m..n indicates that the number of objects is between m and n, inclusively.
-

71

## Relationships- Aggregation

- Aggregation is a form of association
- A hollow diamond is attached to the end of the path to indicate aggregation
- Aggregation is used to represent ownership or a whole/part relationship

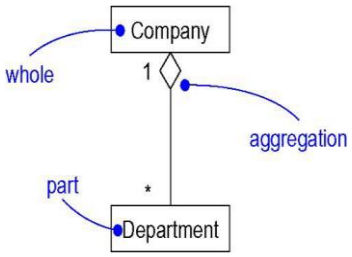



- Aggregation is a special form of association that represents an ownership relationship between two objects.
- Aggregation models has-a relationships.
- The owner object is called an aggregating object, and its class is called an aggregating class. The subject object is called an aggregated object, and its class is called an aggregated class.

72

## Relationships- Association

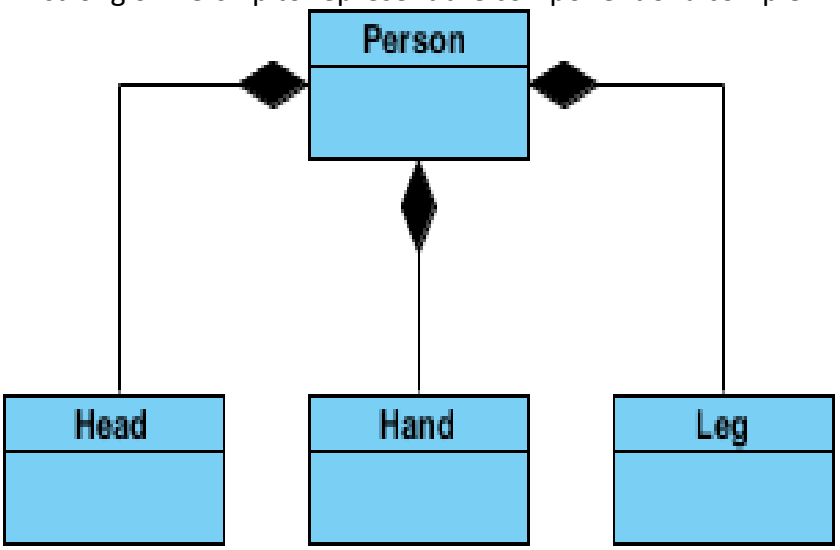
- Sometimes, you will want to model a "whole/part" relationship, in which one class represents a larger thing (the "whole"), which consists of smaller things (the "parts").




73

## Relationships- Composition

- Also known as the *a-part-of*, is a form of aggregation with strong ownership to represent the component of a complex



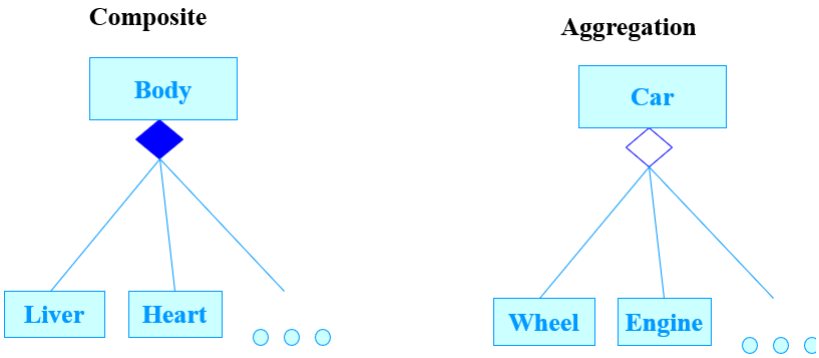

74

## Relationships- Composition

- The UML notation for composition is a solid diamond at the end of a path , and is used to represent an even stronger form of ownership.
- The composite object has sole responsibility for the disposition of its parts in terms of creation and destruction
- In implementation terms, the composite is responsible for memory allocation and deallocation
- The multiplicity of the aggregate end may not exceed one; i.e., it is unshared. An object may be part of only one composite at a time
- If the composite is destroyed, it must either destroy all its parts or else give responsibility for them to some other object
- A composite object can be designed with the knowledge that no other object will destroy its parts

75

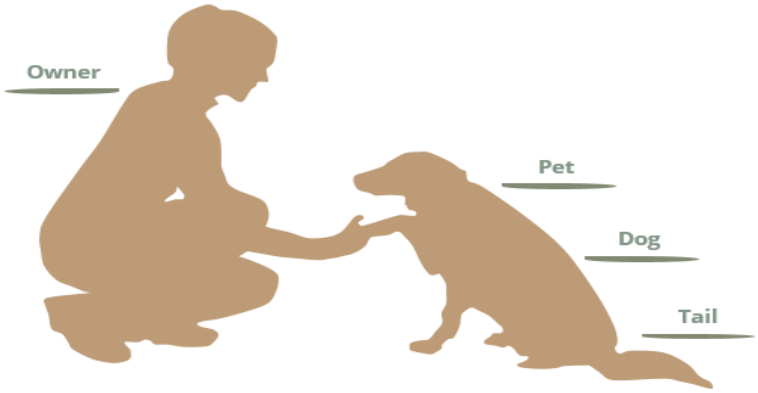
## Relationships




*Composite is a stronger form of aggregation.*
*Composite parts live and die with the whole.*

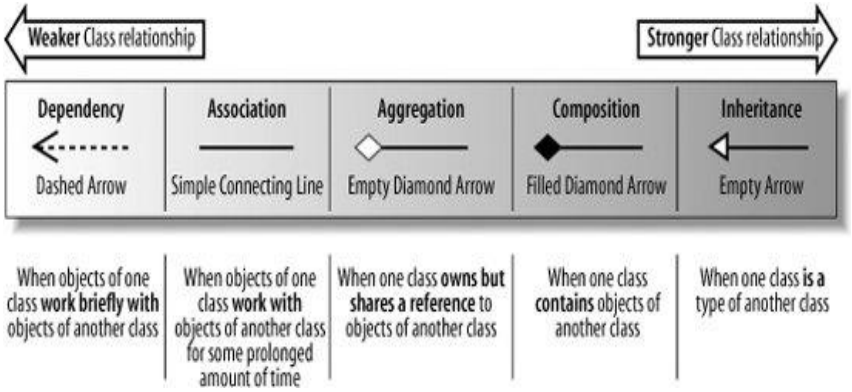
76

## Association • Aggregation • Composition




We see the following relationships:

- owners feed pets, pets please owners (**association**)
- a tail is a part of both dogs and cats (**aggregation / composition**)
- a cat is a kind of pet (**inheritance / generalization**)



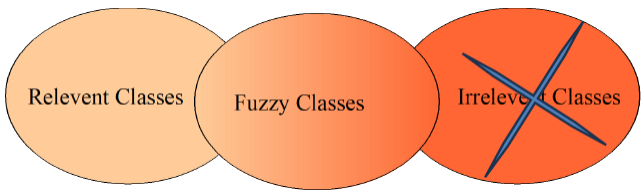
## Approaches for identifying Class

- Following are the alternative approaches to identifying classes
  - The noun phrase approach
  - The common class patterns approach
  - The use-case driven approach
  - The class responsibilities collaboration (CRC) approach

79

## Noun Phrase Approach

- Using this method, you have to read through the Use cases, interviews, and requirements specification carefully, looking for noun phrases
- Nouns in the textual description are classes and verbs are methods of the classes.
- Change all plurals to singular and make a list, which can then be divided into three categories




80

## Guidelines For Identifying Classes

- Look for nouns and noun phrases in the problem statement
- Some classes are implicit or taken from general knowledge
  - E.g if library system : Book, member etc.
  - If Bank system : customer, Bank etc.
- A noun means a content word that can be used to refer to a person, place, thing, quality, or action
- All classes must make sense in the application domain.
- **Redundant Classes:**
  - Do not keep two classes that express the same information.
  - If more than one word is being used to describe the same idea, select the one that is the most meaningful in the context of the system.
  - While choosing vocabulary always select the word that is used by the user of the system.

81

## Guidelines For Identifying Classes

- **Adjective Classes:**
  - An adjective is a word that expresses an attribute of something or the word class that qualifies nouns.
  - Adjectives can be used in many ways. It modifies a noun. It describes quality, state or action that noun refers to.
  - Adjective comes before nouns e.g. a new car
  - Size : big, tiny
  - Age : new, young, old
  - Shape : square, color, material etc.
  - Adjective can come after verbs e.g. your friend looks nice, dinner smells good
  - Adjective can suggest different kind of object, different use of same object or can be irrelevant.

82

## Guidelines For Identifying Classes

- **Adjective Classes:**
  - Does the object represented by the noun behave differently when the adjective is applied to it?
  - N.B. An adjective is a word that expresses an attribute of something or the word class that qualifies nouns
  - If the use of the adjective signals that the behavior of the object is different, then make a new class
  - For example, If Adult Membership and Youth Membership behave differently, than they should be classified as different classes
- **Attribute Classes:**
  - Tentative objects which are used only as values should be defined or restated as attributes and not as a class
  - For example the demographics of Membership are not classes but attributes of the Membership class
  - E.g. client status is attribute of client class.

83

## Guidelines For Identifying Classes

- **Adjective Classes:**
  - Does the object represented by the noun behave differently when the adjective is applied to it?
  - N.B. An adjective is a word that expresses an attribute of something or the word class that qualifies nouns
  - If the use of the adjective signals that the behavior of the object is different, then make a new class
  - For example, If Adult Membership and Youth Membership behave differently, than they should be classified as different classes
- **Attribute Classes:**
  - Tentative objects which are used only as values should be defined or restated as attributes and not as a class
  - For example the demographics of Membership are not classes but attributes of the Membership class
  - E.g. client status is attribute of client class.

84

## Guidelines For Identifying Classes-Noun Phrase

1. **Describe the software product in paragraph.**
2. **Identifying Tentative classes by identify the noun (exclude those that lie outside the problem boundary).**
3. **Selecting classes from the Relevant and fuzzy categories.**
4. **Initial List of Noun phrases: candidate classes.**
5. **Reviewing the Redundant classes and Building a common vocabulary.**
6. **Reviewing the classes containing Adjectives.**
7. **Reviewing the possible Attributes.**
8. **Reviewing the class purpose.**

85

## Case Study-ViaNet Bank ATM

- **Description of the ViaNet bank ATM system's requirements**
- The bank client must be able to deposit an amount to and withdraw an amount from his or her accounts using the touch screen at the ViaNet bank ATM kiosk. Each transaction must be recorded, and the client must be able to review all transactions performed against a given account. Recorded transactions must include the date, time transaction type, amount and account balance after the transaction.
- A ViaNet bank client can have two types of accounts: a checking account and savings account. For each checking account, one related savings account can exist.
- Access to the ViaNet bank accounts is provided by a PIN code consisting of four integer digits between 0 and 9.
- One PIN code allows access to all accounts held by a bank client.
- No receipts will be provided for any account transactions.

86

## Case Study-ViaNet Bank ATM

- The bank application operates for a single banking institution only.
- Neither a checking nor a savings account can have a negative balance. The system should automatically withdraw money from a related savings account if the requested withdrawal amount on the checking account is more than its current balance. If the balance on a savings account is less than the withdrawal amount requested, the transaction will stop and the bank client will be notified.

87

## ViaNet Bank ATM- Identifying classes-Noun Phrase Approach

- **Candidate classes**
- Account
- Account balance
- Amount
- Approval process
- ATM card
- ATM machine
- Bank
- Bank client
- Card
- Cash
- Check
- Current
- Current account
- Client
- Client's account
- Currency
- Envelope
- Four digits
- Fund
- Invalid PIN
- Message
- Money
- Password
- PIN
- PIN code
- Record
- Savings
- Savings account
- Step
- System
- Transaction
- Transaction history

88

## ViaNet Bank ATM- Identifying classes-Noun Phrase Approach

- Remove **irrelevant** classes as they do not belong to problem statement.
  - Envelope
  - four digits
  - step
- **Redundant Classes**
- Client, bank client – Bank client
- Account, client's account – Account
- PIN, PIN code – PIN
- Current, Current account – Current account
- Savings, savings account - Savings account
- Fund, Money – Fund
- ATM card, Card – ATM card

89

## After eliminating redundant class

- Account
- Account balance
- Amount
- Approval process
- ATM card
- ATM machine
- Bank
- Bank client
- Cash
- Check
- Current account
- Currency
- Fund
- Invalid PIN
- Message
- Password
- PIN
- Record
- Savings Account
- System
- Transaction
- Transaction history

90

## Reviewing classes containing adjectives

- Does the object represented by the noun behave differently when the adjective is applied to it?
- If yes, make a new class
- Else class is irrelevant, we must eliminate it
- We have no classes containing adjectives that we can eliminate.

91

## Reviewing the Possible Attributes

- Identifying the noun phrases that are attributes, not classes.
- **Amount** : a value, not a class
- **Account Balance** : An attribute of Account class.
- **Invalid PIN** : It is only a value, not a class.
- **Password** : An attribute of BankClient class.
- **Transaction history** : An attribute of Transaction class
- **PIN** : An attribute possibly of BankClient class.

92

## Candidate class – Final

- Account
- Approval process
- ATM card
- ATM machine
- Bank
- Bank client
- Cash
- Check
- Current account
- Currency
- Fund
- Message
- Record
- Savings Account
- System
- Transaction

93

## Reviewing the class purpose

- ATM Machine class: Provides an interface to the ViaNet bank.
- ATM Card class: Provides a client with key to an account.
- BankClient class: A client is an individual that has checking account, and possibly, a savings account.
- Bank class: Bank clients belong to the Bank. It is repository of accounts and processes the accounts transactions.
- Account class: An account class is a formal (or abstract) class , it defines the common behaviors that can be inherited by more specific classes such as Checking account and Savings Account.
- CheckingAccount class: It models a client's checking account and provides more specialized withdrawal service.
- SavingsAccount class: It models a client's savings account.
- Transaction class: Keeps track of transaction, time, date, type, amount, and balance.

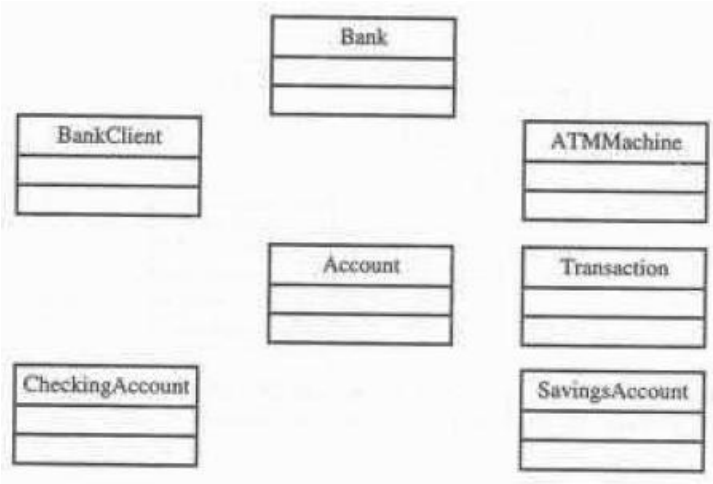
94

## How to draw class diagram

- **Step 1: Determine the class names.** The first step is to determine the system's primary objects.
- **Step 2: Separate relationships** The next step is to figure out how each of the classes or objects is connected to the others. Look for commonalities and abstractions between them to aid in grouping them while building the class diagram.
- **Step 3: Build the Structure** Add the class names first, then connect them with the required connectors. Attributes and functions/methods/operations can be added afterward

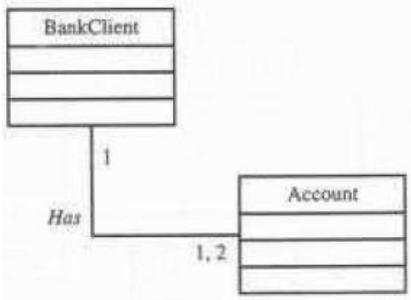
95

## Class diagram for ViaNet ATM banking system



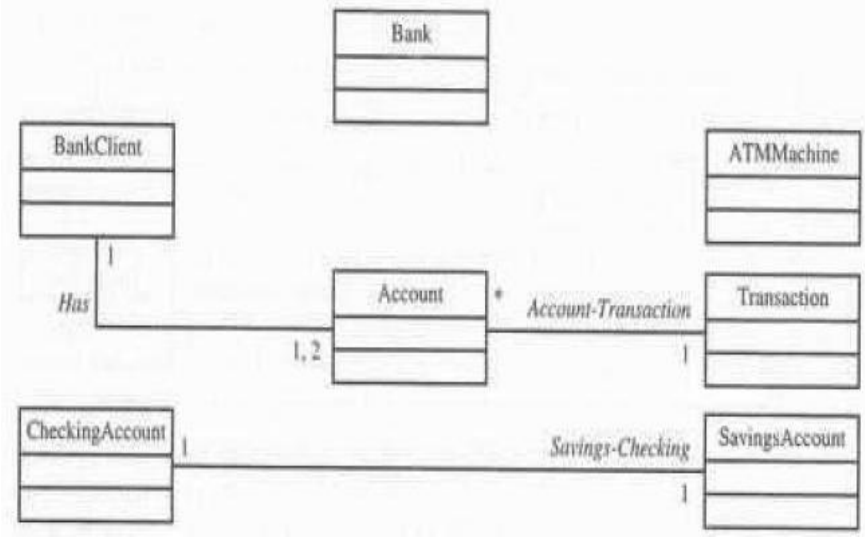

96

# Relationship



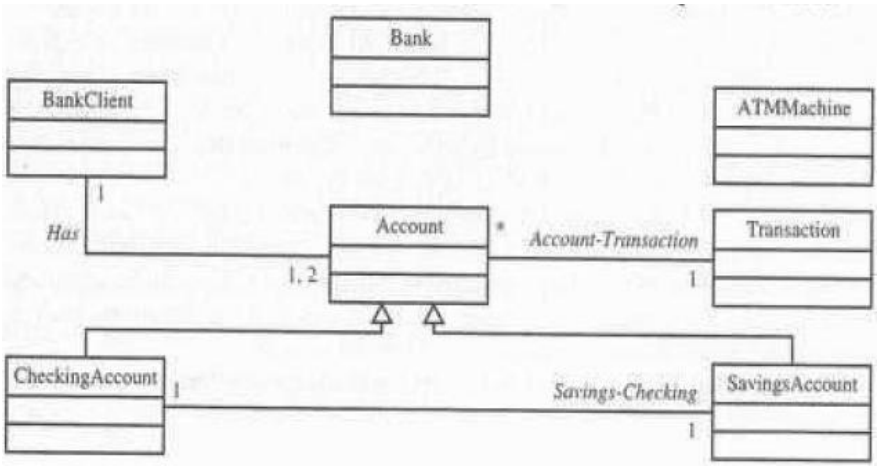

97

# Associations



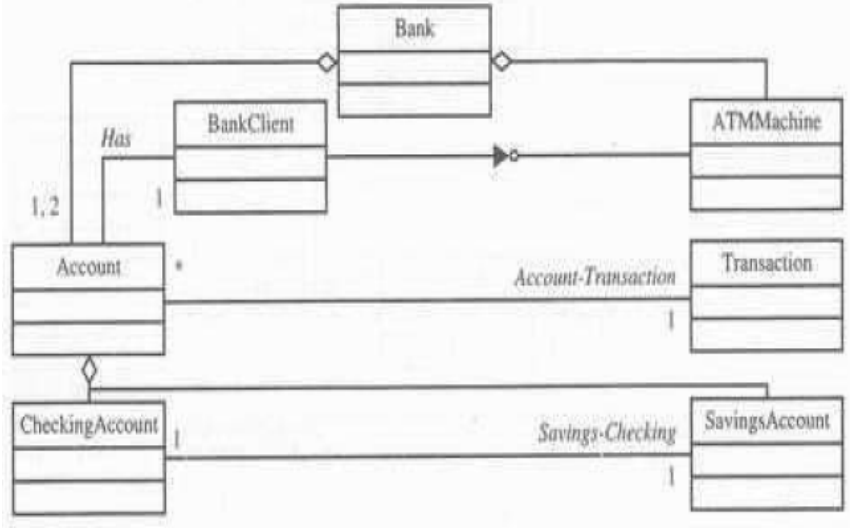

98

## Super-sub relationships



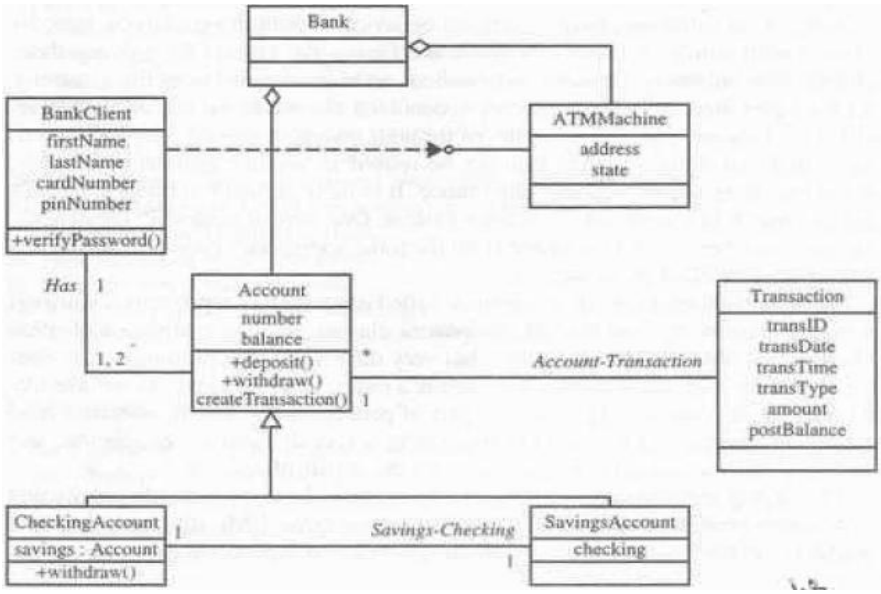

99

## Association, generalization, aggregation and interface



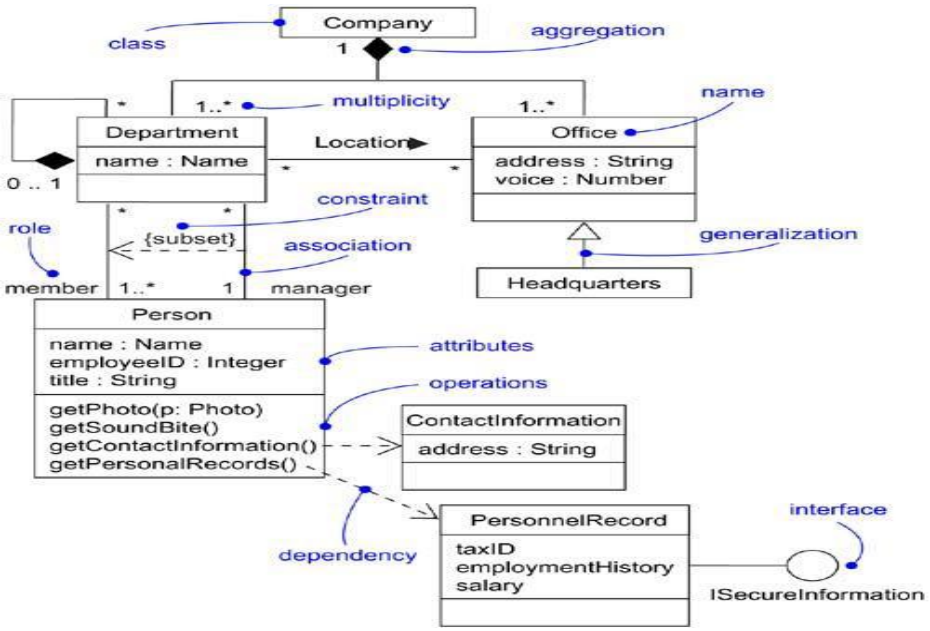

100

## ATM- with attributes and operations

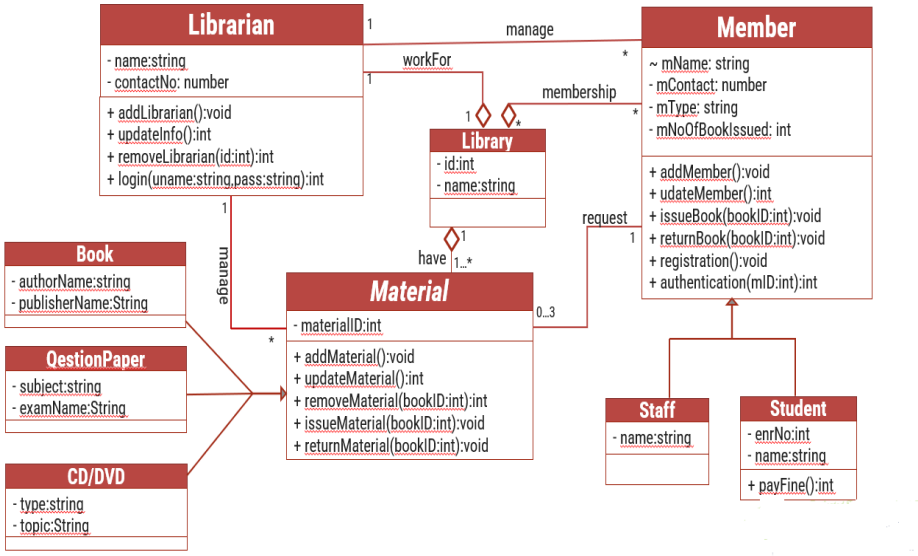



101

## Class Diagram

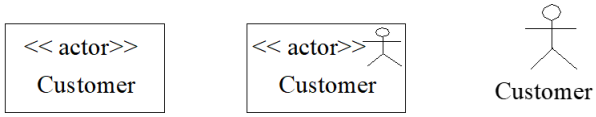


# Class Diagram Of Library Management System



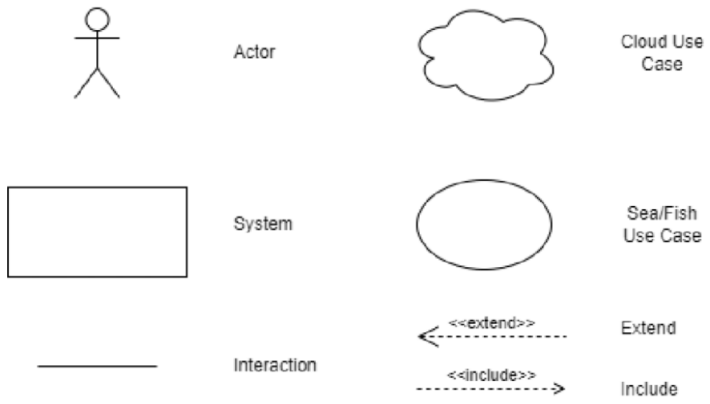

# Use Case Diagrams

- A Use Case is a description of a set of sequence of actions that a system performs that yields an observable result of value to a particular action
- The description of a use case defines what happens in the system when the use case is performed
- An Actor is someone or something that must interact with the system. An Actor initiates the process(that is USECASE)
- The three representations of an actor are equivalent




104

# Use Case Diagrams




105

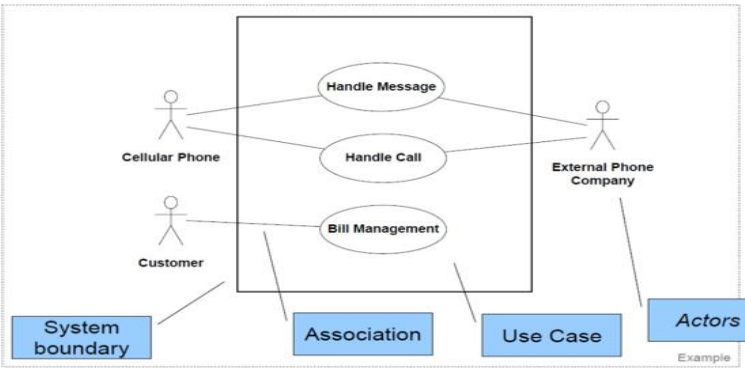
# Use Case Diagrams Notation

| Construct | Description | Notation |
|---|---|---|
| **Use-case** | A sequence of transactions performed by a system that produces a measurable result for a particular actor | Use Case |
| **Actor** | A coherent set of roles that users play when interacting with these use cases | Actor |
| **System Boundary** | The boundary between the physical system and the actors who interact with the physical system | ApplicationName |
| **Association** | The participation of an actor in a use case, i.e. an instance of an actor and instances of a use case communicating with each other | |
| **Extend** | A relationship between an *extension use case* and a *base use case, specifying how the* behavior of the extension use case can be inserted into the behavior defined for the base use case. The arrow head points to the base use case | <<extend>> |

106

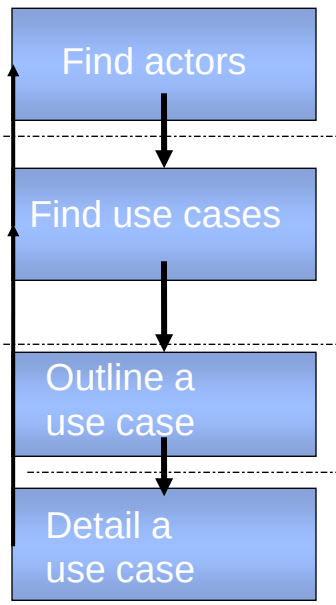
# Use Case Diagrams Notation

| Construct | Description | Notation |
|---|---|---|
| **Include** | A relationship between a *base use case and* an *inclusion use case, specifying how the* behavior for the inclusion use case is inserted into the behavior defined for the base use case. The arrow head points to the inclusion use case | <<include>> ----------> |




107

# Process of writing Use Case




**Important Note:**
Use case writing is an *iterative* process

Registrar

Close Registration

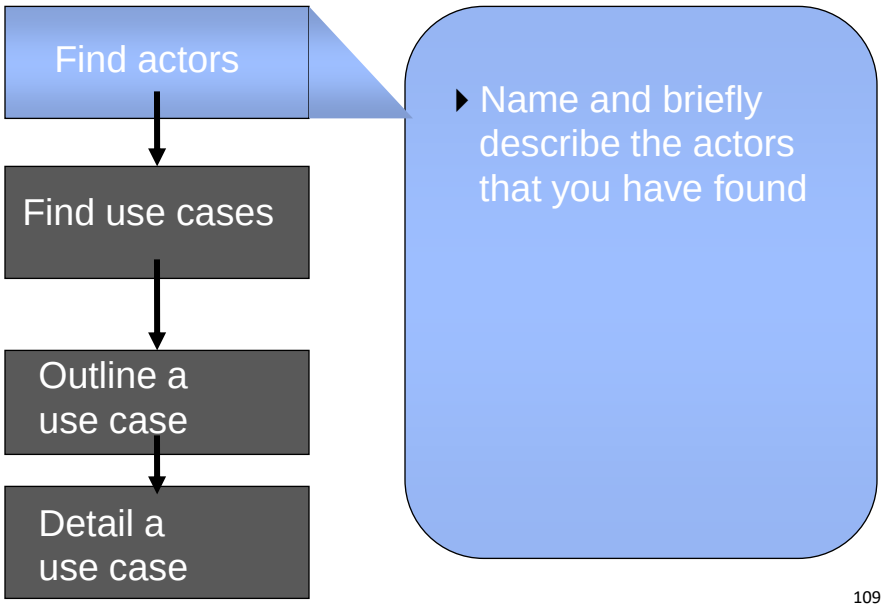
**Brief description**
close the registration process. Courses that do not have enough students are cancelled. The Billing System is notified for each student in each course that is not cancelled, so the student can be billed.

**Close Registration Outline**
Flow of events
-Step by step

**Close Registration Use-Case Specification**
Detailed Flow of Events
-Special Requirements
-Pre/Postconditions

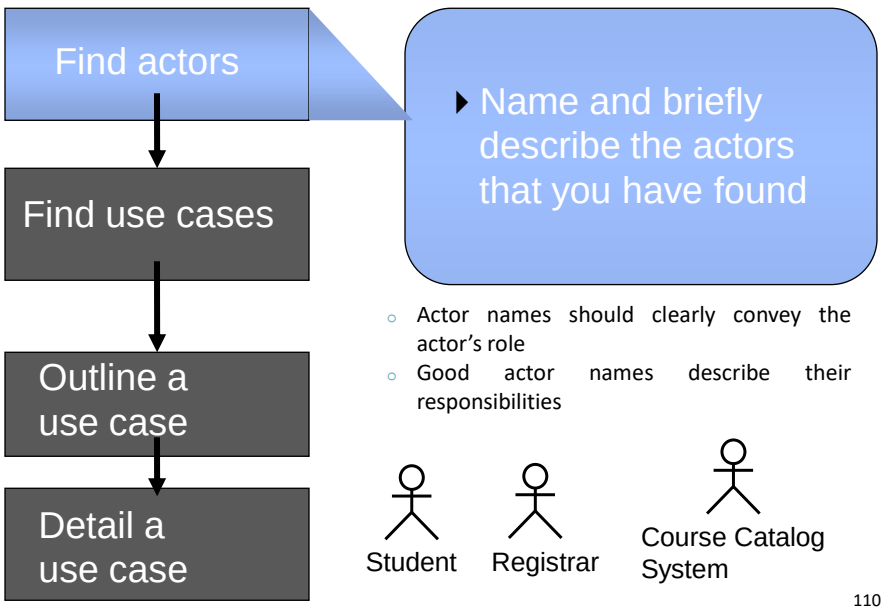
108

## Process of writing Use Case




109

## Process of writing Use Case



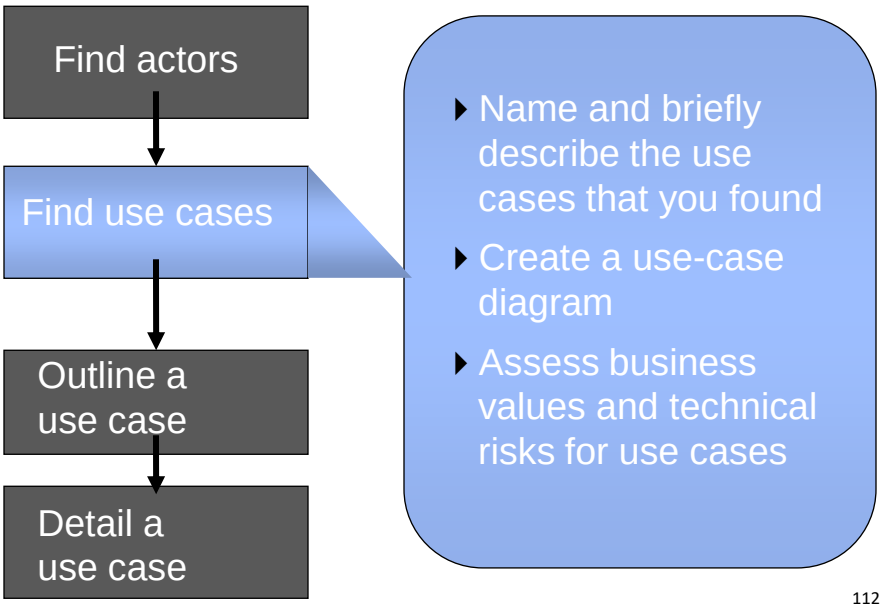

110

## Describe the Actor

- Name Student
- Brief description A person who signs up for a course
- Relationships with use cases


111

## Process of writing Use Case




112

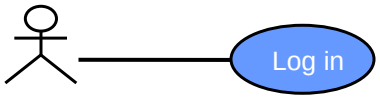
## Find Use Cases

- What are the goals of each actor?
  - Why does the actor want to use the system?
  - Will the actor create, store, change, remove, or read data in the system? If so, why?
  - Will the actor need to inform the system about external events or changes?
  - Will the actor need to be informed about certain occurrences in the system?
- Does the system supply the business with all of the correct behavior?

113

## Is Log in  a use case?

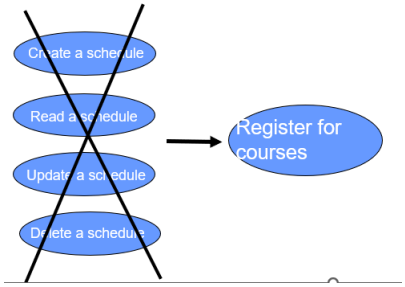
- By UML definition, log in  is not a use case, because it does not produce results of value to an actor.
- However, in many cases, there is a need to capture log in separately because it:
  - Captures more and more complex behaviors (security, compliance, customer experience)
  - Is included in other use cases
- Recommendation: Make an exception and capture log in as a separate use case.




114

## CRUD Use Cases

- A CRUD use case is a Create, Read, Update, or Delete use case
- Remove CRUD use cases if they are data- management use cases that do not provide results that are of value to actors

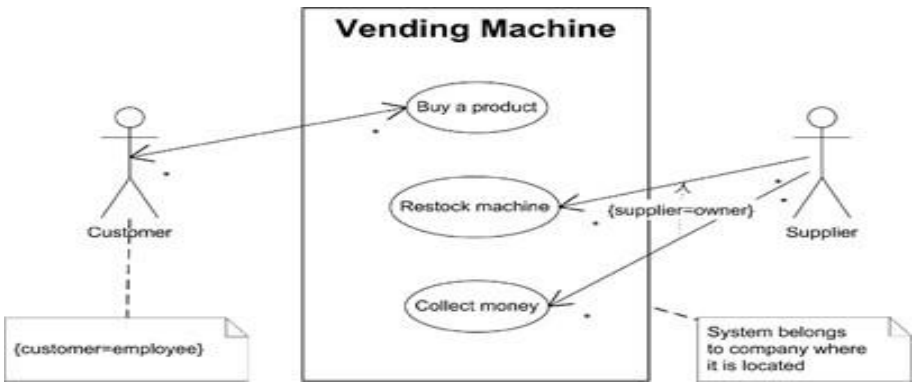



- **Do not confuse use cases with functions**
- **Focus on value**

115

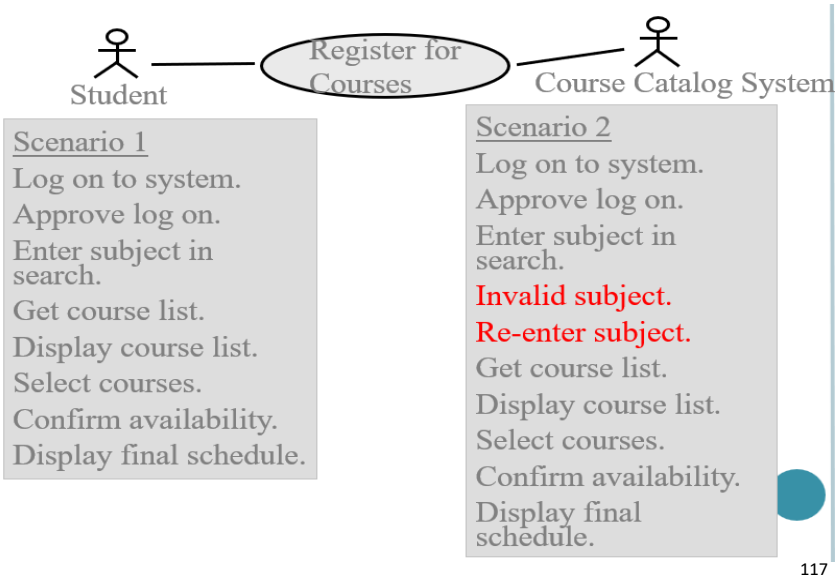
## Use Cases Example

- Introducing annotations (notes) and constraints




116

## A Scenario in Use Case




117

## Linking of Use Cases

- Association relationships
- Generalization relationships
  - One element (child) "is based on" another element (parent)
- Include relationships
  - One use case (base) includes the functionality of another (inclusion case)
  - Supports re-use of functionality
- Extend relationships
  - One use case (extension) extends the behavior of another (base)

118

# Linking of Use Cases