OBJECT ORIENTED ANALYSIS AND DESIGN

Text Book: Object Oriented Systems development

Author: Ali Bahrami

Session - 1

INTRODUCTION

- An Overview of Object Oriented Systems Development
- Object Basics
- Object Oriented Systems Development Life Cycle

Object-oriented analysis and design

- Object-oriented analysis and design (OOAD) is a popular technical approach for
  - analyzing,
  - designing an application, system, or business
  - by applying the object oriented paradigm and
  - visual modeling throughout the development life cycles for better communication and product quality.
- Object-oriented programming (OOP) is a method
  - based on the concept of “objects",
  - which are data structures that contain data,
  - in the form of fields,
  - often known as attributes;
  - and code, in the form of procedures,
  - often known as methods.

- **What is OOAD?-** Object-oriented analysis and design (OOAD) is a software engineering approach that models a system as a group of interacting objects .

- **Analysis** — understanding, finding and describing concepts in the problem domain.

- **Design** — understanding and defining software solution/objects that represent the analysis concepts and will eventually be implemented in code.

- **OOAD** - A software development approach that emphasizes a logical solution based on objects.

- Software development is dynamic and always undergoing major change.

- System development refers to all activities that go into producing information system solution.

- System development activities consist of
  - system analysis,
  - modelling,
  - design,
  - implementation,
  - testing and maintenance.

- A software development methodology → series of processes → can lead to the development of an application.

- Practices, procedures, and rules used to develop software, totally based on system requirements

ORTHOGONAL VIEWS OF THE SOFTWARE

- Two Approaches,
  - **Traditional Approach**
  - **Objected-Oriented Approach**

- **TRADITIONAL APPROACH**

  - Collection of **programs or functions.**
  - A system that is designed for **performing certain actions**.
  - **Algorithms + Data Structures = Programs**.
  - Software Development Models (**Waterfall, Spiral, Incremental, etc..**)

Difference between Traditional and Object Oriented Approach

TRADITIONAL APPROACH	OBJECT ORIENTED SYSTEM DEVELOPMENT
Collection of procedures(functions)	Combination of data and functionality
Focuses on function and procedures, different styles and methodologies for each step of process	Focuses on object, classes, modules that can be easily replaced, modified and reused.
Moving from one phase to another phase is complex.	Moving from one phase to another phase is easier.
Increases duration of project	decreases duration of project
Increases complexity	Reduces complexity and redundancy

❖ OBJECT ORIENTED APPROACH

- OO development offers a different model from the traditional software → **based on functions and procedures.**
- software is a collection of **discrete object that encapsulate their data as well as the functionality.**
- Each object has **attributes (properties) and method (procedures).**
- software by building self contained modules or objects that can be easily **REPLACED, MODIFIED AND REUSED.**
- Objects **grouped in to classes and object are responsible for itself.**

BENEFITS OF OBJECT ORIENTATION

- **Faster development,**
- **Reusability,**
- **Increased quality**
- modeling the **real world and provides us with the stronger equivalence of the real world‘s entities (objects)**.
- Raising the level of abstraction to the point where **application can be implemented in the same terms as they are described.**

WHY OBJECT ORIENTATION

- OO Methods enables to develop **set of objects that work** together → software → similar to traditional techniques.

- It adapts to

  - **Changing requirements**
  - **Easier to maintain**
  - **More robust**
  - **Promote greater design**
  - **Code reuse**

- Others

  - **Higher level of abstraction**
  - **Seamless transition among different phases of software development.**
  - **Encouragement of good programming technique.**
  - **Promotion of reusability.**

OVERVIEW OF UNIFIED APPROACH

- The **unified approach (UA)** is a methodology for **software development.**

- **Booch, Rumbaugh, Jacobson methodologies** gives the best practices, processes and guidelines for OO oriented software development.

- Combines with the **object management groups in unified modelling language.**

- UA utilizes the **unified modeling language (UML)** which is a set of **notations and conventions** used to describe and **model an application.**

Layered Architecture

- UA uses **layered architecture to develop applications**.
- **Creates object** that **represent elements to the user** through **interface or physically stored in database.**
- The layered approach consists of **user interface, business, access layers.**
- This approach **reduces the interdependence of the user interface, database access and business control.**
- **More robust and flexible system.**

OBJECT BASICS

Goals:

The developer should

- Define Objects and classes
- Describe objects, methods, attributes and how objects respond to messages,
- Define Polymorphism, Inheritance, data abstraction, encapsulation, and protocol,
- Describe objects relationships,
- Describe object persistence,

OBJECT ORIENTED PHILOSOPHY

- The **programming languages** provide the programmers the way of **describing the process**.

- The **ease of description, reusability, extensibility, readability, computational efficiency, and ability to maintain** depends on languages used.

- **System Software** – Machine Understandable language (Integers, floating point numbers, chars, Addressing Modes,….)

- Eg., **Financial Investment** → Development of Financial Investment Machine **directly would reduce translation.**

- **Object-Oriented Programming Concepts** allows closer **ideas and terms** for the **development** of certain applications.

- **Financial Investment :**
  - Bond (data type) → character
  - Buy operation on a bond (+) →operation on a number

TRADITIONAL APPROACH

- The traditional approach to software development tends toward **writing a lot of code to do all the things that have to be done.**

- **Algorithmic Centric Methodology** – only the algorithm that can accomplish the task.

- **Data-Centric Methodology** - think about the data to build a structure based on the algorithm

- You are the only active entity and the **code is just basically a lot of building materials.**

OBJECT-ORIENTED APPROACH

- OO approach is more like **creating a lot of helpers**
- take on an **active role, a spirit**, that form a community whose **interactions become the application**.
- Reusable
- Modified
- Replaced

EXAMPLES OF OBJECT ORIENTED SYSTEMS

- In OO system , **"everything is object".**
- **A spreadsheet cell, bar chart, title in bar chart, report,**
- **numbers, arrays, records, fields, files, forms, an invoice, etc.**
- **A window object** is responsible for things like **opening, sizing, and closing itself.**
- A **chart object** is responsible for things like **maintaining data and labels even for drawing itself.**

WHAT IS AN OBJECT?

- The term object was first formally utilized in the **Simula language** to **simulate some aspect of reality.**

- Attributes or properties describe object‘s state (data) and methods (properties or functions) define its behavior.

- An object is an entity.
  - It knows things (has attributes)
  - It does things (provides services or has methods
  - Examples in next Slide …….

OBJECT'S ATTRIBUTES

- **Attributes represented by data type.**
- **They describe objects states.**
- **In the Car example the car's attributes are:**
- **color, manufacturer, cost, owner, model, etc.**

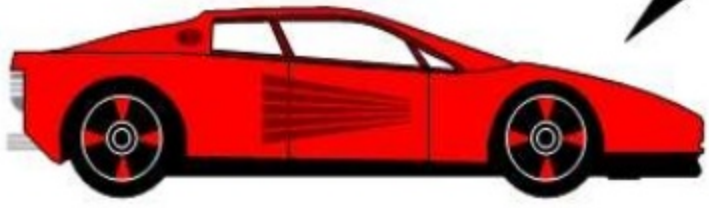
OBJECT'S METHODS

- **Methods define objects behaviour and specify the way in which an Object's data are manipulated.**
- **In the Car example the car's methods are:**
- **drive it, lock it, tow it, carry passenger in it.**

IT KNOWS THINGS (ATTRIBUTES)

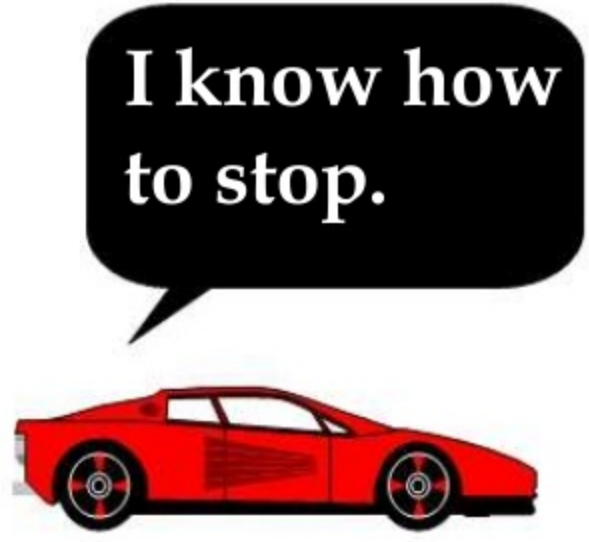
ATTRIBUTES

I am a Car.
I know my color, manufacturer, cost, owner and model.



IT DOES THINGS (METHODS)

METHODS



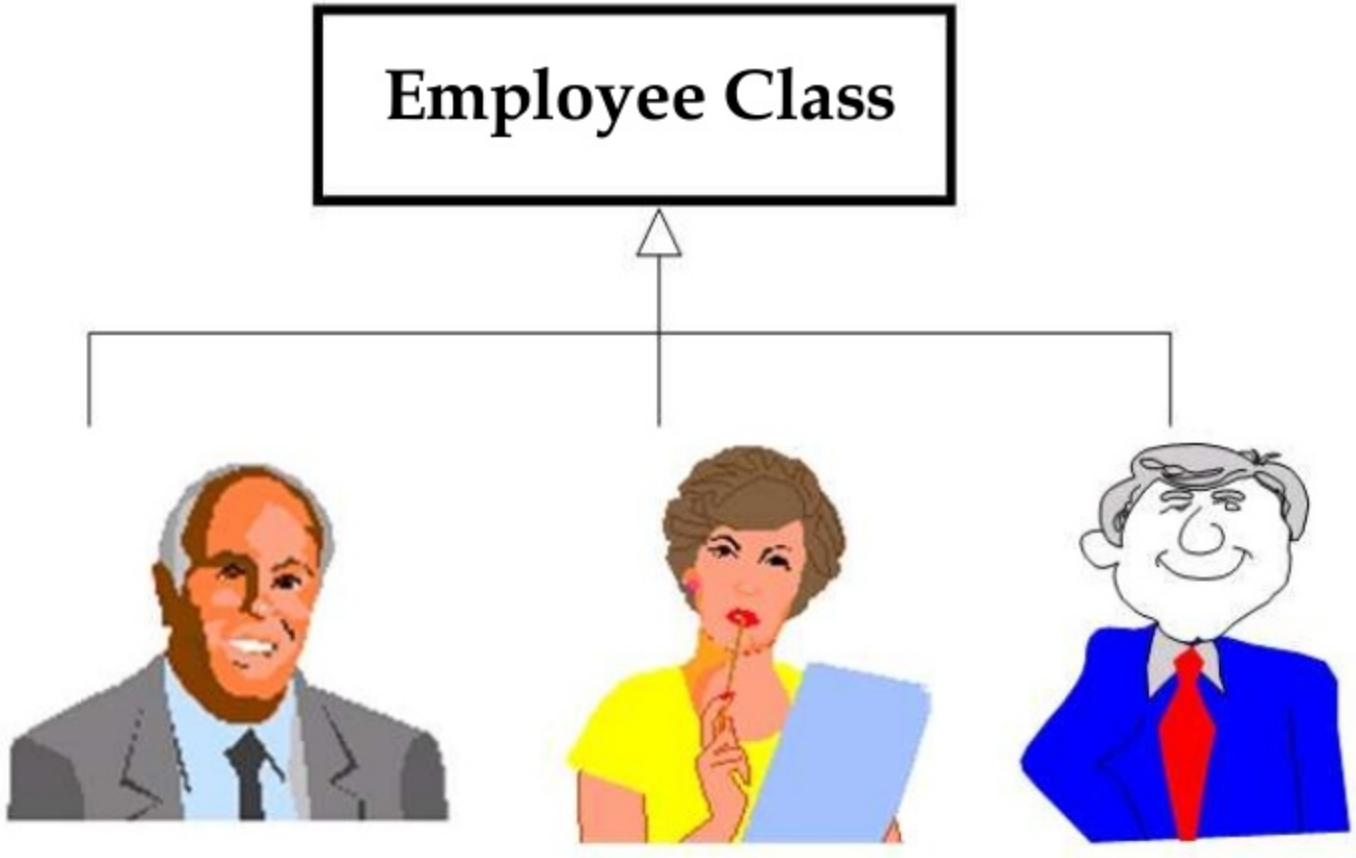
Object is whatever an application wants to talk about.

- For example, Parts and assemblies might be objects of bill of material application.

- Stocks and bonds might be objects of financial investment applications.

OBJECTS ARE GROUPED IN CLASSES

- The role of a class is to **define the attributes and methods** (the state and behaviour) **of its instances.**
- Used to **distinguish one type of object from the other.**
- Set of objects**, that share common methods, structure, behaviour.**
- **Single object** is simply an **instance of class.**
- 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."**


Employee Class
John object
Jane object
Mark object

SESSION - 2

OBJECTS AND CLASSES

Objects

- The concepts of objects and classes are intrinsically linked with each other and form the foundation of object–oriented paradigm.
- Identity that distinguishes it from other objects in the system.
- State that determines the characteristic properties of an object as well as the values of the properties that the object holds.
- Behavior that represents externally visible activities performed by an object in terms of changes in its state.

Class

- A class represents a collection of objects having same characteristic properties that exhibit common behavior.
- Creation of an object as a member of a class is called instantiation.
- Thus, object is an instance of a class.
- Example: **Circle** – a class

  x, to the center

  a, to denote the radius of the circle
- Some of its operations can be defined as follows:

  findArea(), method to calculate area

  findCircumference(), method to calculate circumference

Object oriented Methodologies

- Many methodologies have been developed for object oriented development.
- A methodology usually includes
  - Notation : Graphical representation of classes and their relationships with interactions.
  - Process : Suggested set of steps to carry out for transforming requirements into a working system.
  - Tool : Software for drawings and documentation

UML – Unified modeling language

- UML focuses on standard modeling language and not a standard process.

- UML focuses the concept of

  Booch,

  Rambaugh and

  Jacobson.

- The UML is a standard graphical design for object-oriented graphical design and a medium for presenting important analysis and design concepts.

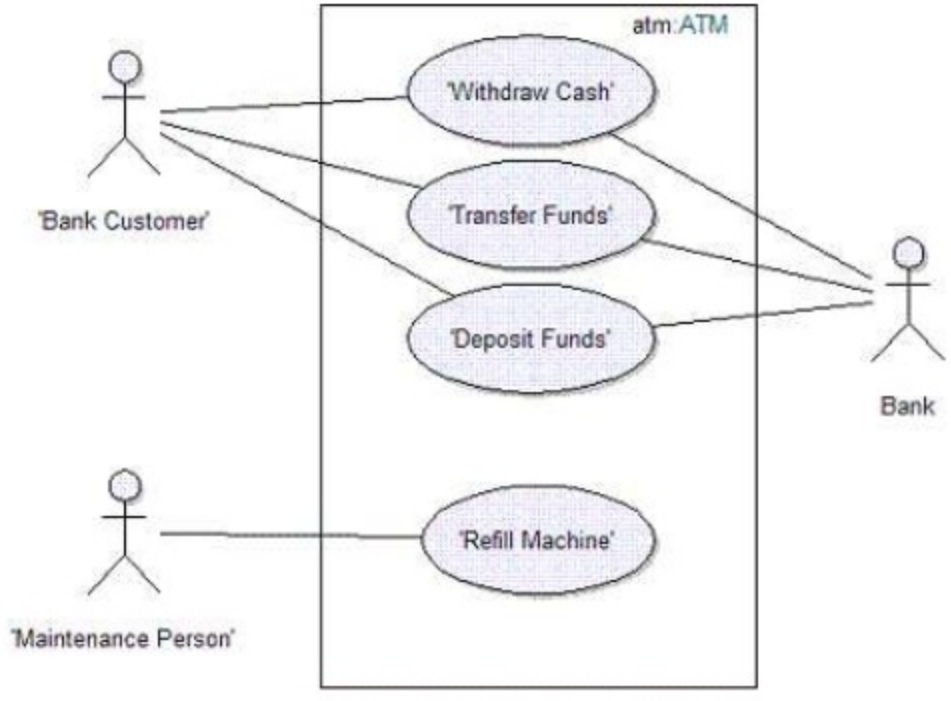
UML Diagrams

- Use Case Diagrams
- Class Diagrams
- Package Diagrams
- Interaction Diagrams
  - Sequence
  - Collaboration
- Activity Diagrams
- State Transition Diagrams
- Deployment Diagrams

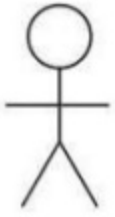
USE CASE DIAGRAMS

Use Case Diagrams

- A **use case diagram** at its simplest is a representation of a user's interaction with the system that shows the relationship between the user and the different use cases in which the user is involved.



Actor



An **actor instance is someone** or something outside the system that interacts with the system.

- An actor is anything that exchanges data with the system.
- An actor can be a user, external hardware, or another system.

How to Find Actors

- Actors:
- Supply/use/remove information
- Use the functionality.
- Will be interested in any requirement.
- Will support/maintain the system.
- The system's external resources.
- The other systems will need to interact with this one.

Documenting Actor Characteristics

Brief Description:

- What or who the actor represents?
- Why the actor is needed?
- What interests the actor has in the system?

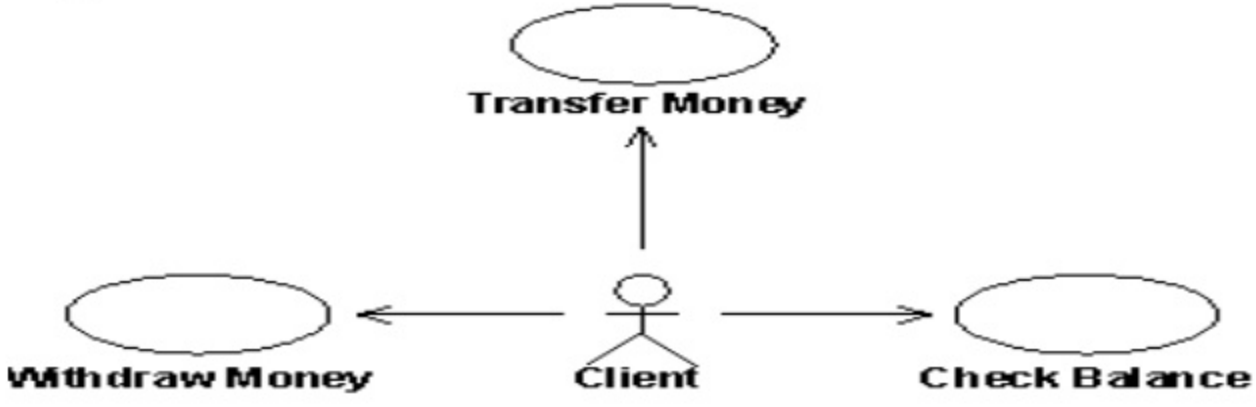
Actor characteristics might influence how the system is developed:

- The actor's scope of responsibility.
- The physical environment in which the actor will be using the system.
- The number of users represented by this actor.

Use case

- A set of scenarios that describes an interaction between a user and a system, including alternatives.

- Example



How to Find Use Cases

- What are the system tasks for each actor you have identified?
- Does the actor need to be informed about certain occurrences in the system?
- Will the actor need to inform the system about sudden, external changes?
- Does the system supply the business with the correct behavior?
- Can all features be performed by the use cases you have identified?
- What use cases will support and maintain the system?
- What information must be modified or created in the system?

Use cases types:

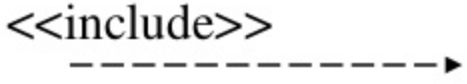
- System start and stop.
- Maintenance of the system (add user, …).
- Maintenance of data stored in the system.
- Functionality needed to modify behavior in the system.

Terminologies

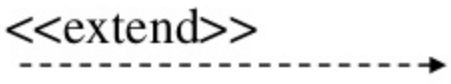
- **System boundary**: rectangle diagram representing the boundary between the actors and the system.
- **Use Case Diagram(core relationship)**
  Association: communication between an actor and a use case; Represented by a solid line.

- Generalization: relationship between one general use case and a special use case (used for defining special alternatives)
- Represented by a line with a triangular arrow head toward the parent use case.

- Include: a dotted line labeled <<include>> beginning at base use case and ending with an arrows pointing to the include use case. The include relationship occurs when a chunk of behavior is similar across more than one use case. Use "include" in stead of copying the description of that behavior.

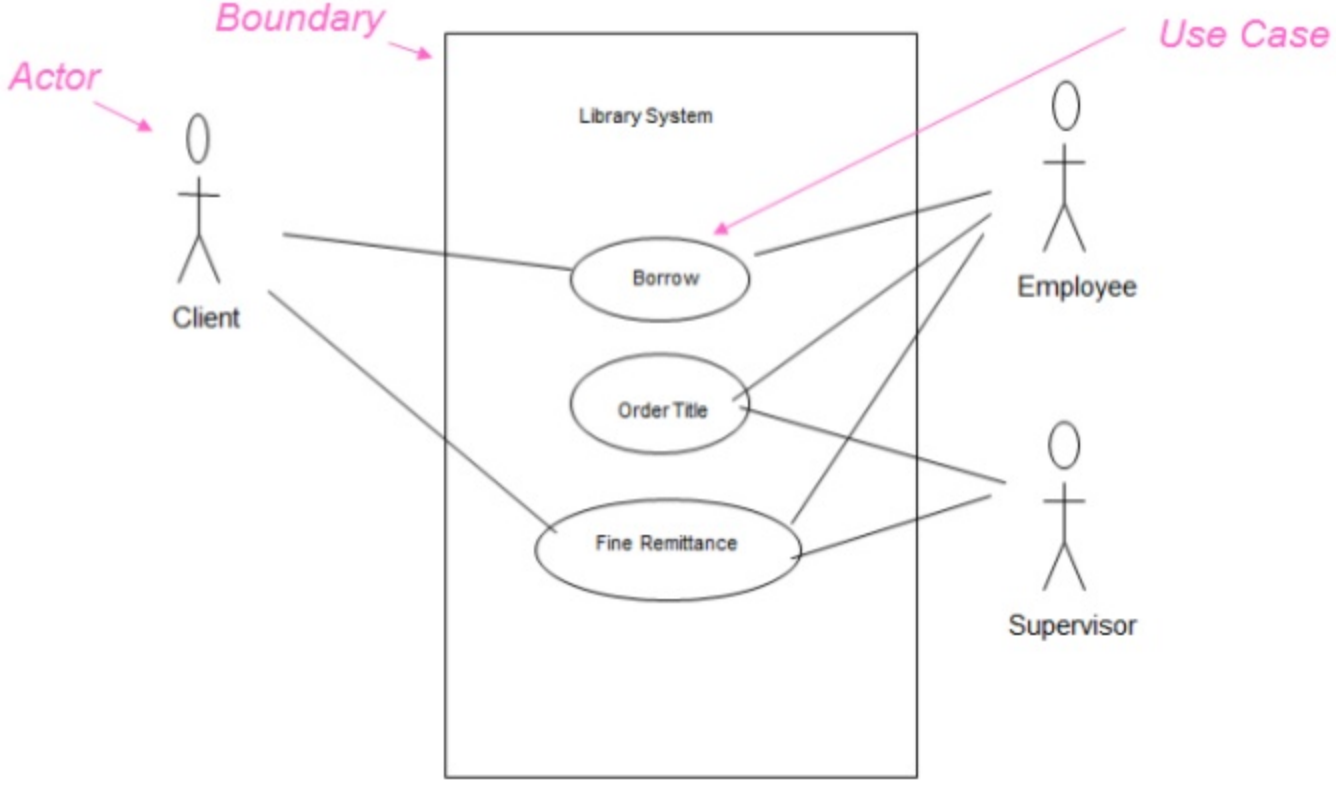


- Extend: a dotted line labeled <<extend>> with an arrow toward the base case. The extending use case may add behavior to the base use case. The base class declares "extension points".

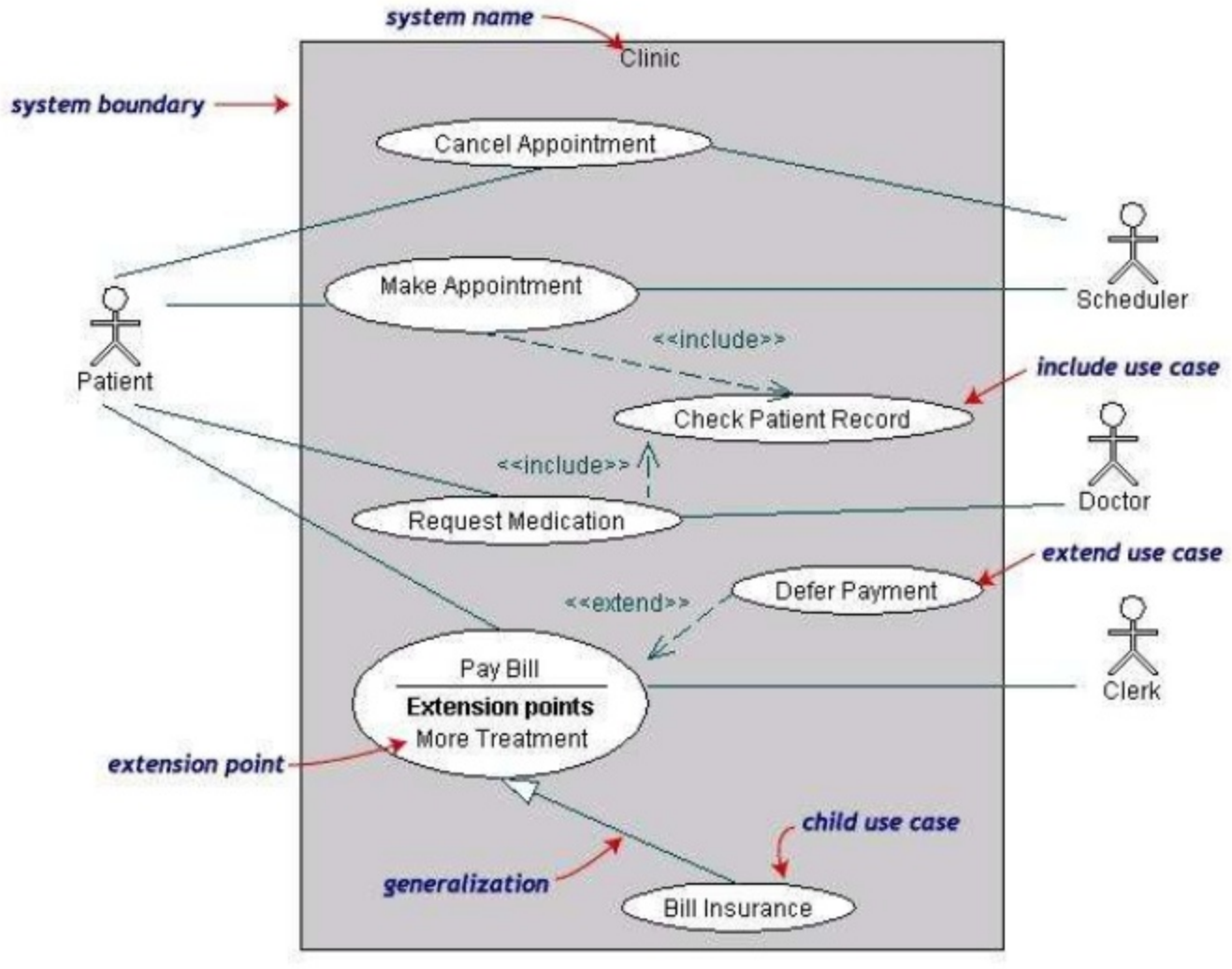


Example: Library management System

- A generalized description of how a system will be used.
- Provides an overview of the intended functionality of the system



Use Case Diagrams(cont.)



Continued…

- **Pay Bill** is a parent use case and **Bill Insurance** is the child use case. (generalization)

- Both **Make Appointment** and **Request Medication** include **Check Patient Record** as a subtask.(include)

- The **extension point** is written inside the base case
- **Pay bill**; the extending class **Defer payment** adds the behavior of this extension point. (extend)

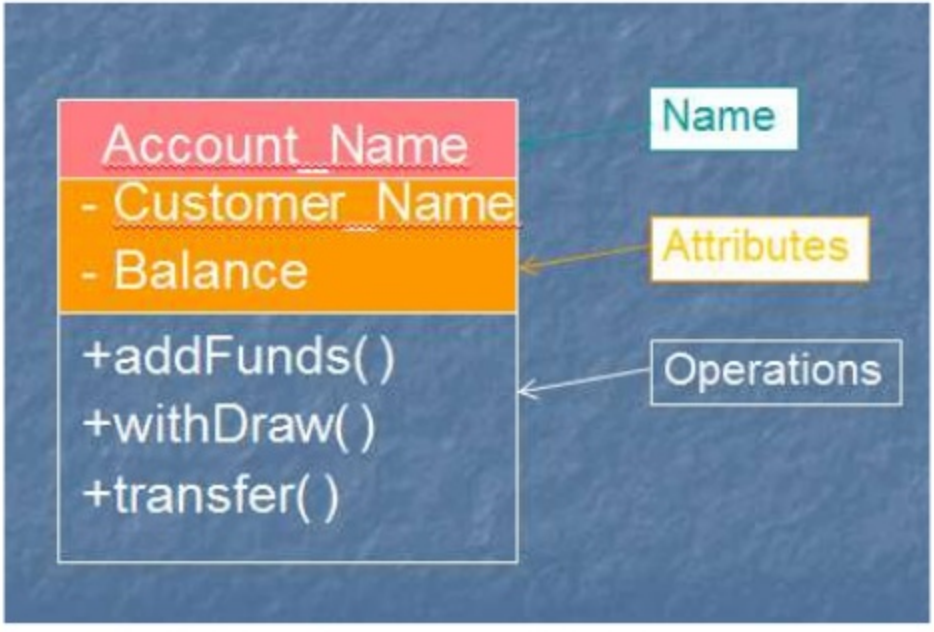
Class diagram

- Used for describing structure and behavior in the use cases
- Provides a conceptual model of the system in terms of entities and their relationships
- Used for requirement capture, end-user interaction
- Detailed class diagrams are used for developers

Class representation

- Each class is represented by a rectangle subdivided into three compartments
  - Name
  - Attributes
  - Operations
- Modifiers are used to indicate visibility of attributes and operations.
  - '+' is used to denote *Public* visibility (everyone)
  - '#' is used to denote *Protected* visibility (friends and derived)
  - '-' is used to denote *Private* visibility (no one)
- By default, attributes are hidden and operations are visible.

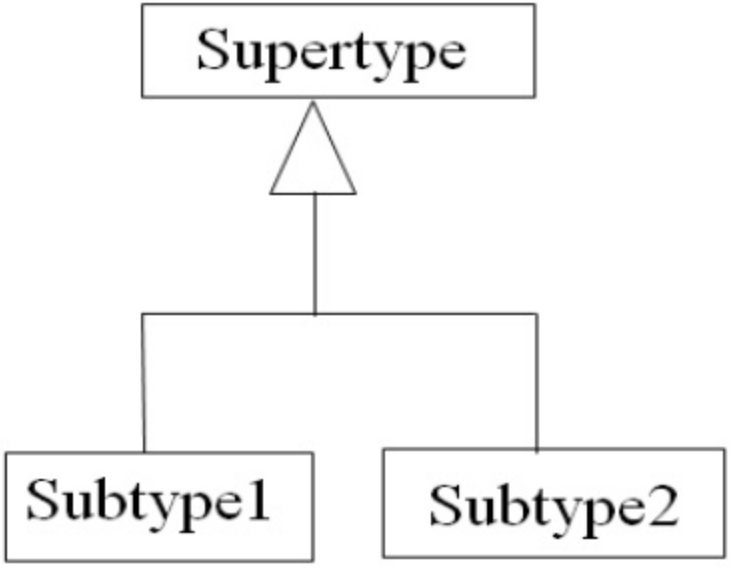
An example of Class



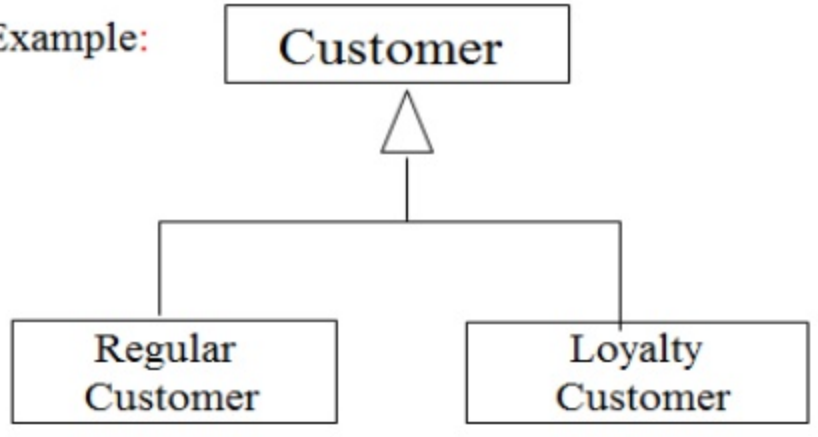
OO Relationships

- There are two kinds of Relationships
  - Generalization (parent-child relationship)
  - Association (student enrolls in course)
- Associations can be further classified as
  - Aggregation
  - Composition

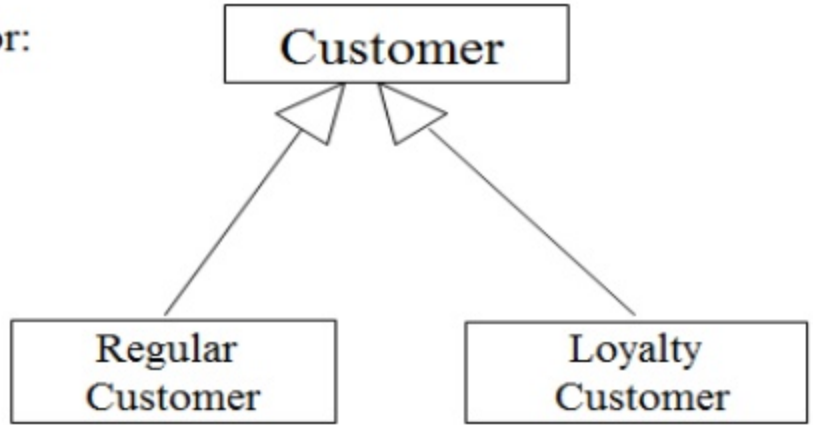
OO Relationships: **Generalization**



Example:



or:



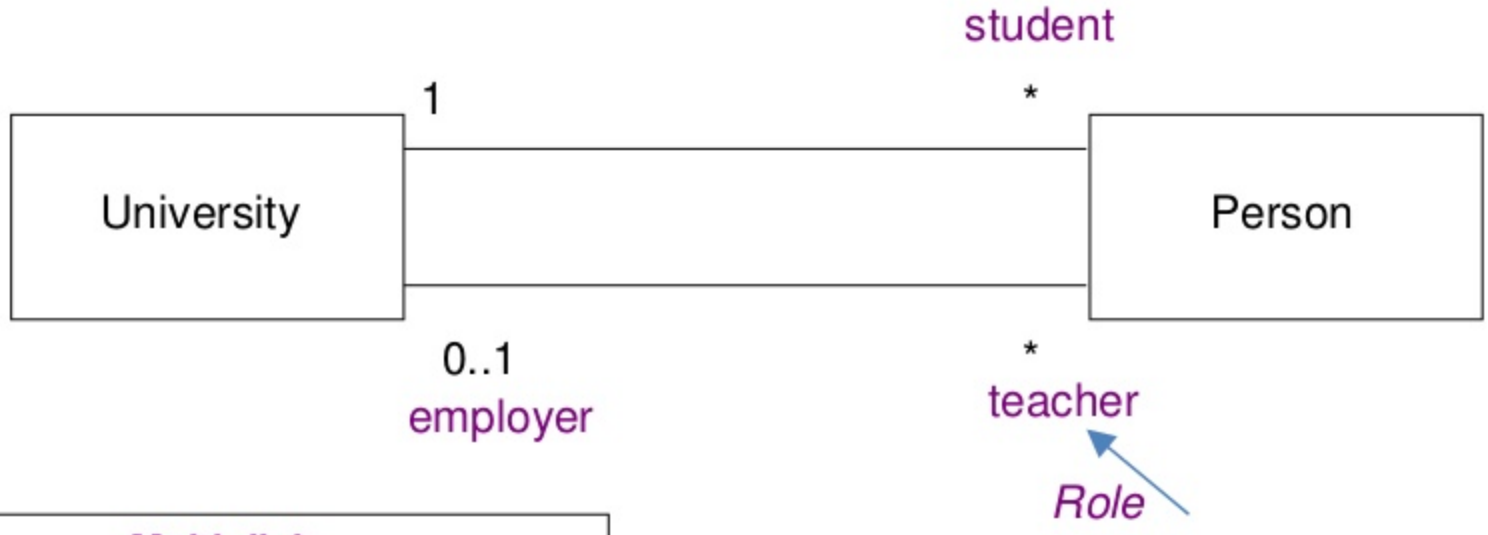
- Generalization expresses a parent/child relationship among related classes.

- Used for abstracting details in several layers

OO Relationships: **Association**

- Represent relationship between instances of classes
  - Student enrolls in a course
  - Courses have students
  - Courses have exams
  - Etc.
- Association has two ends
  - Role names (e.g. enrolls)
  - Multiplicity (e.g. One course can have many students)
  - Navigability (unidirectional, bidirectional)

Association: Multiplicity and Roles



Multiplicity

Symbol	Meaning
1	One and only one
0..1	Zero or one
M..N	From M to N (natural language)
*	From zero to any positive integer
0..*	From zero to any positive integer
1..*	From one to any positive integer

Role

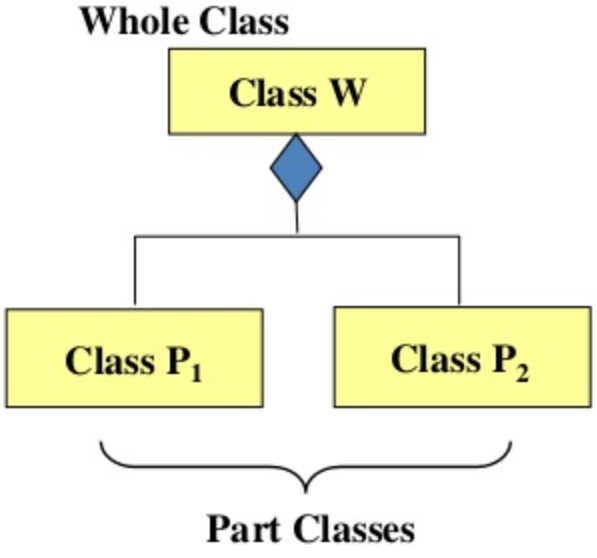
"A given university groups many people; some act as students, others as teachers. A given student belongs to a single university; a given teacher may or may not be working for the university at a particular time."

Association: Model to Implementation


```
Class Student {
    Course enrolls[4];
}

Class Course {
  Student have[];
}
```

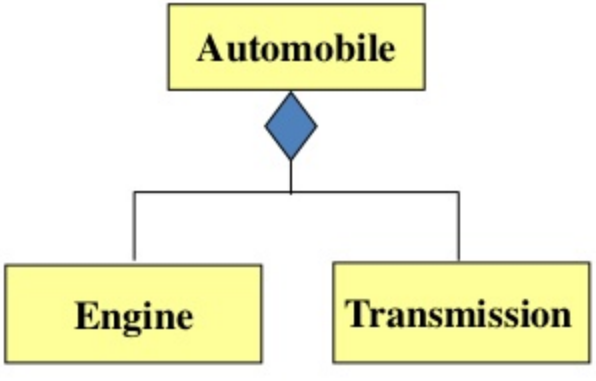
OO Relationships: **Composition**



Composition: expresses a relationship among instances of related classes. It is a specific **kind of Whole-Part** relationship.

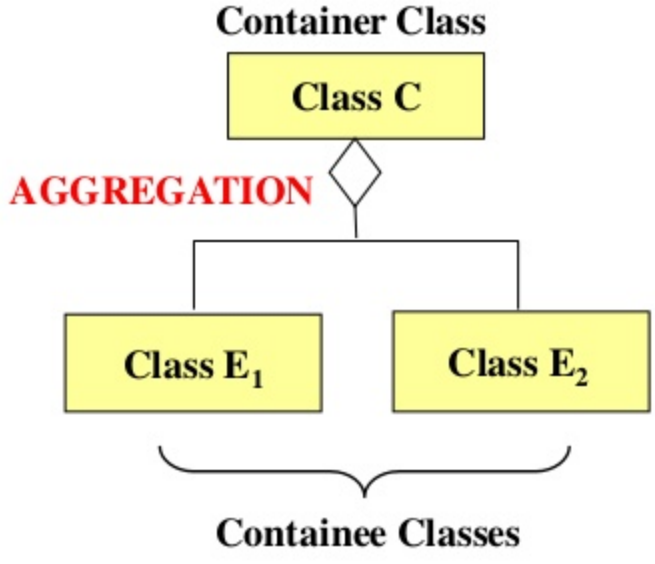
It expresses a relationship where an instance of the Whole-class has the responsibility to **create and initialize instances** of each Part-class.

Example



It may also be used to express a relationship where instances of the Part-classes have **privileged access or visibility** to certain attributes and/or behaviors defined by the Whole-class.

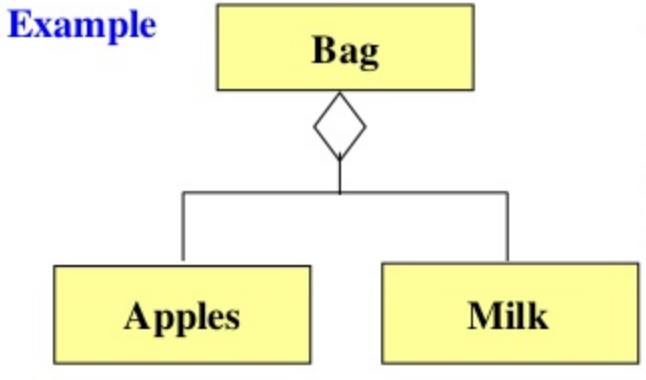
OO Relationships: **Aggregation**



Aggregation: expresses a relationship among instances of related classes. It is a specific kind of Container-Containee relationship.

It expresses a relationship where an instance of the Container-class has the responsibility to hold and maintain instances of each Containee-class that have been created outside the Container-class.

Aggregation should be used to express a more informal relationship than composition expresses. That is, it is an appropriate relationship where the Container and its Containees



Aggregation is appropriate when Container and Containees have no special access privileges to each other.

Aggregation vs. Composition

- **Composition** is really a strong form of **aggregation**
  - components have only one owner
  - components cannot exist independent of their owner
  - components live or die with their owner

  e.g. Each car has an engine that can not be shared with other cars.

- **Aggregations** may form "part of" the aggregate, but may not be essential to it. They may also exist independent of the aggregate.

  e.g. Apples may exist independent of the bag.

Session 3

Interaction Diagram -

Sequence diagram

Collaboration Diagram

Interaction Diagram

- From the name *Interaction* it is clear that the diagram is used to describe some type of interactions among the different elements in the model.
- So this interaction is a part of dynamic behavior of the system.

- This interactive behavior is represented in UML by two diagrams known as *Sequence diagram* and *Collaboration diagram*.

- The basic purposes of both the diagrams are similar.

- Sequence diagram emphasizes on time sequence of messages and collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

- The purposes of interaction diagram can be describes as:
  - To capture dynamic behavior of a system.
  - To describe the message flow in the system.
  - To describe structural organization of the objects.
  - To describe interaction among objects.

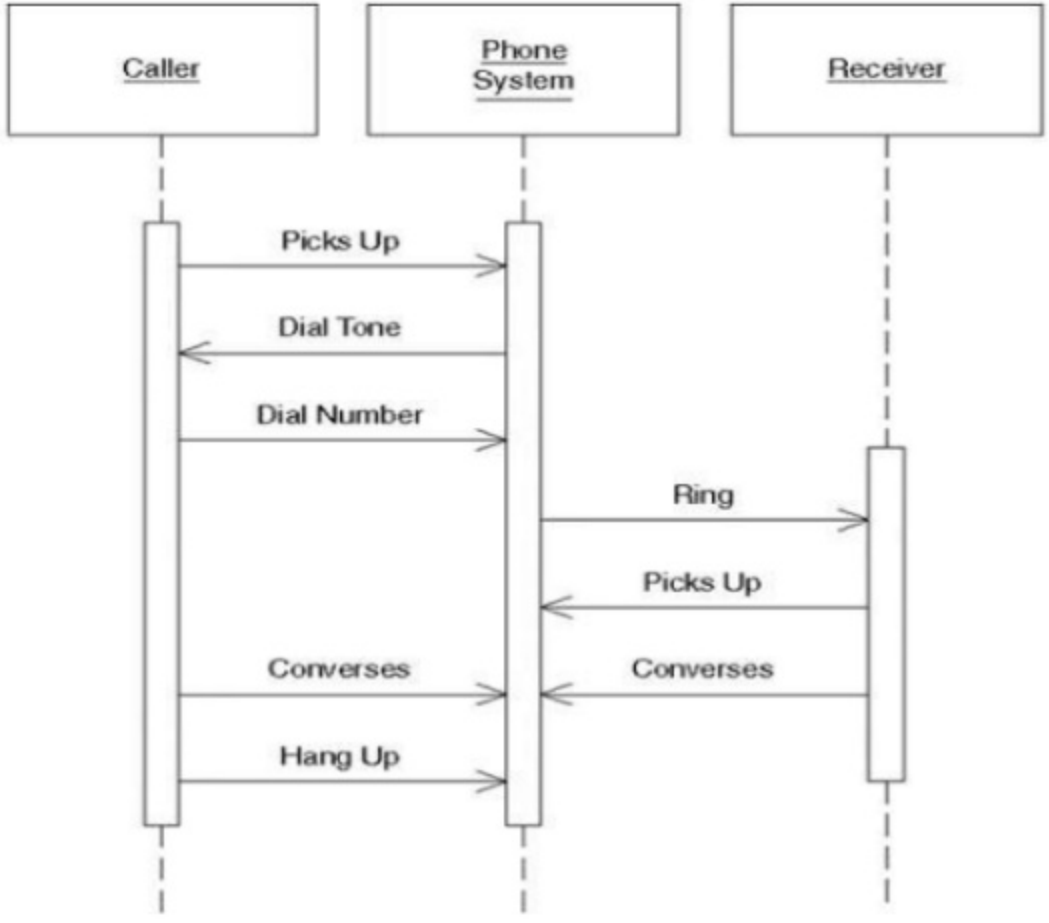
The following factors are to be identified clearly before drawing the interaction diagram:

- Objects taking part in the interaction.
- Message flows among the objects.
- The sequence in which the messages are flowing.
- Object organization.

Example :

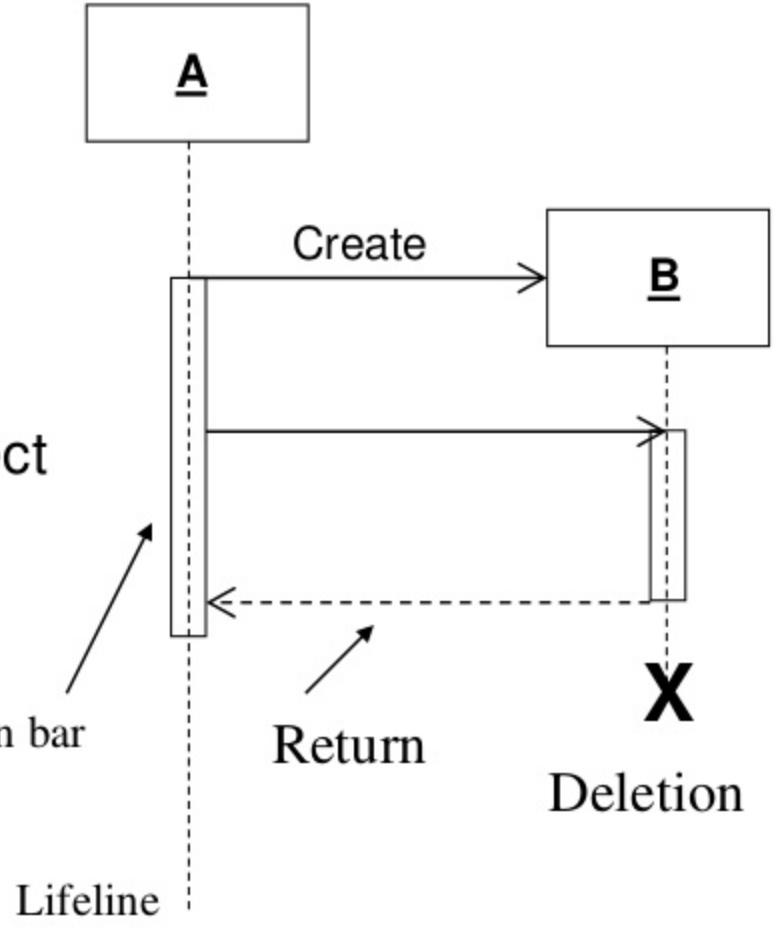
Making a phone call.

Sequence Diagram(Telephone call)



Sequence Diagrams – Object Life Spans

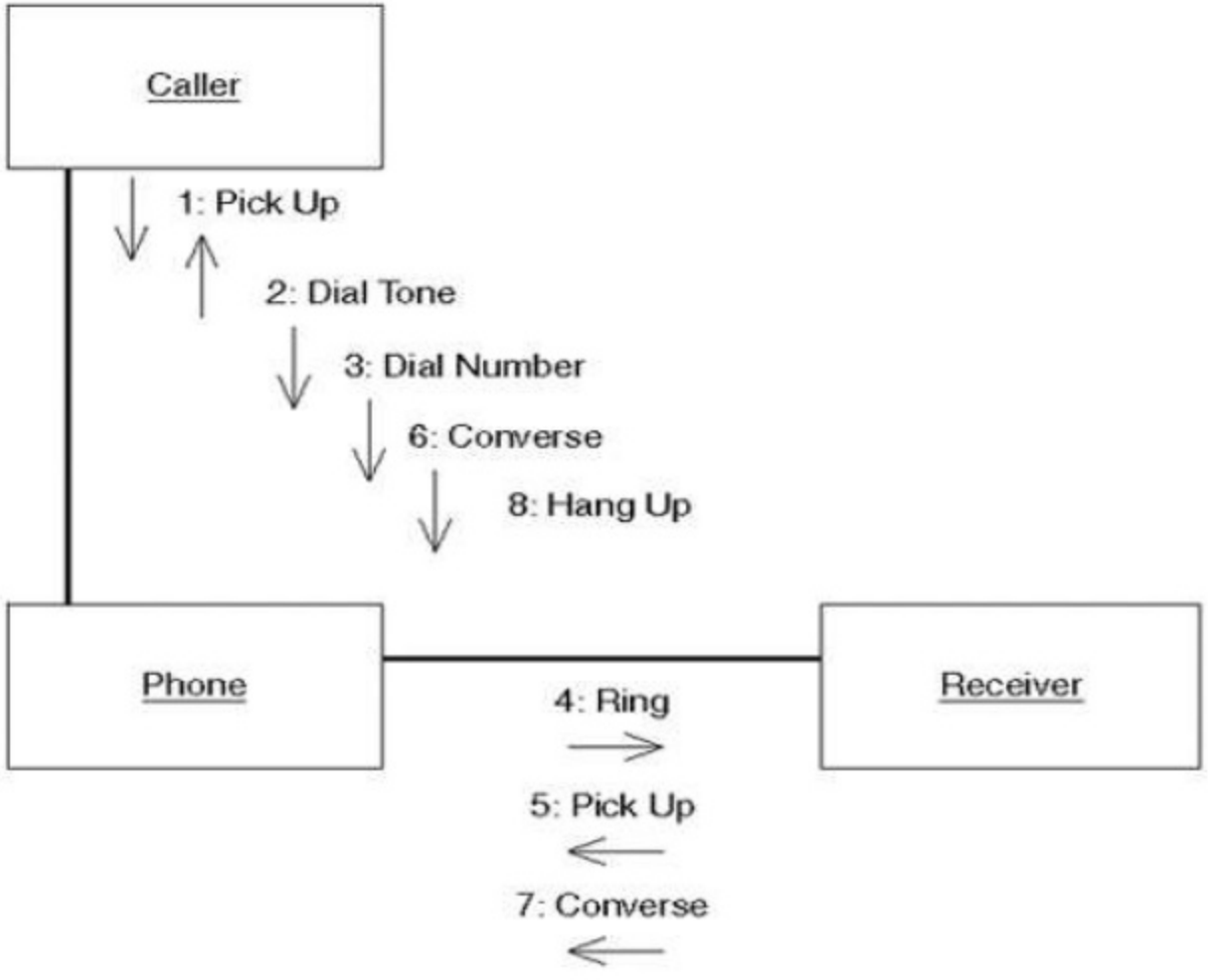
- Creation
  - Create message
  - Object life starts at that point
- Activation
  - Symbolized by rectangular stripes
  - Place on the lifeline where object is activated.
  - Rectangle also denotes when object is deactivated.
- Deletion
  - Placing an 'X' on lifeline
  - Object's life ends at that point



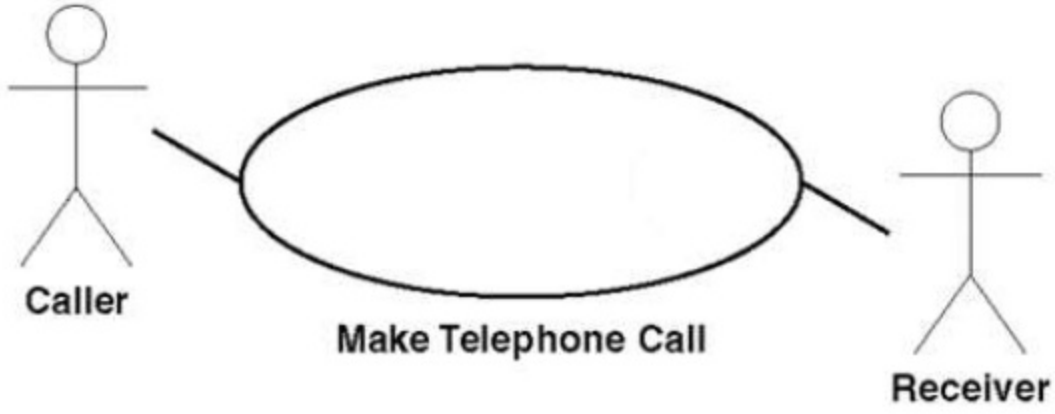
Collaboration Diagram

- A *collaboration diagram* also shows the passing of messages between objects, but focuses on the objects and messages and their order instead of the time sequence.
- The sequence of interactions and the concurrent threads are identified using sequence numbers.
- A collaboration diagram shows an interaction the relationships among the objects playing the different roles.
- The *UML Specification* suggests that collaboration diagrams are better for real-time specifications and for complex scenarios than sequence diagrams.

Collaboration Diagram(Telephone call)



Use Case diagram – Telephone Call



Activity Diagram

Activity Diagram

- *Activity Diagram* – a special kind of State chart diagram, but shows the flow from activity to activity

- *Activity state* –*non-atomic* execution, ultimately result in some action; a composite made up of other activity/action states; can be represented by other activity diagrams

- *Action state* –*atomic* execution, results in a change in state of the system or the return of a value (i.e., calling another operation, sending a signal, creating or destroying an object, or some computation); non-decomposable

continued…

- Activity diagram is basically a flow chart to represent the flow form one activity to another activity.
- The activity can be described as an operation of the system.
- So the control flow is drawn from one operation to another.
- This flow can be sequential, branched or concurrent.
- Activity diagrams deals with all type of flow control by using different elements like fork, join etc.
- The basic purposes of activity diagrams are similar to other four diagrams.

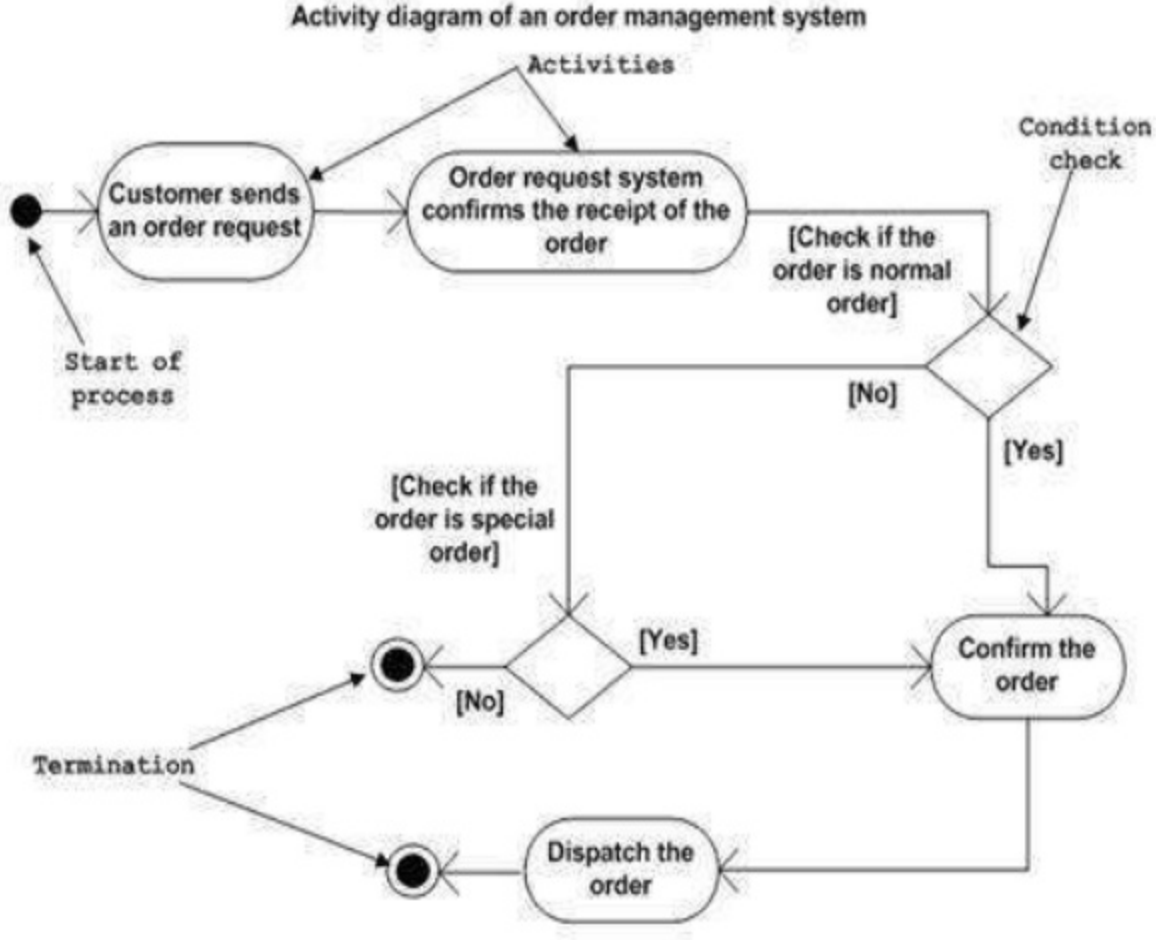
- It captures the dynamic behavior of the system.
- Other four diagrams are used to show the message flow from one object to another but activity diagram is used to show message flow from one activity to another.
- Activity diagrams are not only used for visualizing dynamic nature of a system but they are also used to construct the executable system by using forward and reverse engineering techniques.
- The only missing thing in activity diagram is the message part.

How to draw Activity Diagram?

- Activity diagrams are mainly used as a flow chart consists of activities performed by the system.
- So before drawing an activity diagram we should identify the following elements:
- Activities
- Association
- Conditions
- Constraints

Example : order management system

- Send order by the customer
- Receipt of the order
- Confirm order
- Dispatch order



Uses of Activity Diagrams

- Modeling work flow by using activities.
- Modeling business requirements.
- High level understanding of the system's functionalities.
- Investigate business requirements at a later stage.

Example : ATM System