Tribhuvan University
Institute of Engineering
Pulchowk Campus
Department of Electronics and Computer Engineering

# Software Engineering

## Chapter Two

## *System models*

by
Er. Bishwas Pokharel
Lecturer, Kathford Int'l College of Engg. and Mgmt.

Date: 10th jan, 2018

# Chapter Two: System models

Course Outline: **3 hours, 5 Marks**

**2. System models (3 hours)**

2.1. Context models

2.2. Behavioral models

2.3. Data and object models

Case Tools

# What is use case ?

- A requirements analysis concept
- Describes the system's actions from a the point of view of a user.
- **Tells a story**
  - A sequence of events involving.
  - Interactions of a user with the system .
  - Specifies one aspect of the behavior of a system, **without specifying the structure of the system**
- Is oriented toward satisfying a user's goal.

# How do we describe use cases?

- Textual or tabular description
- User stories
- Diagram( we focused on this !!)

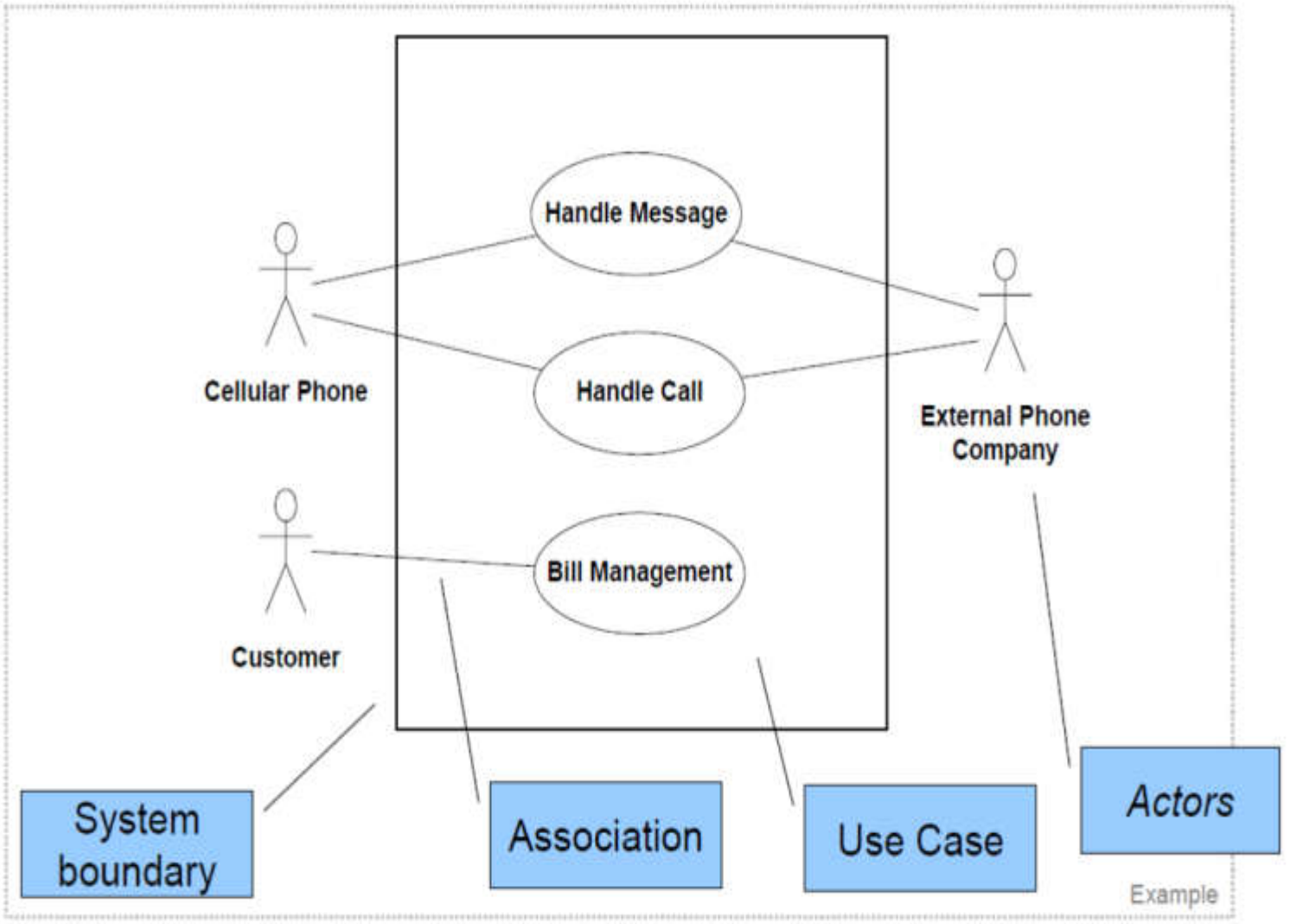
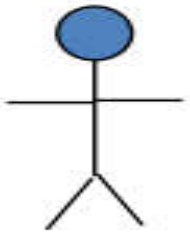



**Fig : Example of use case diagram**

# Elements of use case diagram:

## a. Actor:

- Actor is someone interacting with use case (system function). Named by noun.
- Actor has responsibility toward the system (inputs), and Actor have expectations from the system (outputs)
- Actor triggers use case.



# Finding Actors

**Ask the following questions:**

- Who are the system's primary users?
- Who requires system support for daily tasks?
- Who are the system's secondary users?
- What hardware does the system handle?
- Which other (if any) systems interact with the system?
- Which other entities (human or otherwise) might have an interest in the system's output?

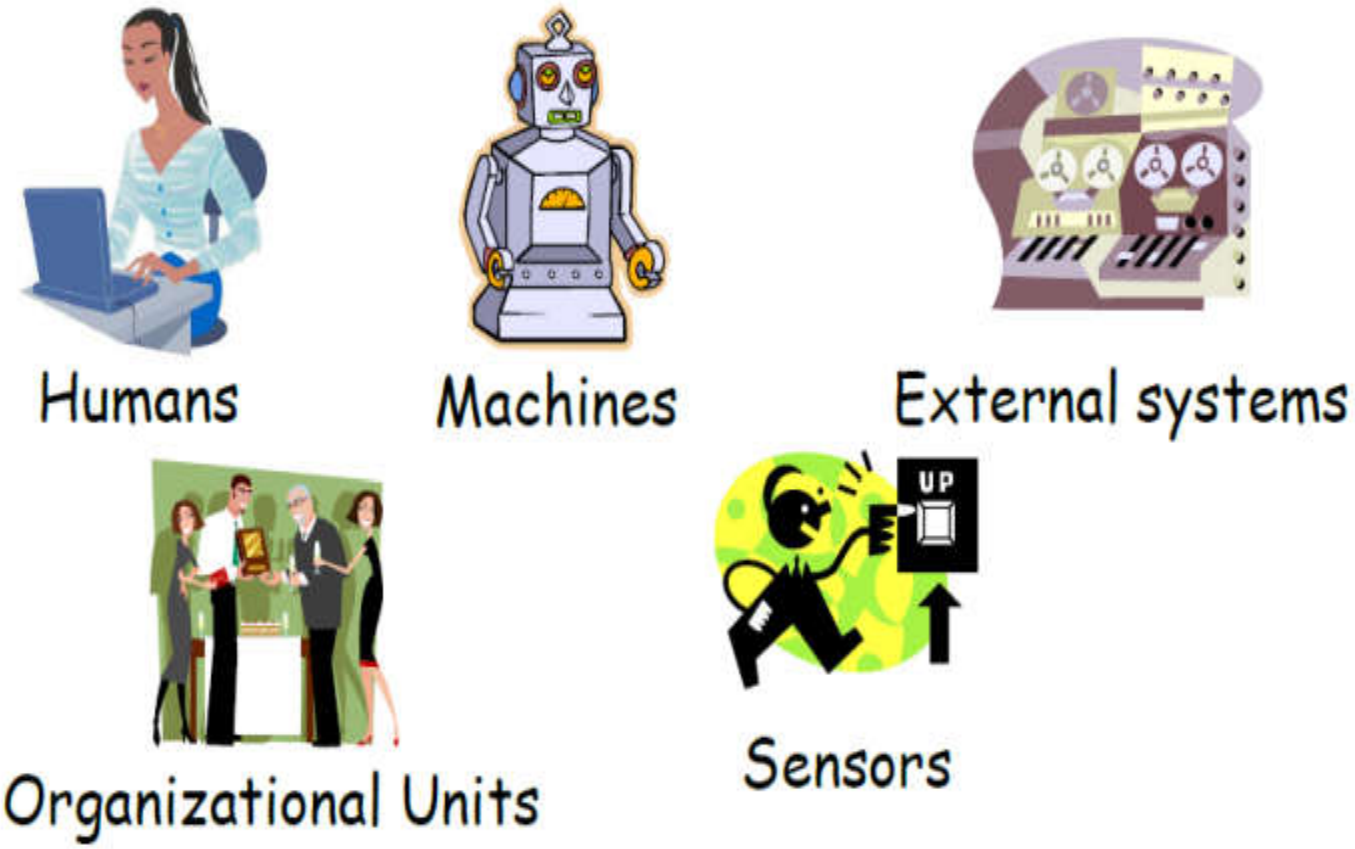



**Fig: Some examples of actors**

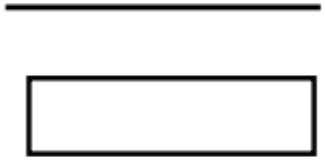
## b. Use case

- System function (process–automated or manual). Named by verb.

- Each Actor must be linked to a use case, while some use cases may not be linked to actors.

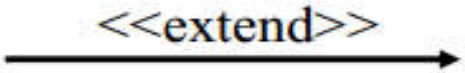
| USER/ACTOR | USER GOAL = Use Case |
|---|---|
| Order clerk | Look up item availability<br>Create new order<br>Update order |
| Shipping clerk | Record order fulfillment<br>Record back order |
| Merchandising manager | Create special promotion<br>Produce catalog activity report |

## c. Other details



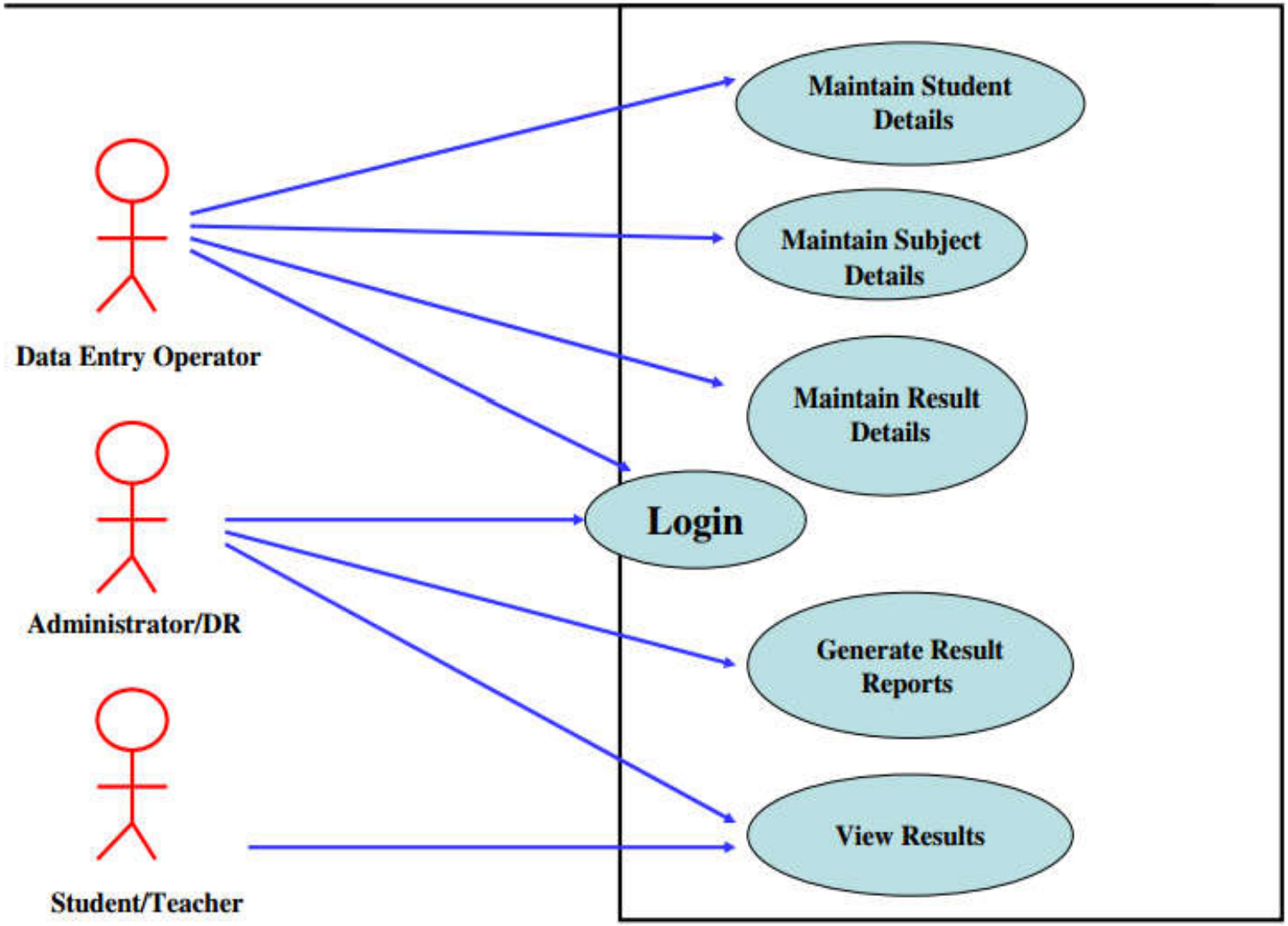
Connection between Actor and Use Case Boundary of system

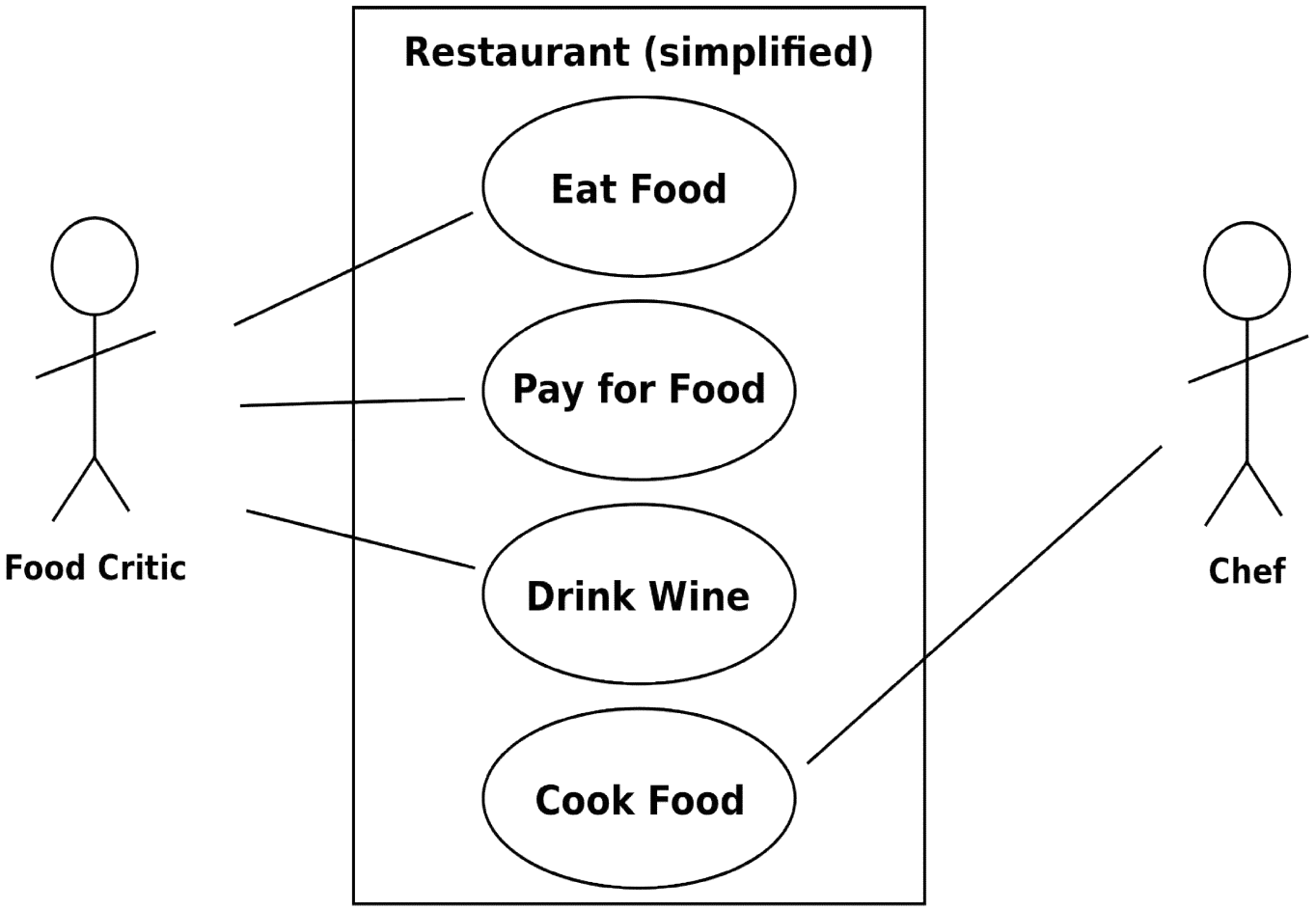

**Include** relationship between Use Cases (one UC must call another; e.g., Login UC includes User Authentication UC)

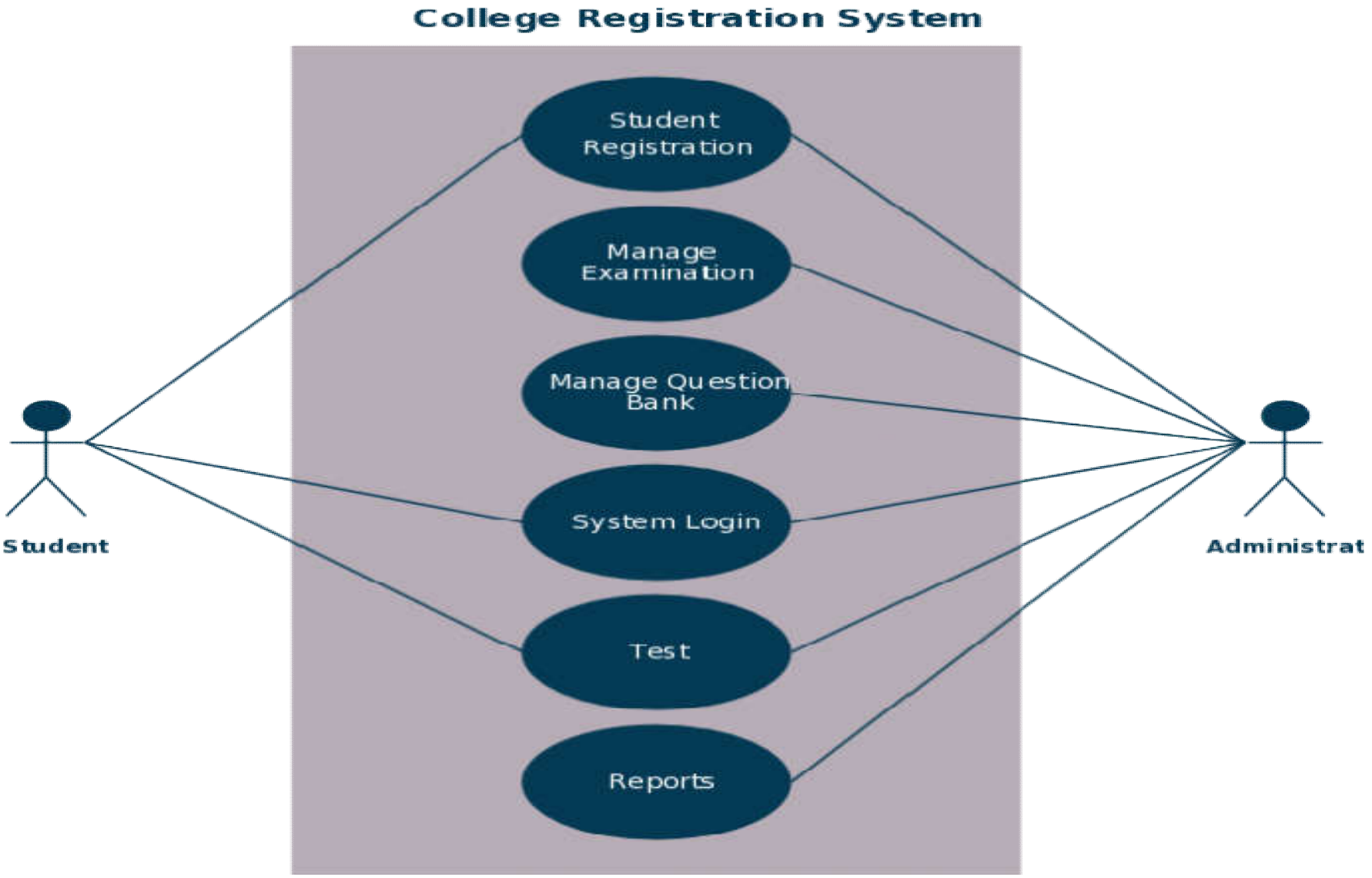



**Extend** relationship between Use Cases (one UC calls Another under certain condition; think of if-then decision points)

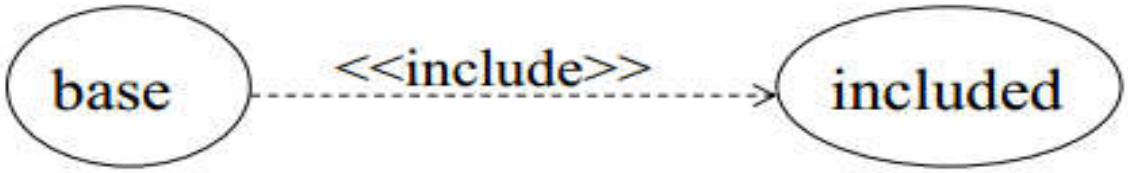
# Use case diagram for Result Management System




Restaurant (simplified)
Eat Food
Pay for Food
Drink Wine
Cook Food
Food Critic
Chef

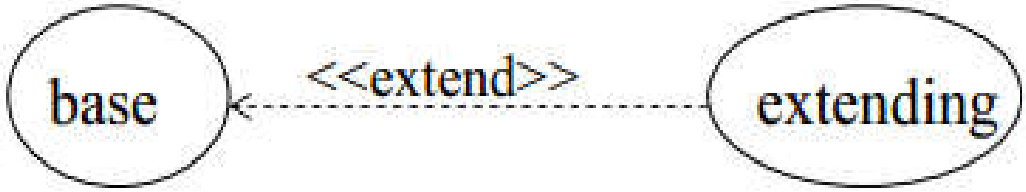

College Registration System
Student Registration
Manage Examination
Manage Question Bank
System Login
Test
Reports
Student
Administrat

# Include



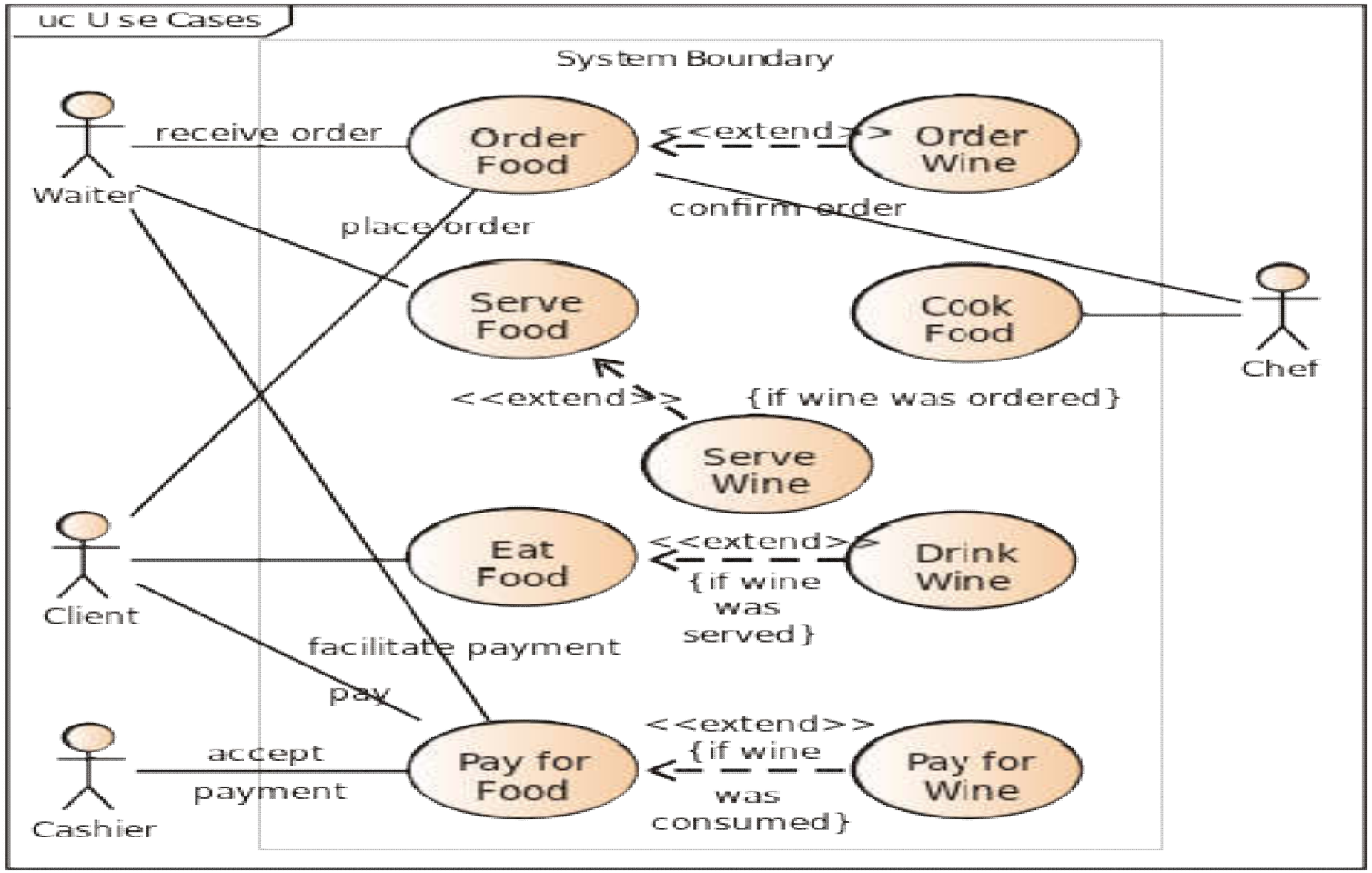

The included use case never stands alone. It only occurs as a part of some larger base that includes it.

# Extend




The base use case may stand alone, but under certain conditions its behavior may be extended by the behavior of another use case.

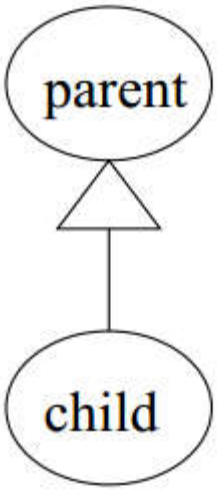
# Include and Extend

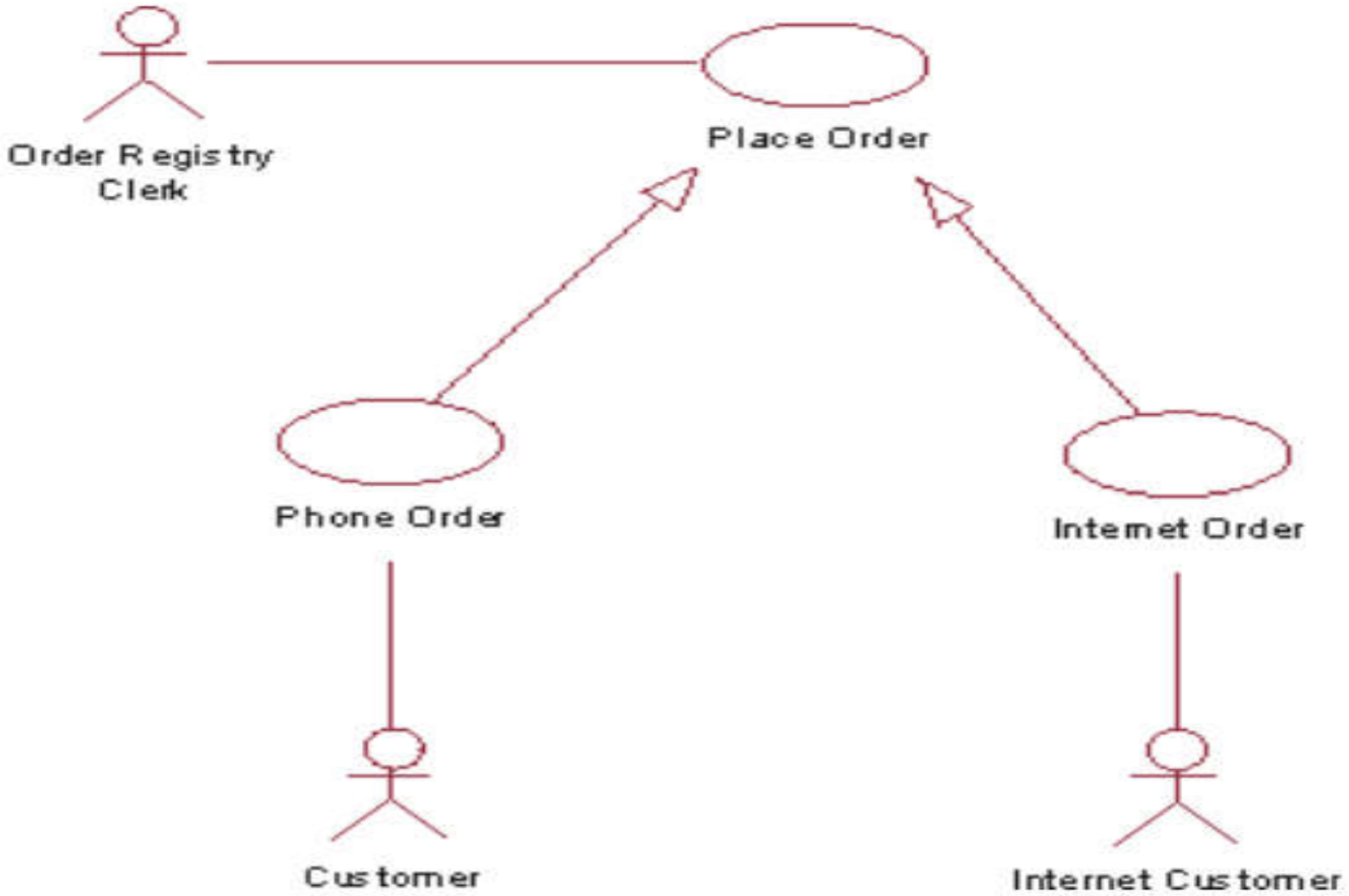


# Generalization

The child use case inherits the behavior and meaning of the parent use case.

The child may add to or override the behavior of its parent

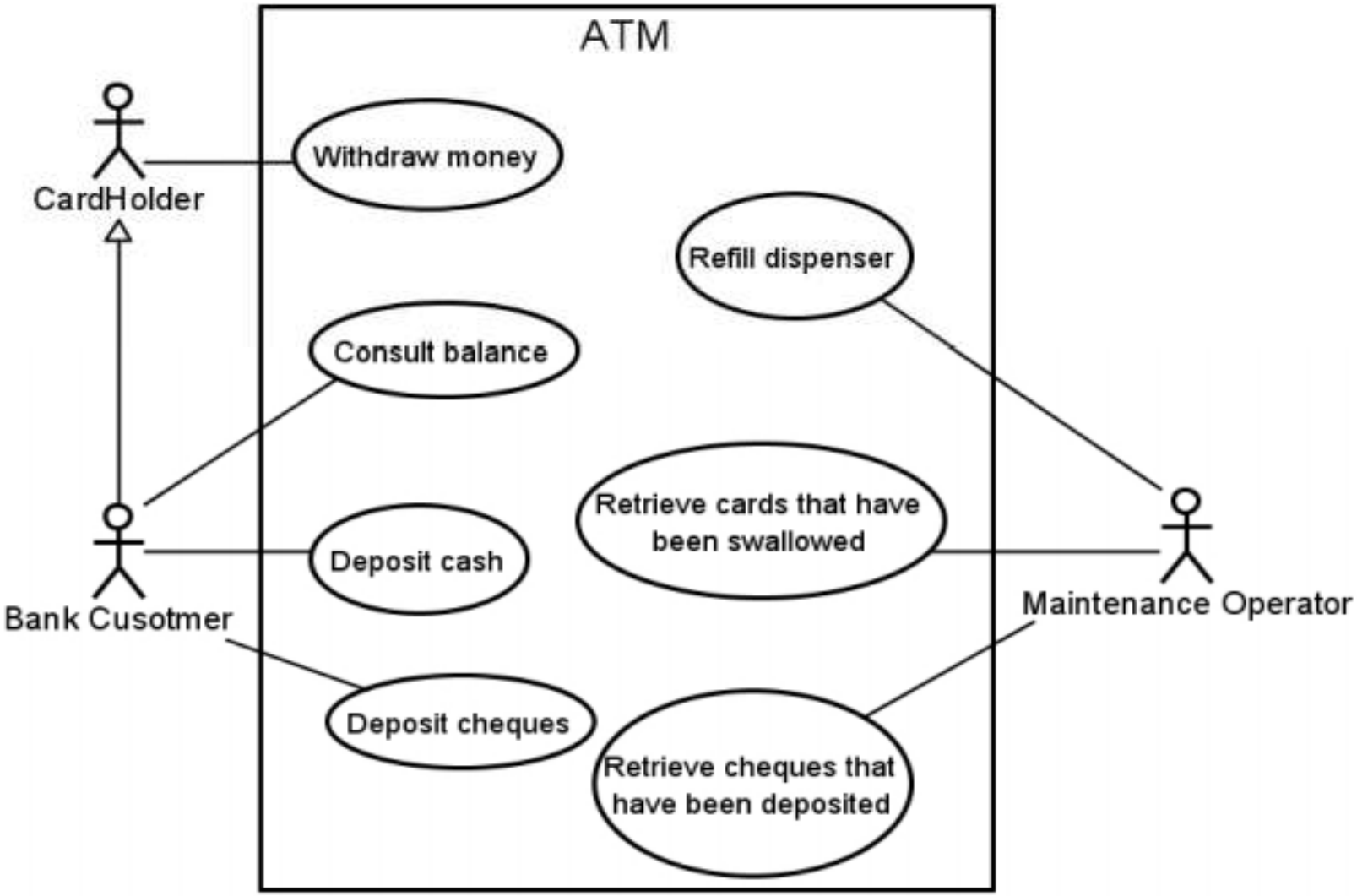



Order Registry Clerk
Place Order
Phone Order
Internet Order
Customer
Internet Customer

**Create use case diagram for the ATM machine system. ( consider all possible actor and cases) !!!!**

**Answer may differ from one other based on the scenario**

# An ATM System



# Structural modeling

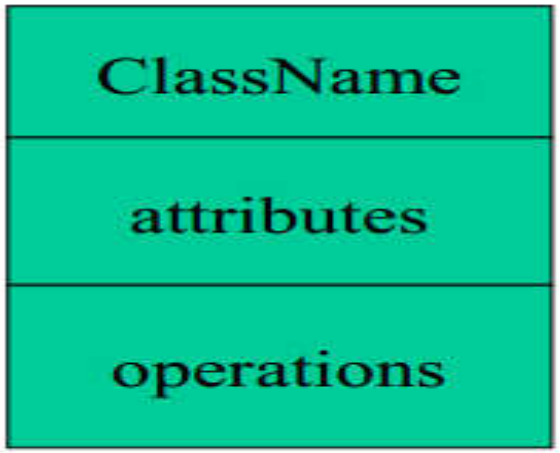
# What is Class diagram?

A ***class*** is a description of a set of objects that share the same attributes, operations, relationships, and semantics.

It consists the :

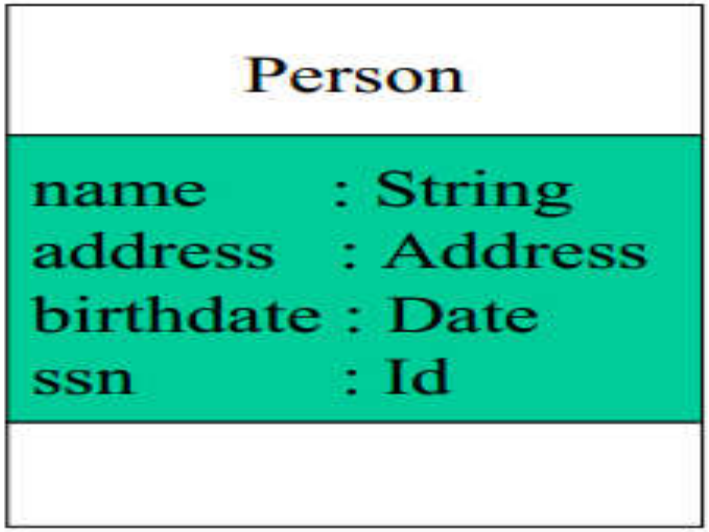
- Classes
- Attributes
- Operations (or methods),
- Relationships among the classes.

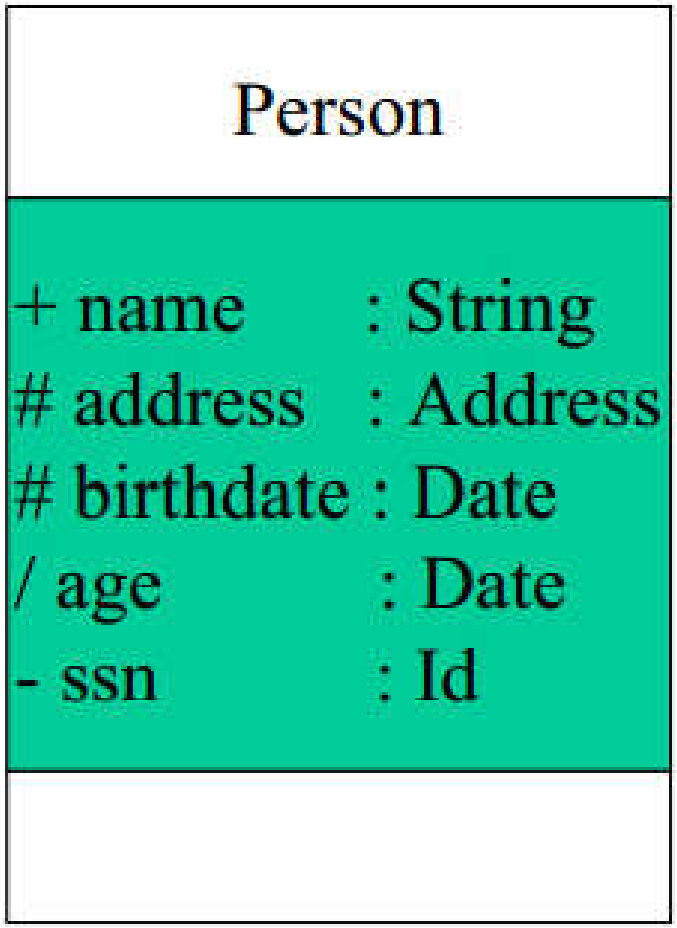
Graphically, a class is rendered as a rectangle, usually including its name, attributes, and operations in separate, designated compartment.



The ***name of the class*** is the only required tag in the graphical representation of a class. It always appears in the top-most compartment.

An ***attribute*** is a named property of a class that describes the object being modeled. In the class diagram, attributes appear in the second compartment just below the name-compartment.






Attributes can be:

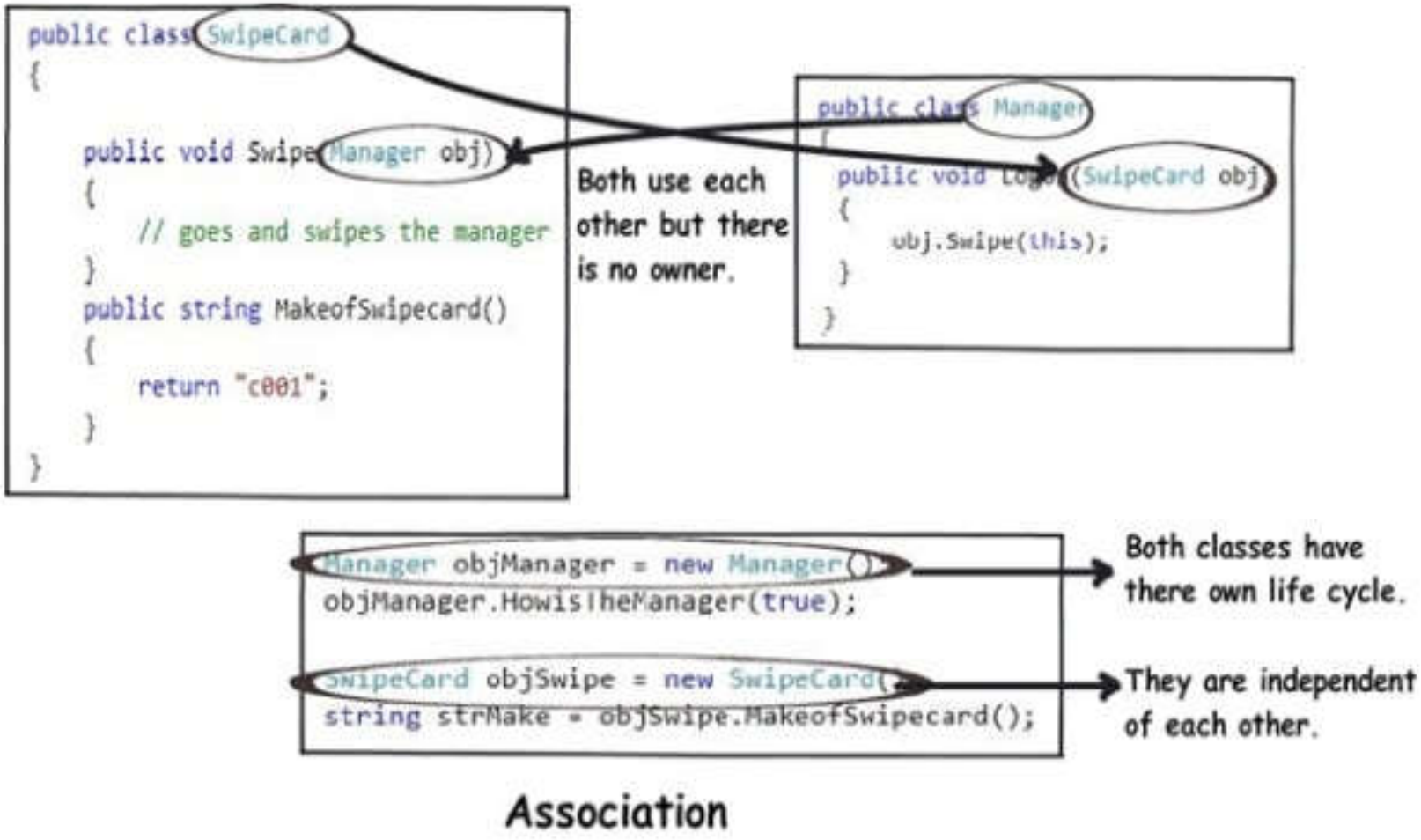
+ public
# protected
- private
/ derived

# Relationship

## ASSOCIATION

These are the most general type of relationship:

- It shows BI directional connection between two classes
- It is a weak coupling as associated classes remain somewhat independent of each other
- Example

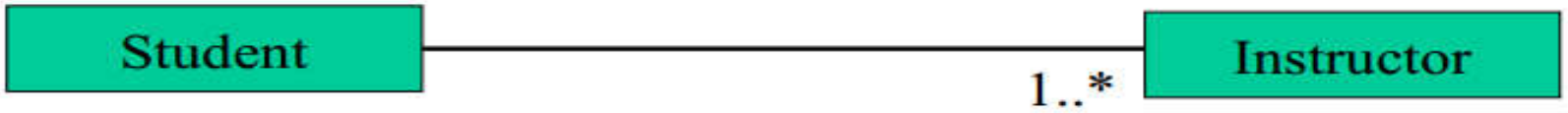



The above diagram shows how the SwipeCard class uses the Manager class and the Manager class uses the SwipeCard class. You can also see how we can create objects of the Manager class and SwipeCard class independently and they can have their own object life time.

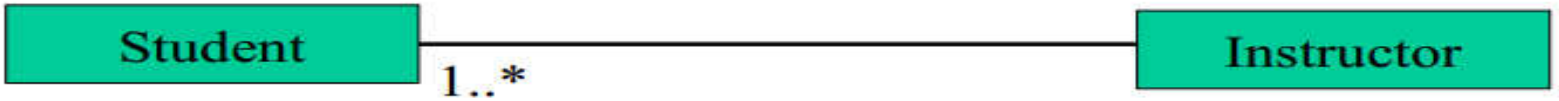
This relationship is called the "Association" relationship.

We can indicate the **multiplicity** of an association by adding multiplicity adornments to the line denoting the association.

The example indicates that a *Student* has one or more *Instructors*:

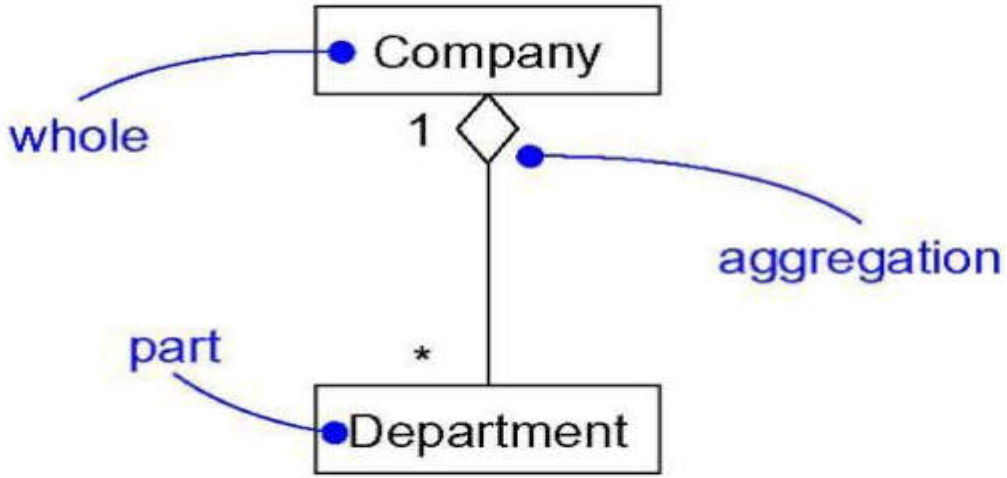



The example indicates that every *Instructor* has one or more *Students*:



# AGGREGATION

- The association with label “contains” or “is part of” is an aggregation >> represents “has a“ relationship
- It is used when one object contains other.
- Container is called as aggregate with diamond at end.
- It expresses a whole - part relationships, like



**Aggregation** is a specialized form of Association where all objects have their own lifecycle, but there is ownership and a child object cannot belong to another parent object. Let's take an example of a Department and teacher. we delete the department the teacher object will not be destroyed. We can think of it as a "has-a" relationship.

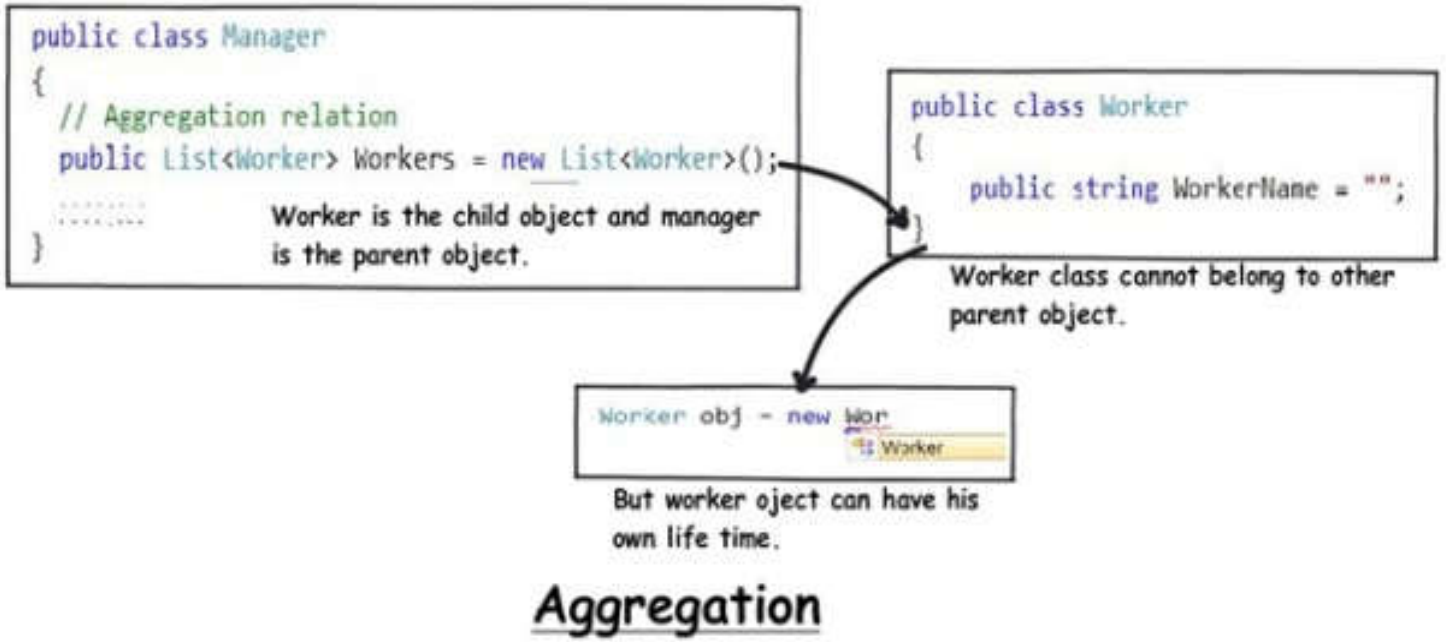
For example, the class “library” is made up of one or more books, among other materials. In aggregation, the contained classes are not strongly dependent on the lifecycle of the container. In the same example, books will remain so even when the library is dissolved.

The third requirement from our list (Manager has workers who work under him) denotes the same type of relationship like association but with a difference that one of them is an owner. So as per the requirement, the `Manager` object will own `Worker` objects.

The child `Worker` objects can not belong to any other object. For instance, a `Worker` object cannot belong to a `SwipeCard` object.

But... the `Worker` object can have its own life time which is completely disconnected from the `Manager` object. Looking from a different perspective, it means that if the `Manager` object is deleted, the `Worker` object does not die.

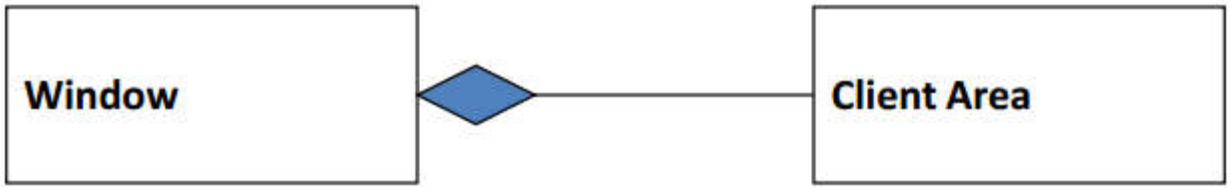
This relationship is termed as an "Aggregation" relationship.



# COMPOSITION

This is a strong form of aggregation

- It expresses the stronger coupling between the classes
- The owner is explicitly responsible for creation and deletion of the part
- Any deletion of whole is considered to cascade its part
- The aggregate has a filled diamond at its end



- The composition relationship is very similar to the aggregation relationship. with the only difference being its key purpose of emphasizing the dependence of the contained class to the life cycle of the container class. That is, the contained class will be obliterated when the container class is destroyed. For example, a shoulder bag's side pocket will also cease to exist once the shoulder bag is destroyed.

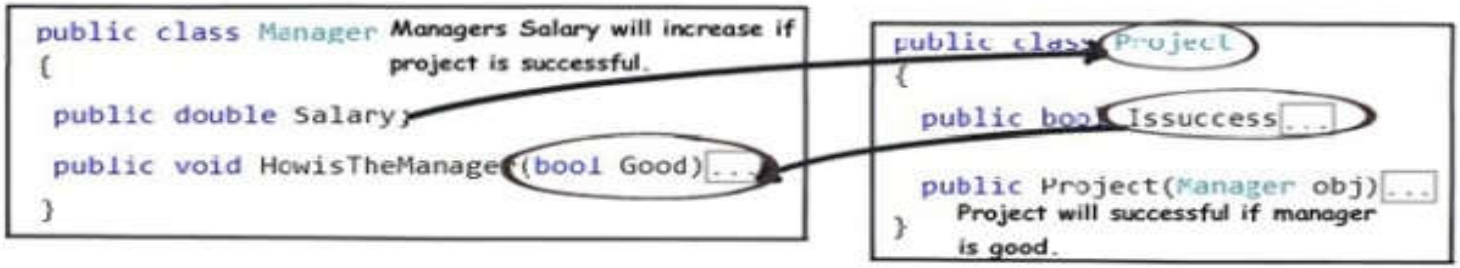
The last two requirements are actually logically one. If you read closely, the requirements are as follows:

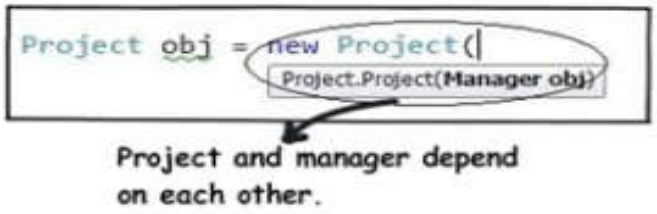
1. Manager has the responsibility of ensuring that the project is successful.
2. Manager's salary will be judged based on project success.

Below is the conclusion from analyzing the above requirements:

1. Manager and the project objects are dependent on each other.
2. The lifetimes of both the objects are the same. In other words, the project will not be successful if the manager is not good, and the manager will not get good increments if the project has issues.

Below is how the class formation will look like. You can also see that when I go to create the project object, it needs the manager object.

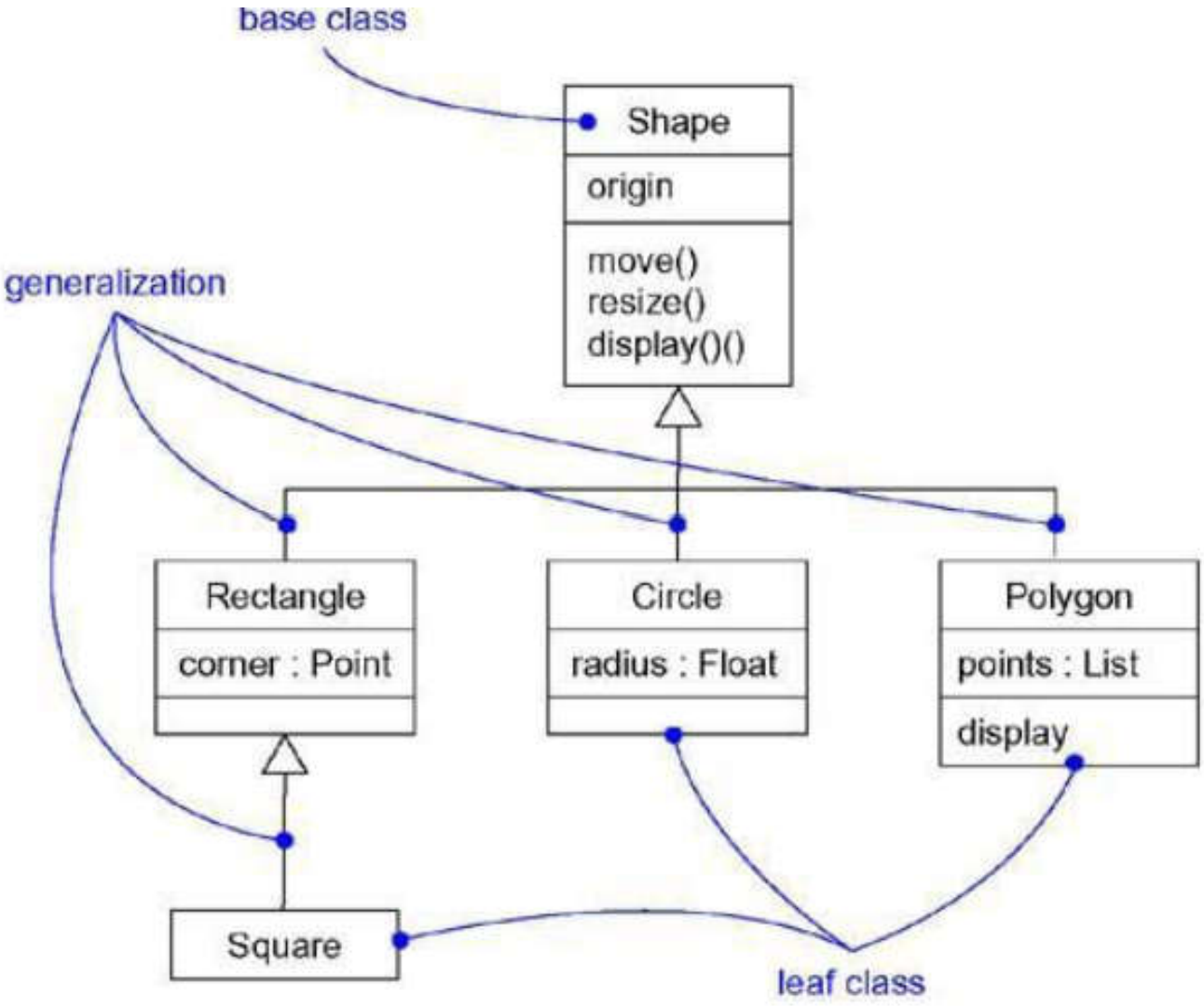



This relationship is termed as the composition relationship. In this relationship, both objects are heavily dependent on each other. In other words, if one goes for garbage collection the other also has to be garbage collected, or putting from a different perspective, the lifetime of the objects are the same. That's why I have put in the heading "Death" relationship.

# INHERITANCE/GENERALIZATION

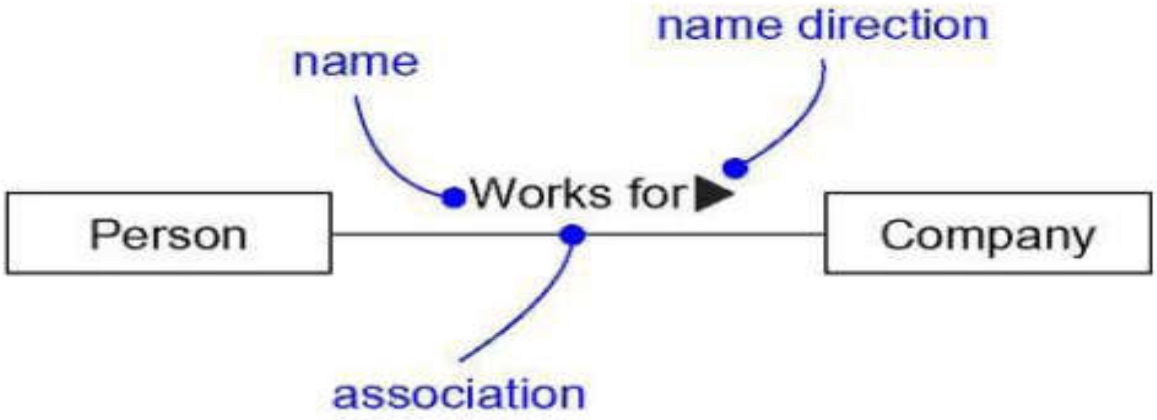
- The inheritance relationship helps in managing the complexity by ordering objects within trees of classes with increasing levels of abstraction. Notation used is solid line with arrowhead, shown below.
- Generalization and specialization are points of view that are based on inheritance hierarchies..

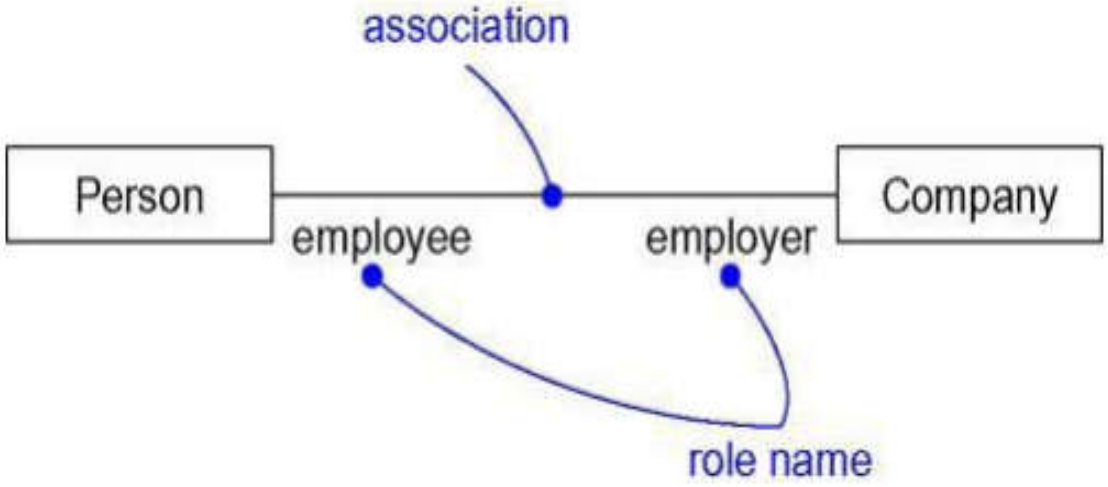

base class
Shape
origin
move()
resize()
display()()
generalization
Rectangle
corner : Point
Circle
radius : Float
Polygon
points : List
display
Square
leaf class

## Dependency

- A dependency relationship indicates that changes to one model element (the supplier or independent model element) can cause changes in another model element (the client or dependent model element). The supplier model element is independent because a change in the client does not affect it. The client model element depends on the supplier because a change to the supplier affects the client.

- This association is unidirectional and is shown with dotted arrowhead line.
- In the following example it shows the dependency relationship between client and server.
- The client avails services provided by server so it should have semantic knowledge of server.


name direction
name
Works for
Person
Company
association

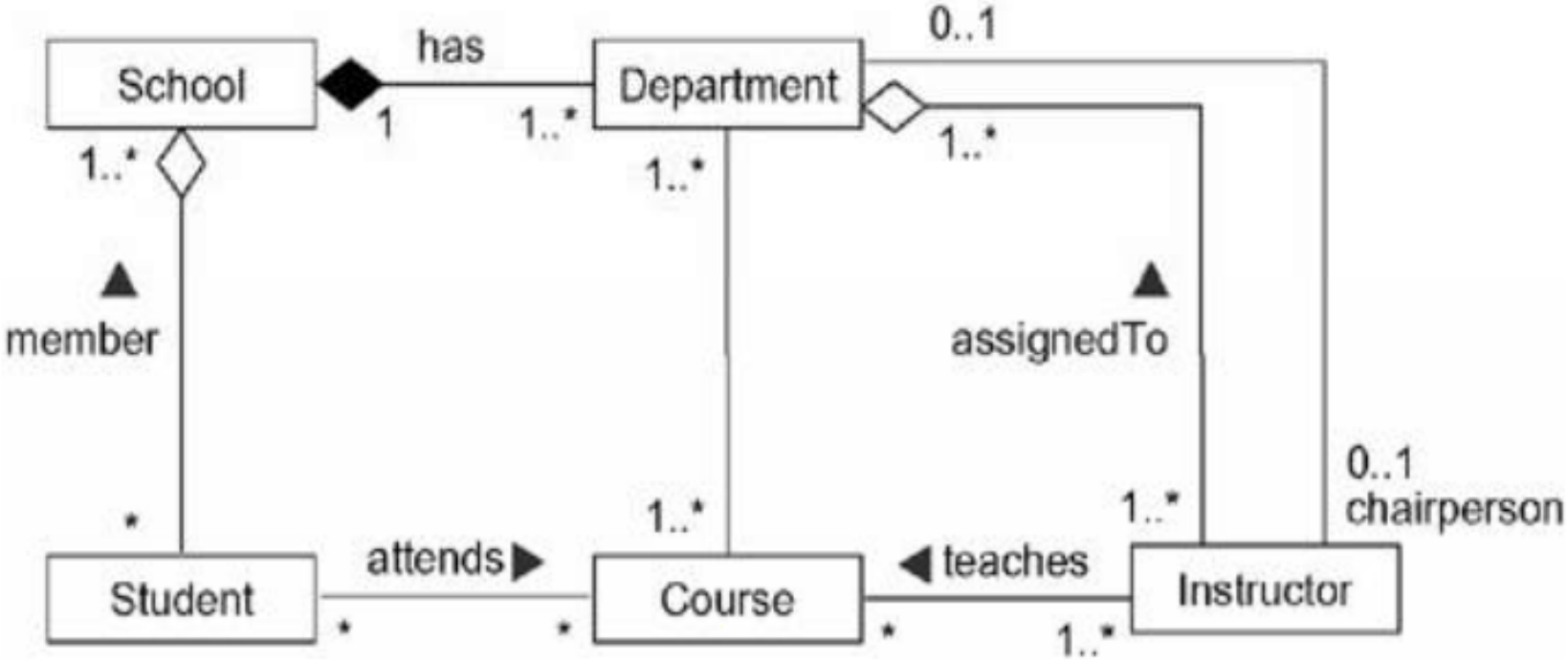

association
Person
Company
employee
employer
role name

**Definition:** Number of instances of each class involved in the dialogue is specified by cardinality.

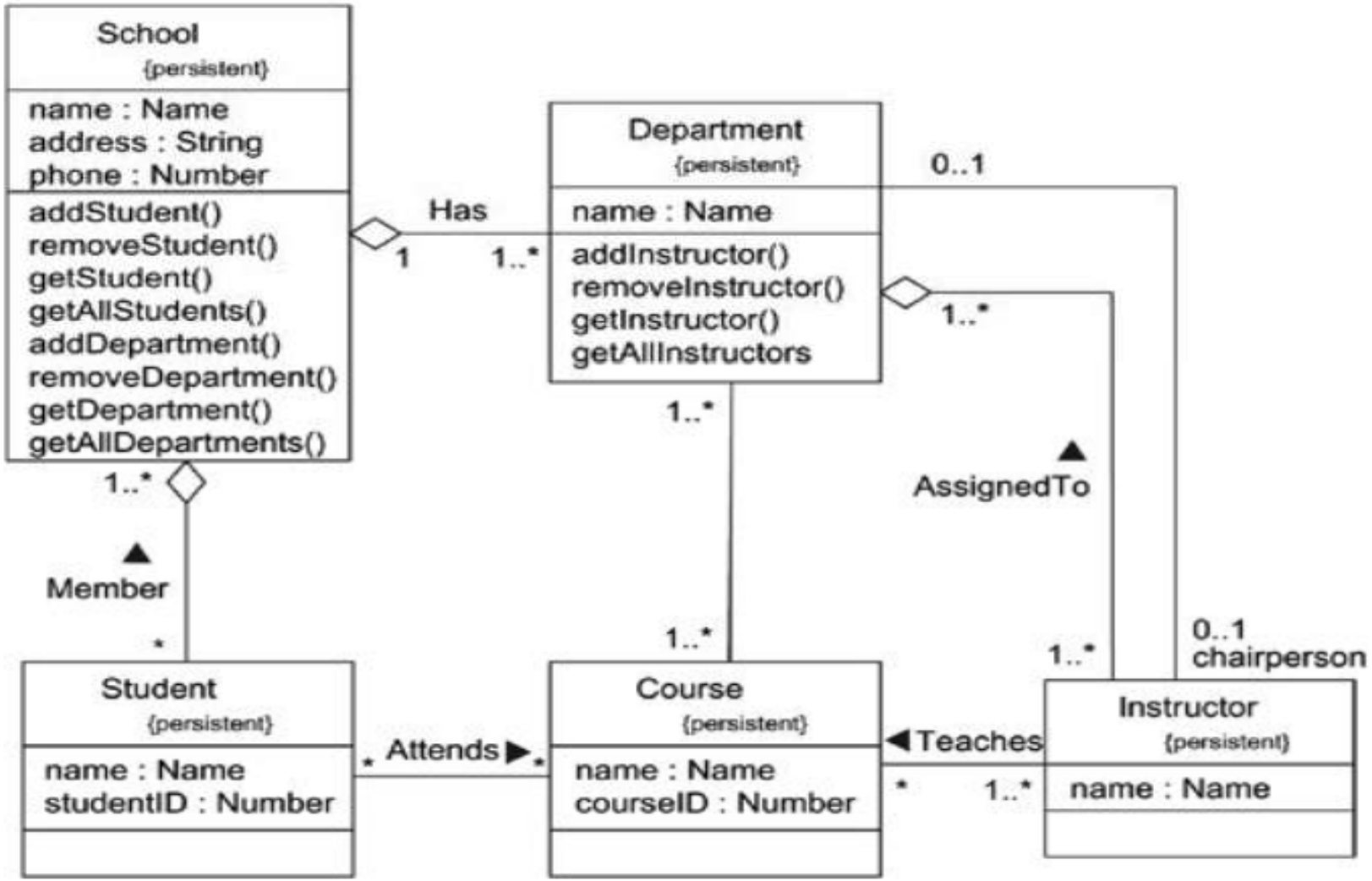
- **Common multiplicity values:**

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

# Structural Relationships



# Modeling a Schema



# ER Diagram

ER-Diagram is a visual representation of data that describes how data is related to each other

**Components of E-R Diagram**

The E-R diagram has three main components.

**1) Entity**

**2)Attributes**

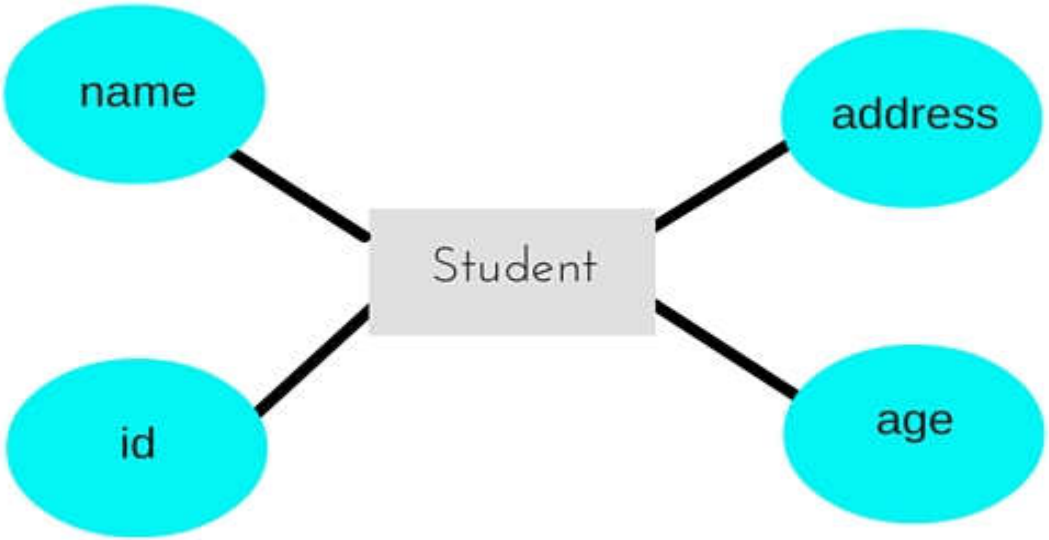
**3)Relationship**

## 1) Entity

An **Entity** can be any object, place, person or class. In E-R Diagram, an **entity** is represented using rectangles. Consider an example of an Organisation. Employee,Manager,Department, Product and many more can be taken as entities from an Organisation.

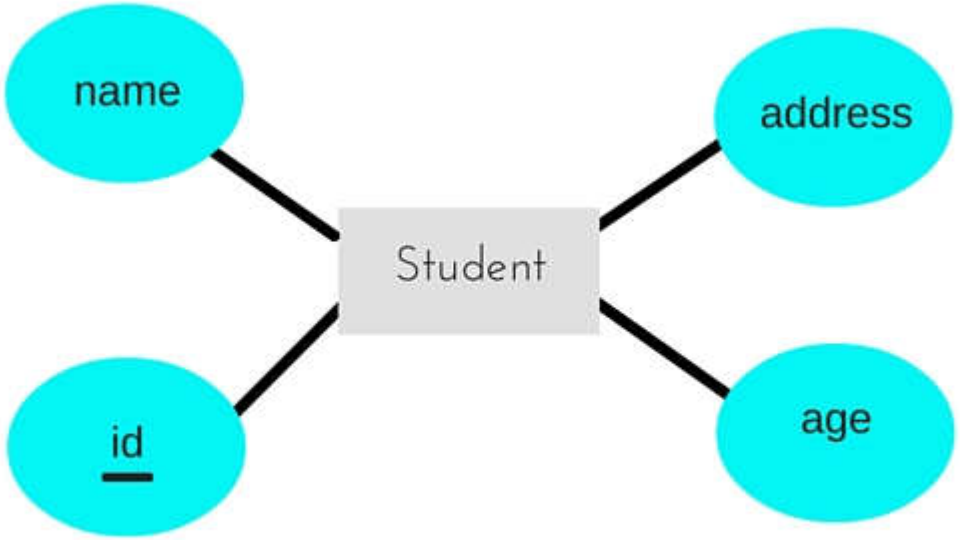
## 2) Attribute

An **Attribute** describes a property or characteristic of an entity. For example Name, Age, Address etc can be attributes of a Student. An attribute is represented using eclipse.



## i) Key Attribute

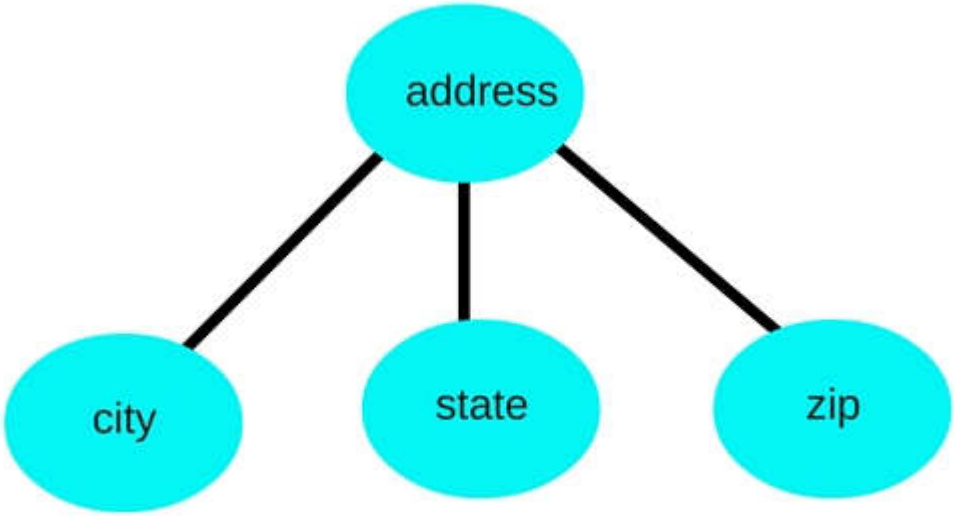
Key attribute represents the main characterstic of an Entity. It is used to represent Primary key. Ellipse with underlying lines represent Key Attribute.



## ii)Composite Attribute

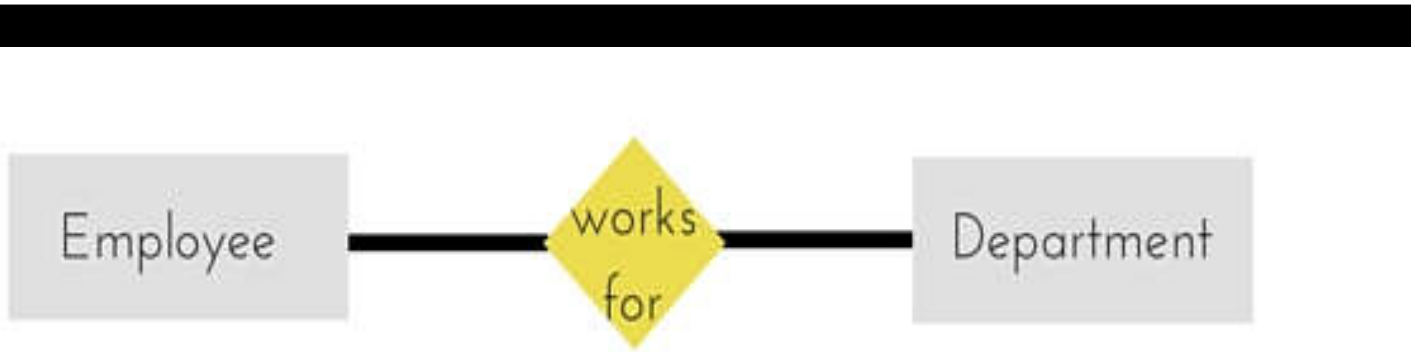
An attribute can also have their own attributes.

These attributes are known as **Composite** attribute.



## 3) Relationship

A Relationship describes relations between **entities**.
Relationship is represented using diamonds



## 3) Relationship

A Relationship describes relations between **entities**.
Relationship is represented using diamonds


A. One to One

B. One to Many

C. Many to One

D. Many to Many

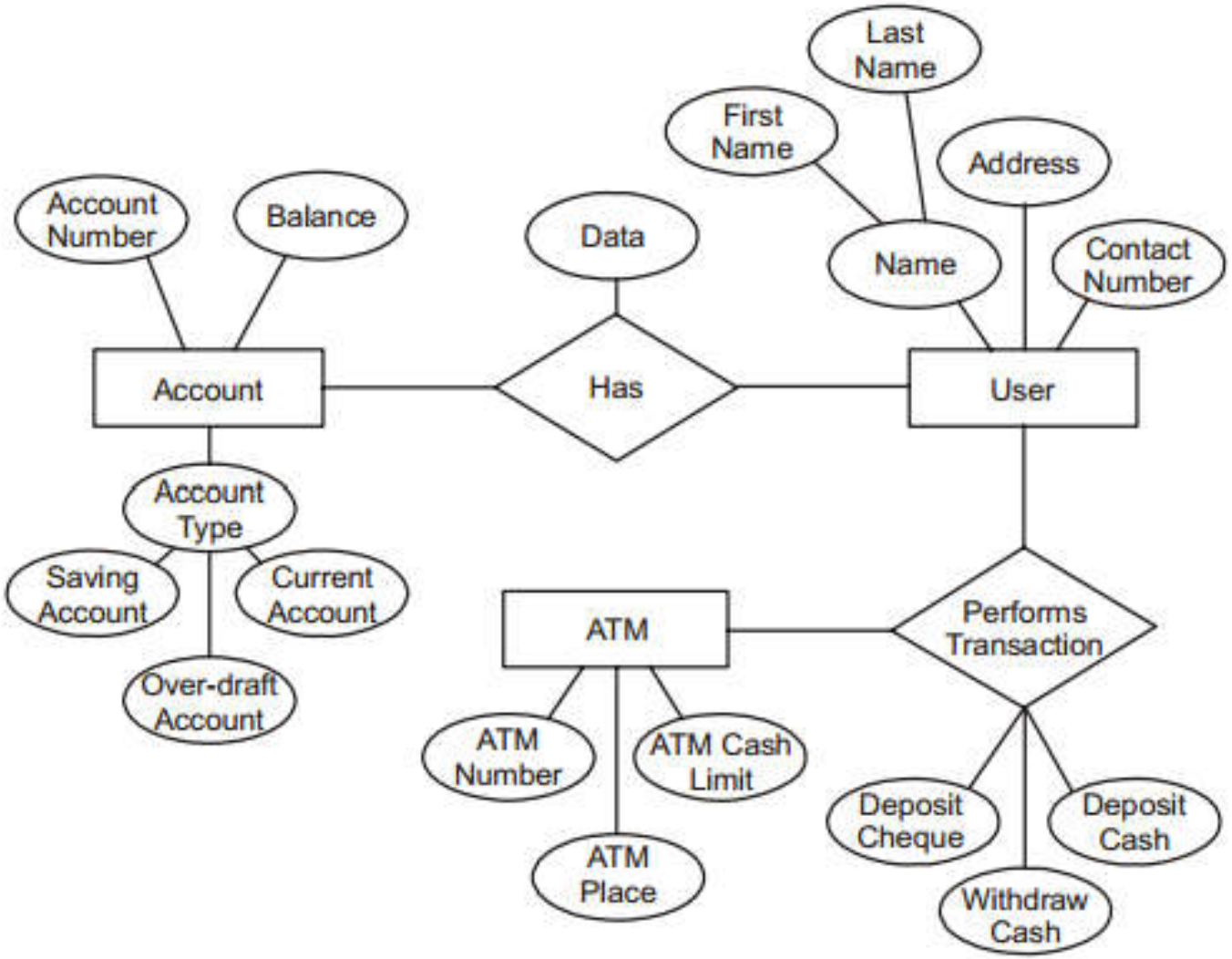



Fig : ER Of Banking System

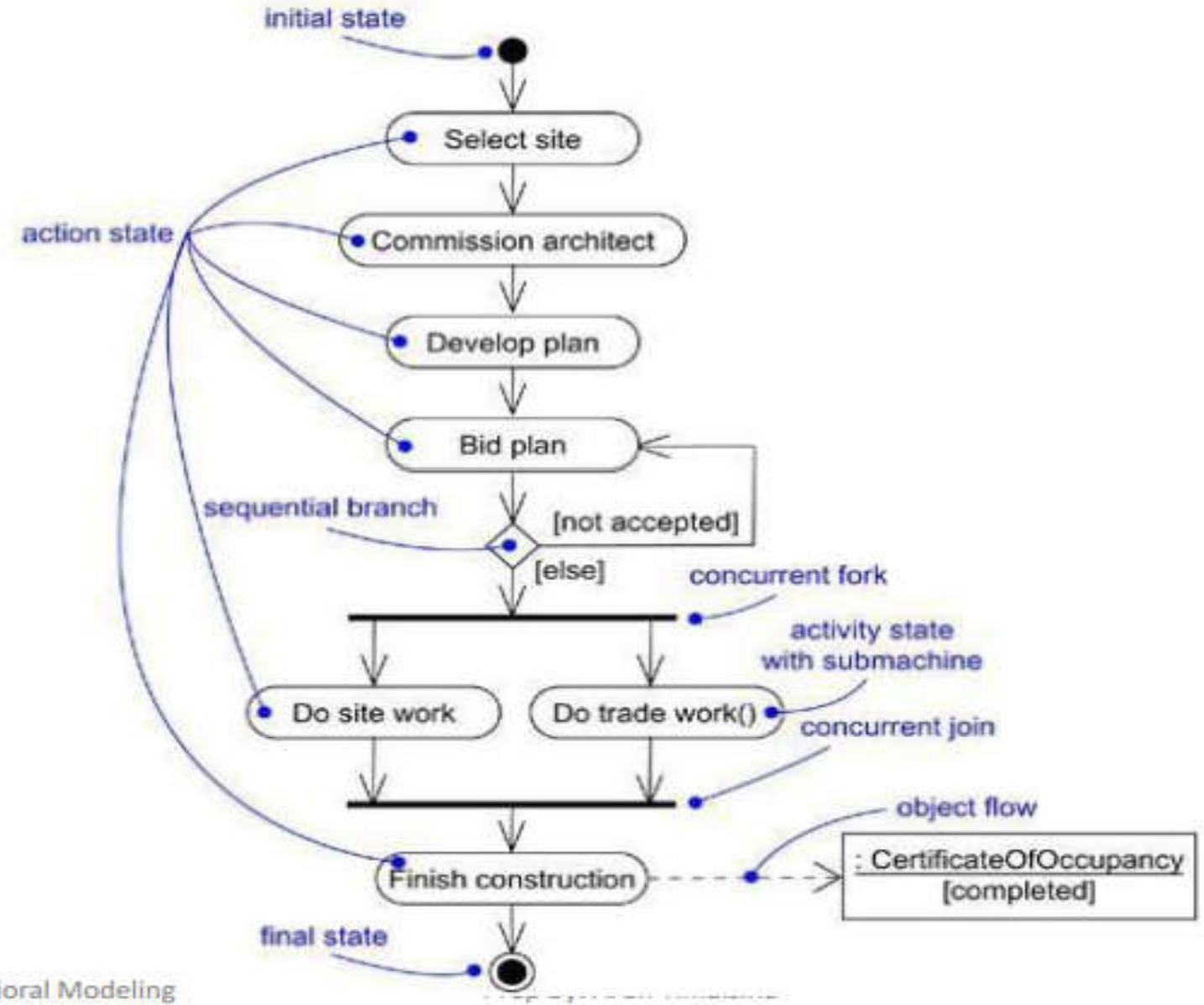
# Behavior Modeling

# What is Activity diagram ?

- It is essentially a flowchart, showing flow of control from activity to activity.
- It models the sequential and concurrent steps in computational process.
- Model the flow of an object as it moves from state to state at different points in the flow of control.

Consider the workflow associated with building a house. First, you select a site. Next, you commission an architect to design your house. After you've settled on the plan, your developer asks for bids to price the house. Once you agree on a price and a plan, construction can begin. Permits are secured, ground is broken, the foundation is poured, the framing is erected, and so on, until everything is done. You're then handed the keys and a certificate of occupancy, and you take possession of the house.

**Fig: Building a house**


initial state
Select site
action state
Commission architect
Develop plan
Bid plan
sequential branch
[not accepted]
[else]
concurrent fork
activity state
with submachine
Do site work
Do trade work()
concurrent join
object flow
Finish construction
: CertificateOfOccupancy
[completed]
final state

## Components of activity diagram are:

a) **Activities or action state:** notation used for activity/action is rectangle with rounded corner and it indicate action.

b) **State**: activity diagram can have only one start state but can have several end/stop state. UML may describe two special state called start state and stop state represented as: black dot and black dot with round circle.

c)**Transition:** used to show control flow from one state to another state. It shows flow from state to an activity, between activity or between states.

d) **Branching:**

- Branch specifies alternate paths taken based on some Boolean expression
- Branch is represented as a diamond
- May have one incoming transition and two or more outgoing transitions.

e)**Forking and joining:** used to show control flow from one state to another state. It shows flow from state to an activity, between activity or between states.

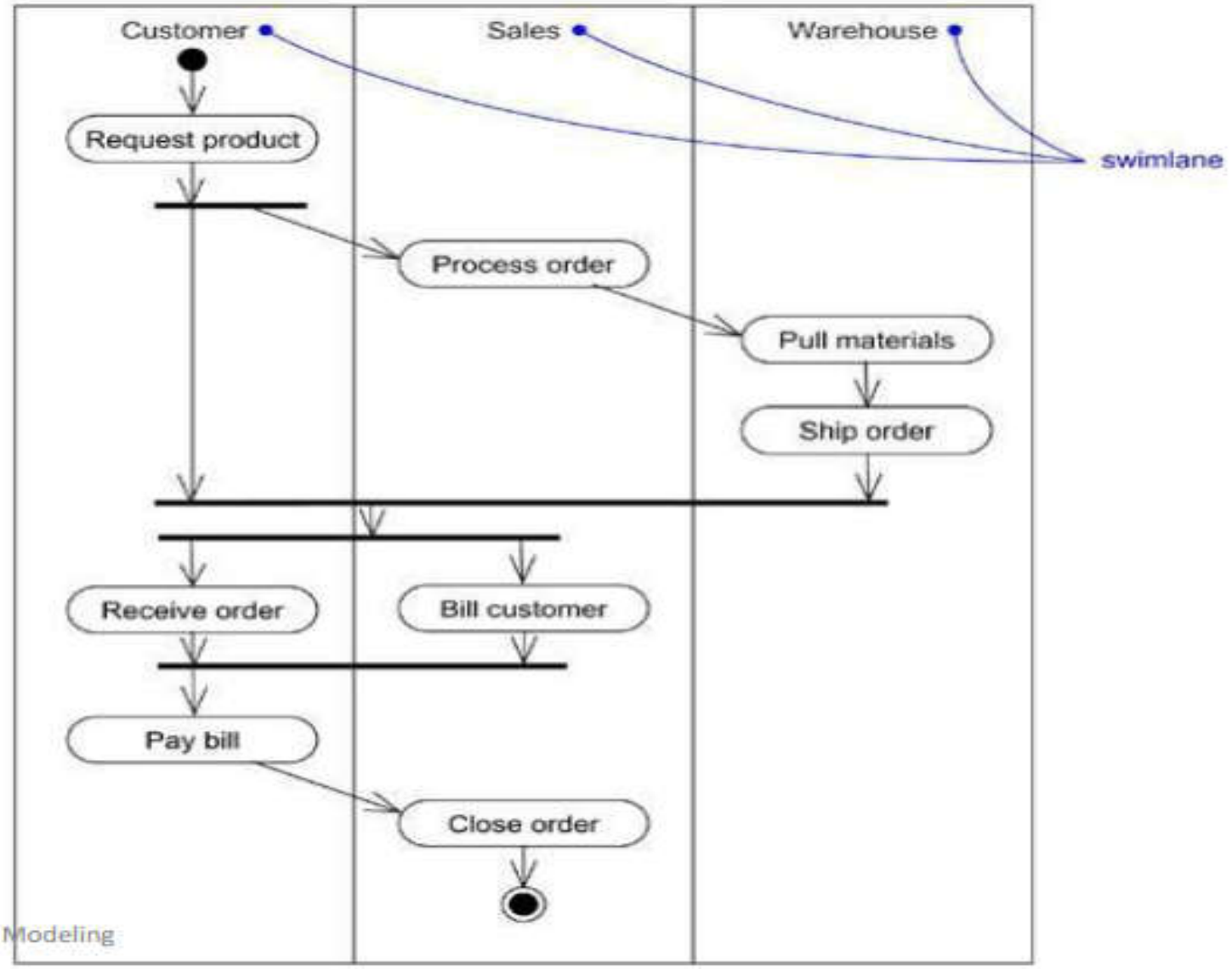
- Forking and Joining are used while modeling the business process workflows and that encounter the concurrent flows.
- A synchronization bar is rendered as a thick horizontal or vertical line
- Below the fork, the activities associated with each of these paths continues in parallel

## f) **Swimlanes**

- *Swimlanes* are useful in partitioning various activities into groups, where each group represents the business organization responsible for those activities.

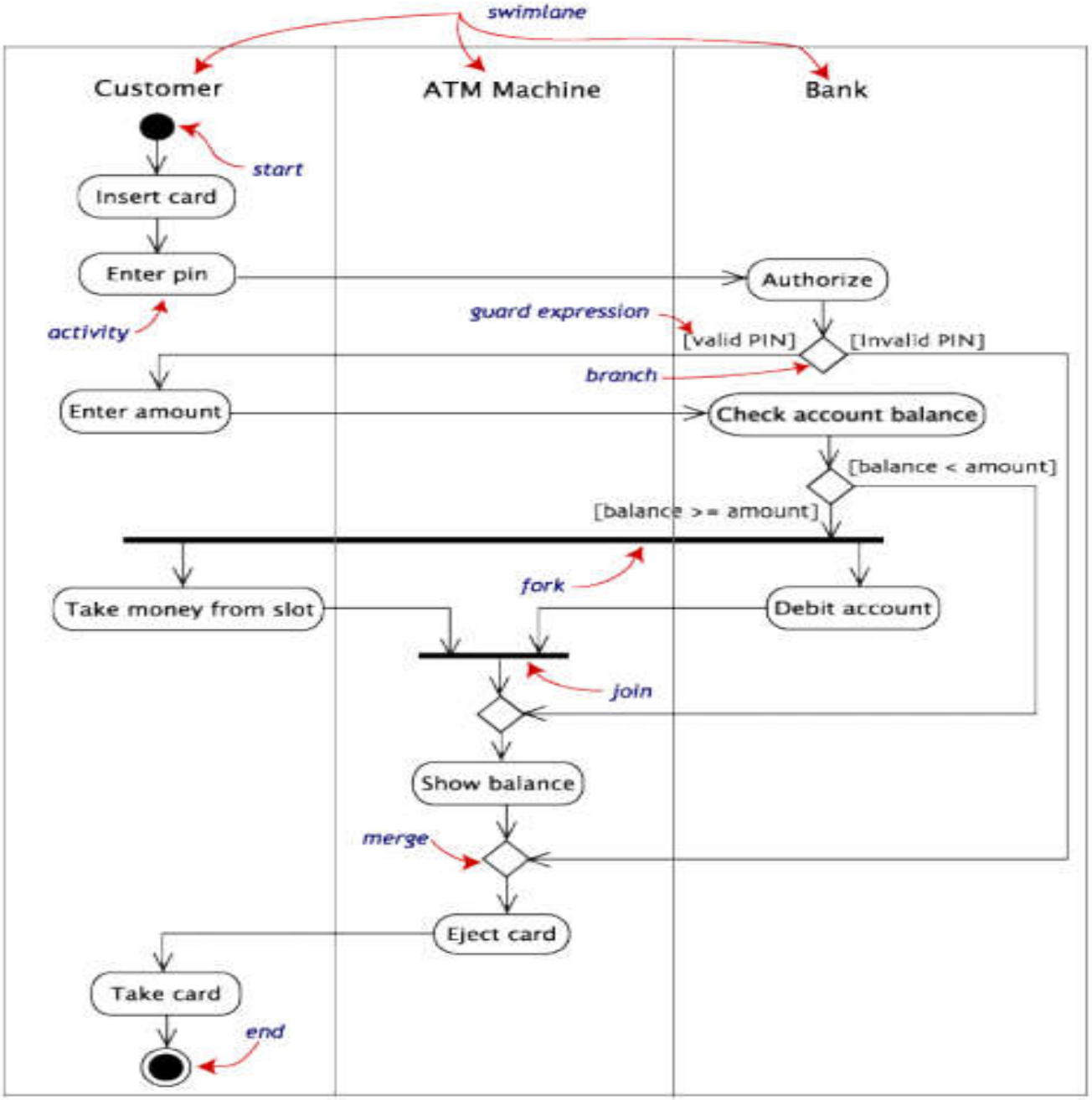
- In UML terminology, each group is called as *Swimlanes* and represented with separating vertical solid lines among neighbors

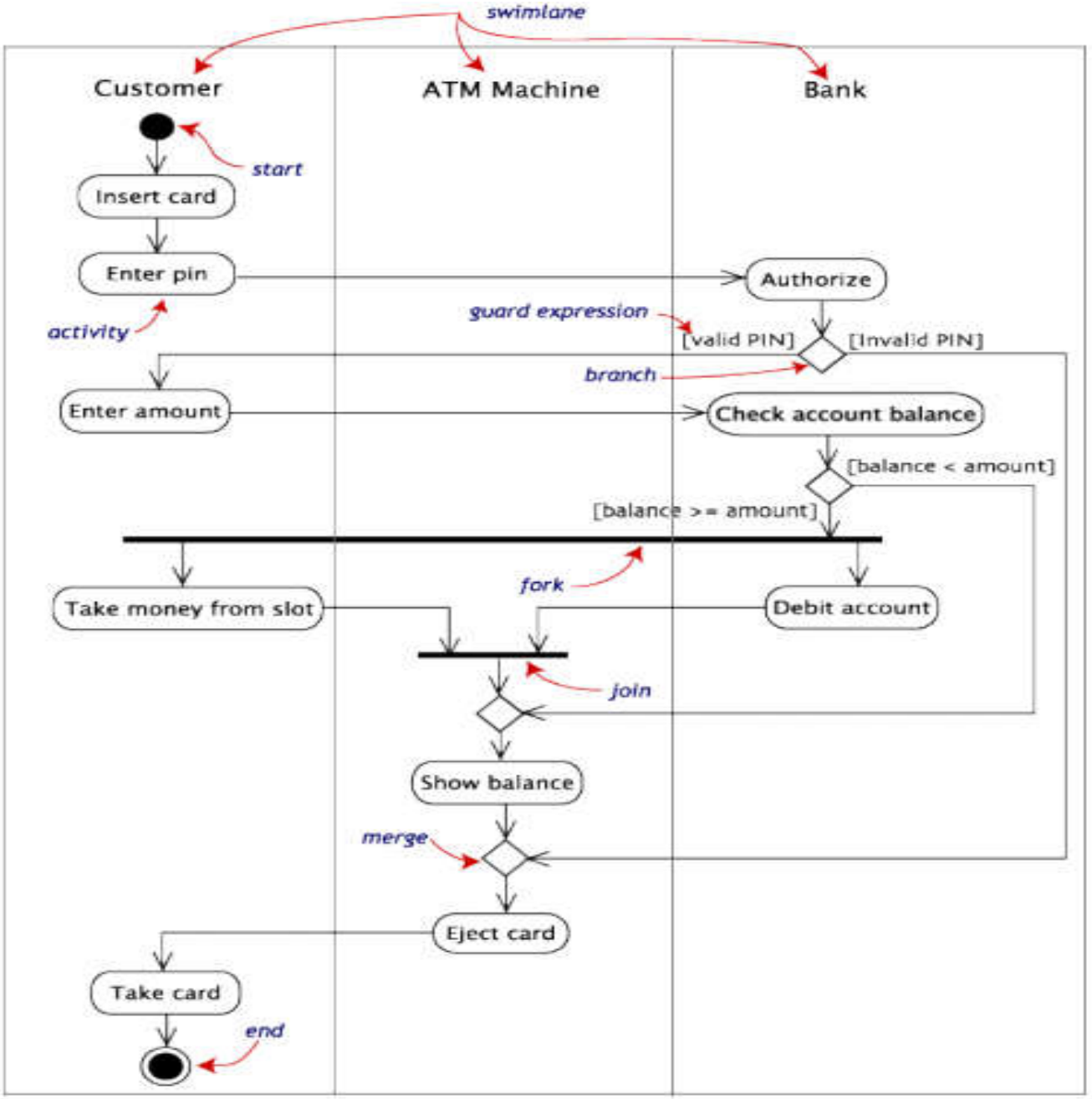
# Example : Swimlanes



Q. Draw Activity diagram for condition as:

When you insert an Atm card to withdraw the money (consider swimlanes also)


swimlane
Customer
ATM Machine
Bank
start
Insert card
Enter pin
activity
Authorize
guard expression
[valid PIN]
[Invalid PIN]
branch
Enter amount
Check account balance
[balance < amount]
[balance >= amount]
fork
Take money from slot
Debit account
join
Show balance
merge
Eject card
Take card
end


swimlane
Customer
ATM Machine
Bank
start
Insert card
Enter pin
activity
Authorize
guard expression
[valid PIN]
[Invalid PIN]
branch
Enter amount
Check account balance
[balance < amount]
[balance >= amount]
fork
Take money from slot
Debit account
join
Show balance
merge
Eject card
Take card
end

# Block Diagram

## What is a Block Diagram?

A block diagram is a specialized, high-level flowchart used in engineering. It is used to design new systems or to describe and improve existing ones. Its structure provides a high-level overview of major system components, key process participants, and important working relationships.

- Its high-level perspective, it may not offer the level of detail required for more comprehensive planning or implementation.

- A block diagram is especially focused on the input and output of a system.

- It cares less about what happens getting from input to output. This principle is referred to as black box in engineering

## How to Make a Block Diagram ?

- Block diagrams are made similar to flowcharts.

- You will want to create blocks, often represented by rectangular shapes, that represent important points of interest in the system from input to output.

- Lines connecting the blocks will show the relationship between these components.

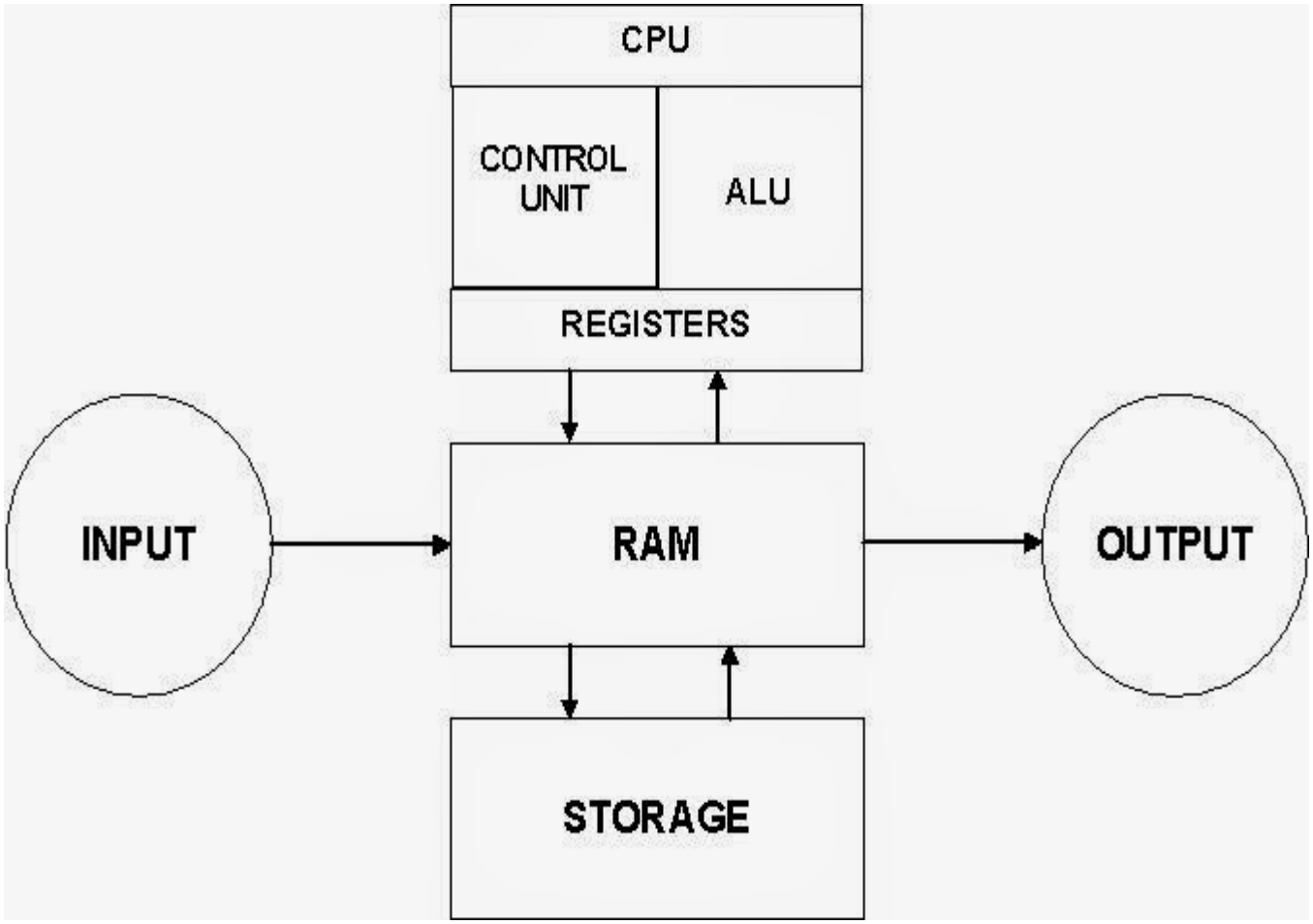



Fig: Ex :1 Block diagram of computer

# Block Diagram

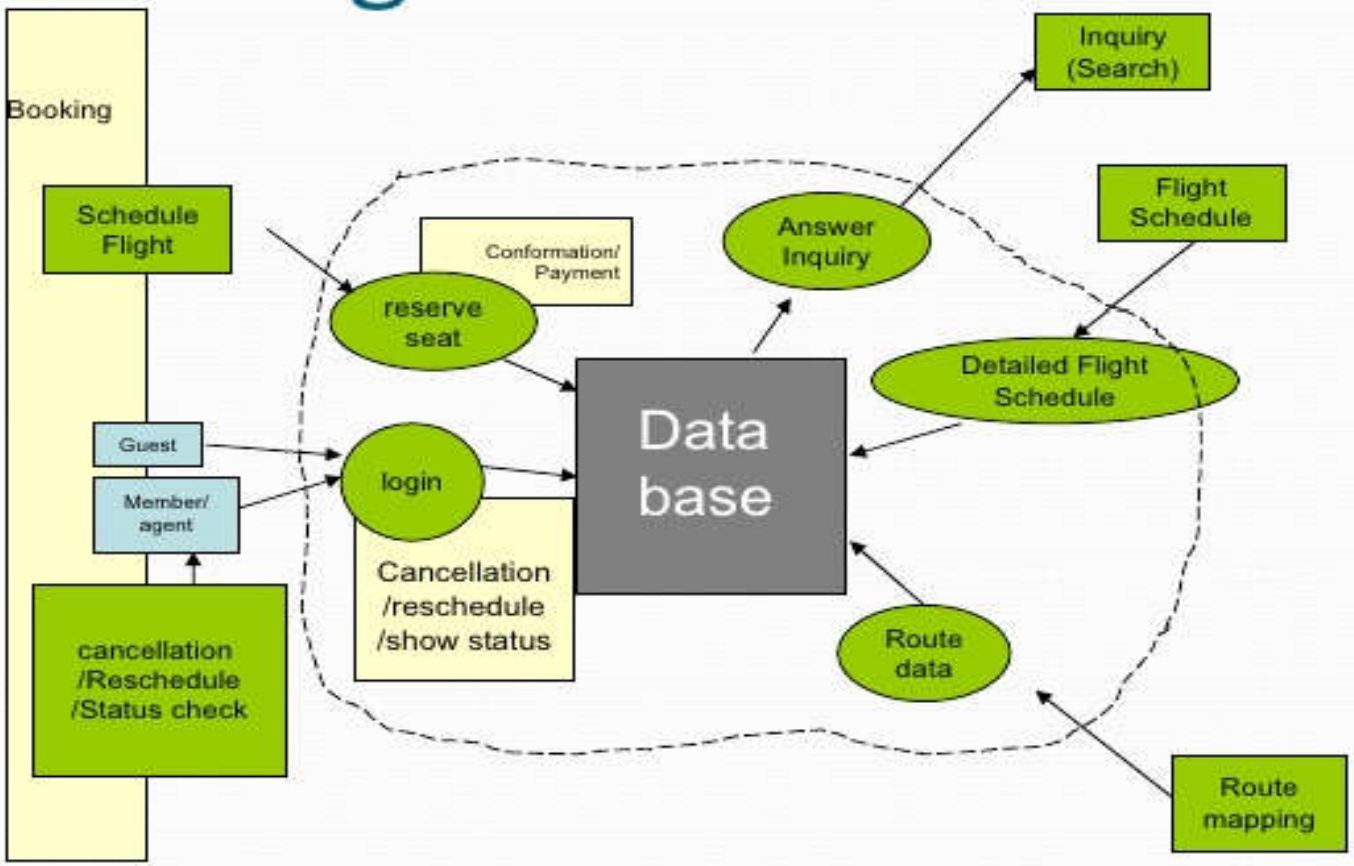



Fig: Flight reservation system

# DFD(Data flow diagram)

- DFD helps a software designer to describe the transformations taking place in the path of data from input to output.

- DFD should not be confused with a flowchart. A DFD represents the flow of data whereas flowchart depicts the flow of control.

# The components of a DFD

## 1)The process:

- The first component of the DFD is known as a process. Common synonyms are a bubble, a function, or a transformation.

- The process shows a part of the system that transforms inputs into outputs; that is, it shows how one or more inputs are changed into outputs.

The process is represented graphically as a circle,

## 2)The flow

- A flows is represented graphically by an arrow into or out of a process.

- For most of the systems that you model as a systems analyst, the flows will indeed represent data, that is, bits, characters, messages, floating point numbers, and the various other kinds of information that computers can deal with.


**Fig: flow**


**Fig: flow with the process**

## 3)The store

- The store is used to model a collection of data packets at rest. The notation for a store is two parallel lines.

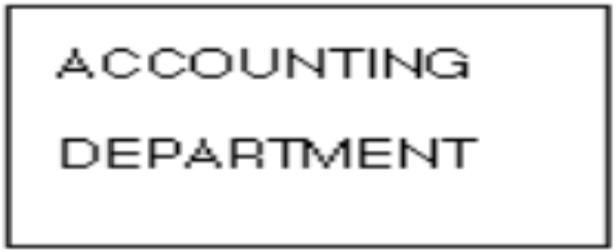
- For the systems analyst with a data processing background, it is tempting to refer to the stores as files or databases (e.g., a disk file organized with Oracle, DB2, Sybase, Microsoft Access, or some other well-known database management system


**Or ,**


**Fig:Store**

**3) Terminator**

- it is graphically represented as a rectangle, Terminators represent external entities with which the system communicates.
- Typically, a terminator is a person or a group of people, for example, an outside organization or government agency, or a group or department that is within the same company or organization



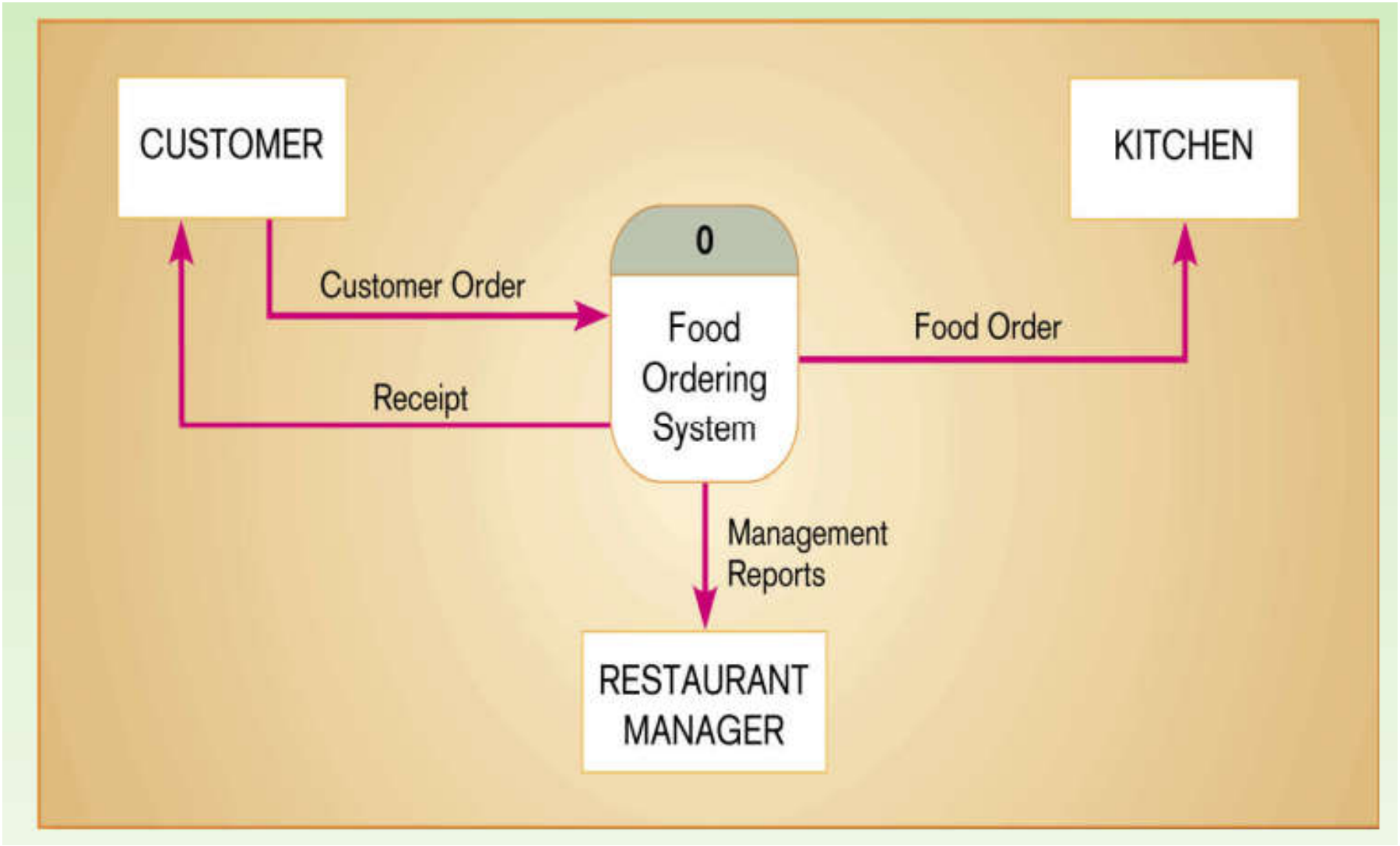



**Fig: overall view of component of DFD**

# RULES OF DATA FLOW

- **Data can flow from**

-external entity to process
-process to external entity
-process to store and back
-process to process

- **Data cannot flow from**

-external entity to external entity
-external entity to store
-store to external entity
-store to store

There are various levels of DFD,

- **Level 0 DFD** (also known as **context diagram**): Shows an overall view of the system.

- **Level 1 DFD**: Elaborates level 0 DFD and splits the process into a detailed form

- **Level 2 DFD**: Elaborates level 1 DFD and displays the process(s) in a more detailed form

.• **Level 3 DFD**: Elaborates level 2 DFD and displays the process(s) in a detailed form.

which provide detail about the input, processes, and output of a system. Note that the level of detail of process increases with increase in level(s).

To understand various levels of DFD, let us consider an example of banking system.....

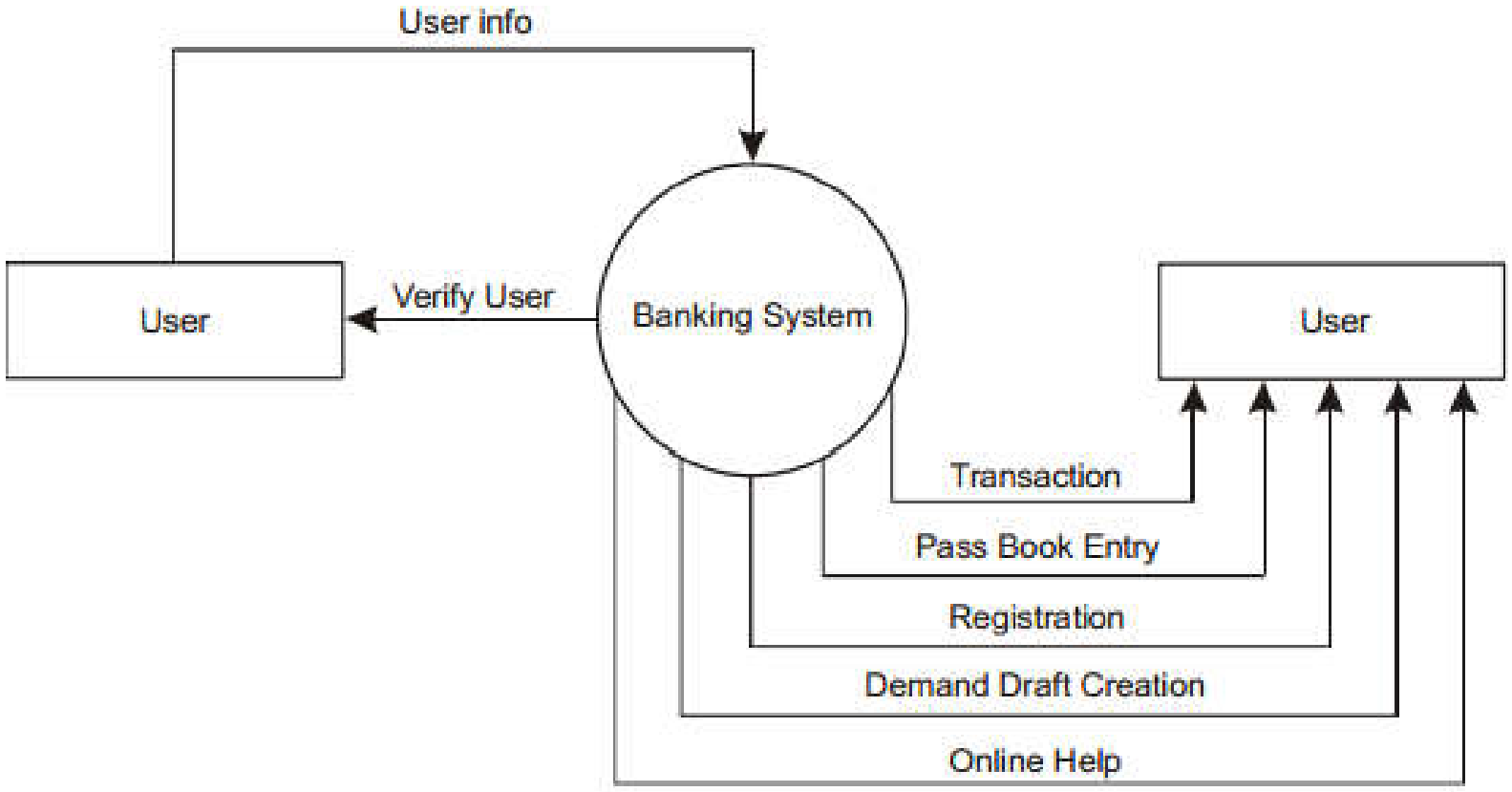



Fig : DFD level 0

This DFD represents how ‘user’ entity interacts with ‘banking system’ process and avails its services. The level 0 DFD depicts the entire banking system as a single process. There are various tasks performed in a bank, such as transaction processing, pass book entry, registration, demand draft creation, and online help. The data flow indicates that these tasks are performed by both the user and bank. Once the user performs transaction, the bank verifies whether the user is registered in the bank or not.

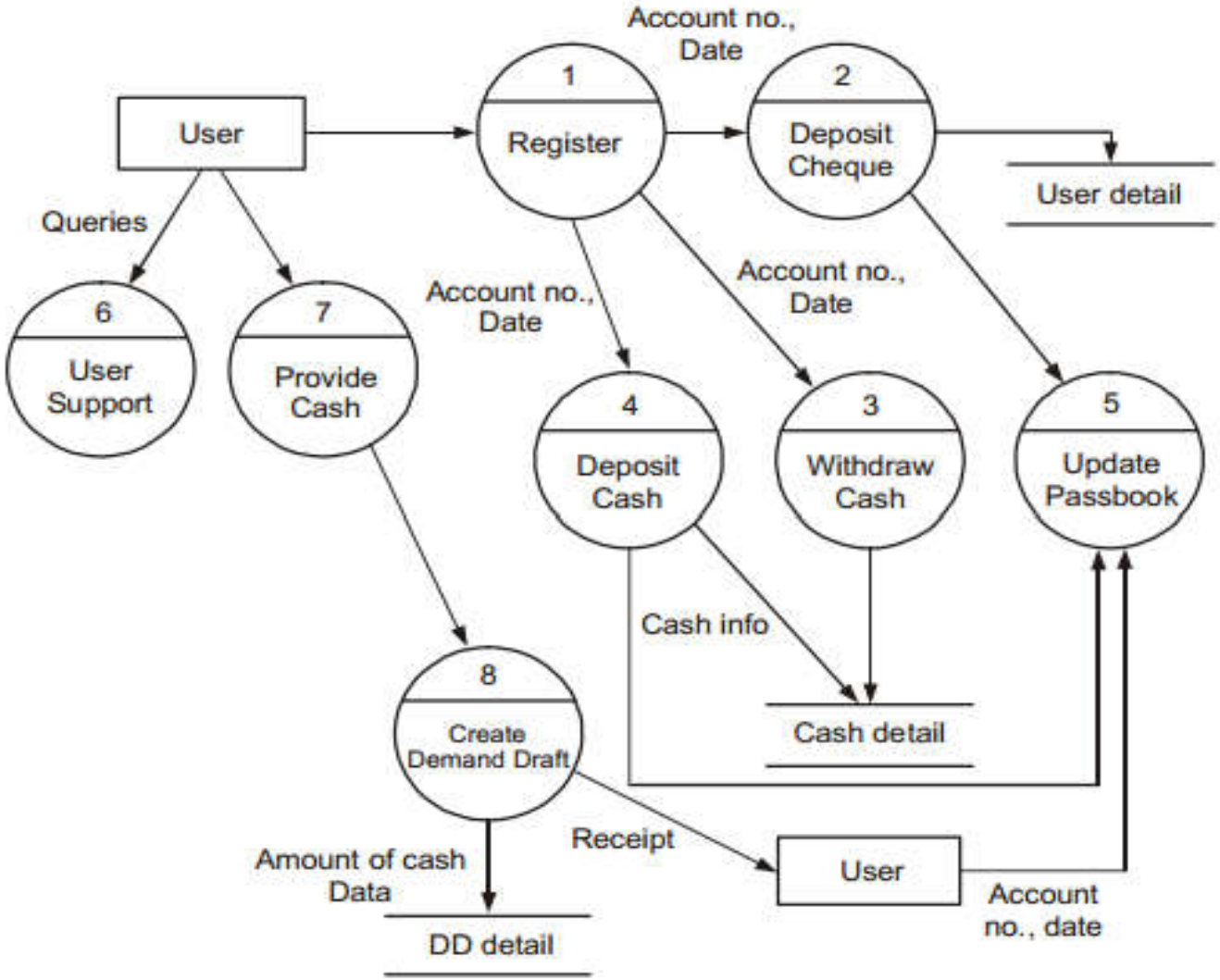



Fig : DFD level 1

- In this DFD, ‘user’ entity is related to several processes in the bank, which include ‘register’, ‘user support’, and ‘provide cash’. Transaction can be performed if user is already registered in the bank. Once the user is registered, user can perform transaction by the processes, namely, ‘deposit cheque’, ‘deposit cash’, and ‘withdraw cash’. Note that the line in the process symbol indicates the level of process and contains a unique identifier in the form of a number.

If user is performing transaction to deposit cheque, the user needs to provide cheque to the bank. The user's information, such as name, address, and account number is stored in 'user_detail' data store, which is a database. If cash is to be deposited and withdrawn, then, the information about the deposited cash is stored in 'cash_detail' data store. User can get demand draft created by providing cash to the bank. It is not necessary for the user to be registered in that bank to have demand draft.

The details of amount of cash and date are stored in ‘DD_detail’ data store. Once the demand draft is prepared, its receipt is provided to the user. The ‘user support’ process helps users by providing answers to their queries related to the services available in the bank…

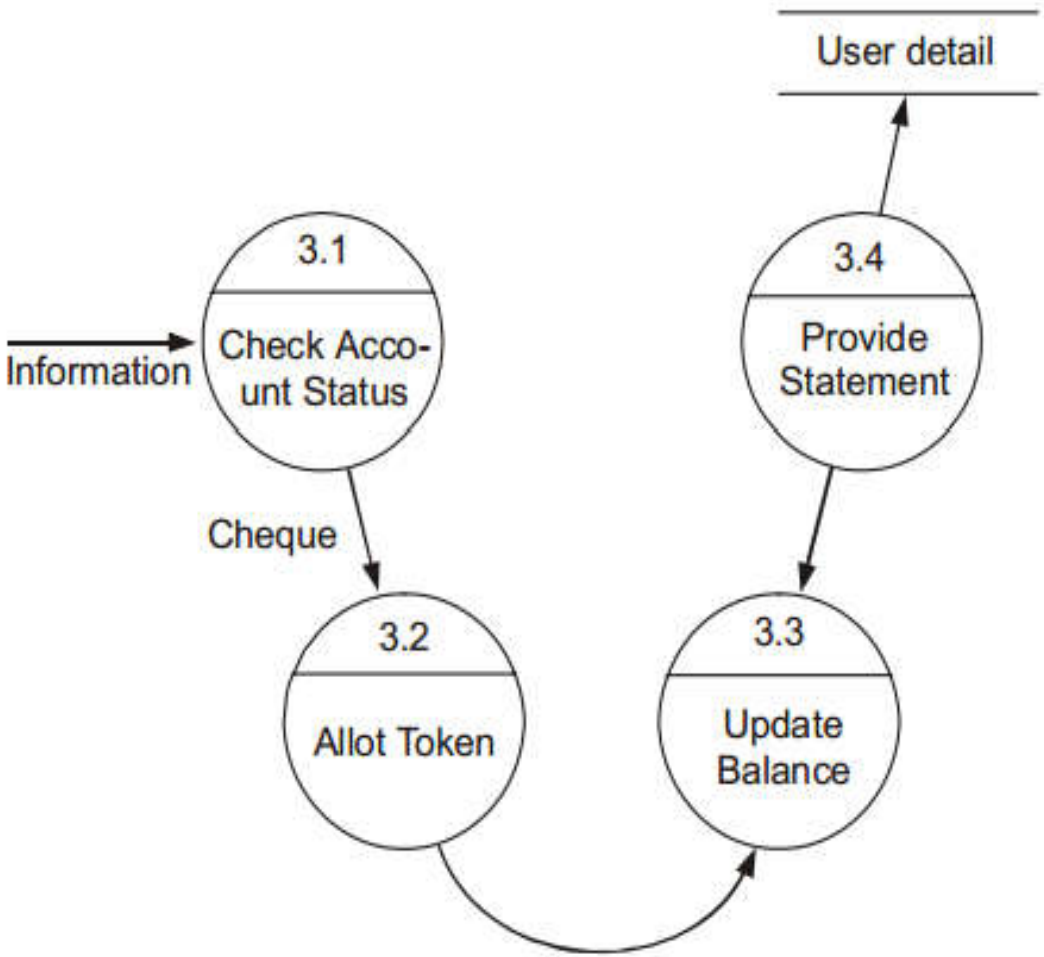



Fig : DFD level 2

To illustrate level 2 DFD. The information collected from level 1 DFD acts as an *input* to level 2 DFD. Note that 'withdraw cash' process is numbered as '3' in level 1 and contains further processes, which are numbered as '3.1', '3.2', '3.3', and '3.4' in level 2 . These numbers represent the sublevels of 'withdraw cash' process. To withdraw cash, bank checks the status of balance in user's account (as shown by 'check account status' process) and then allots token (shown as 'allot token' process).

After the user withdraws cash, the balance in user's account is updated in the 'user_detail' data store and statement is provided to the user.

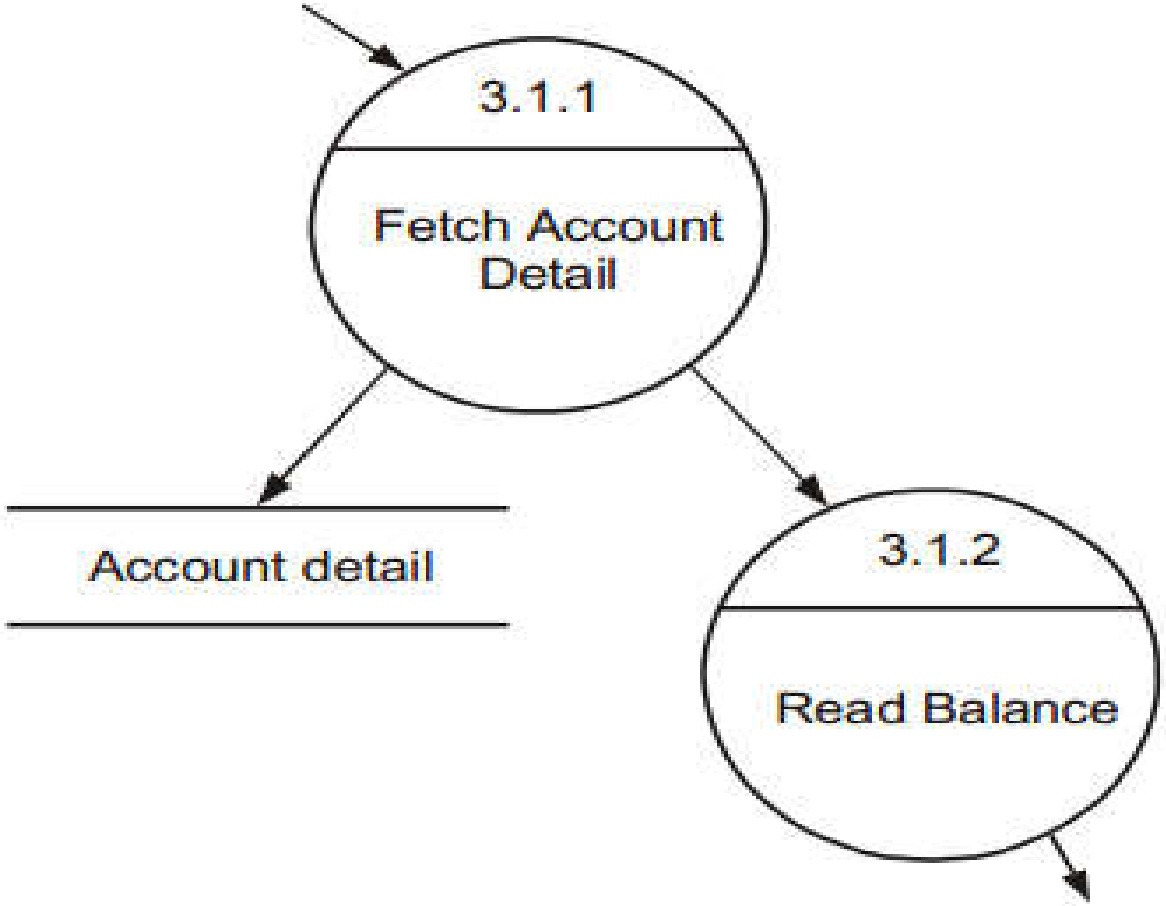



Fig : DFD level 3

- If a particular process of level 2 DFD requires elaboration, then this level is further refined into level 3 DFD. Let us consider the process ‘check account status’ (see level 2) to illustrate level 3 DFD. In level 3, this process contains further processes numbered as ‘3.1.1’ and ‘3.1.2’, which describe the sublevels of ‘check account status’ process. To check the account status, the bank fetches the account detail (shown as ‘fetch account detail’ process) from the ‘account_detail’ data store. After fetching the details, the balance is read (shown as ‘read balance’ process) from the user’s account

# Strengths

- DFDs have diagrams that are easy to understand, check and change data.
- DFDs help tremendously in depicting information about how an organization operations.
- They give a very clear and simple look at the organization of the interfaces between an application and the people or other applications that use it.

## Weaknesses

- Modification to a data layout in DFDs may cause the entire layout to be changed.

- The number of units in a DFD in a large application is high. Therefore, maintenance is harder, more costly and error prone. This is because the ability to access the data is passed explicitly from one component to the other. This is why changes are impractical to be made on DFDs especially in large system.

## *(CASE)*

*COMPUTER AIDED SOFTWARE ENGINEERING*

CASE is a term covering a whole range of tools and methods that SUPPORT SOFTWARE SYSTEM DEVELOPMENT.

- CASE tools are programs (software) that automate or support one or more phases of a systems development life cycle.

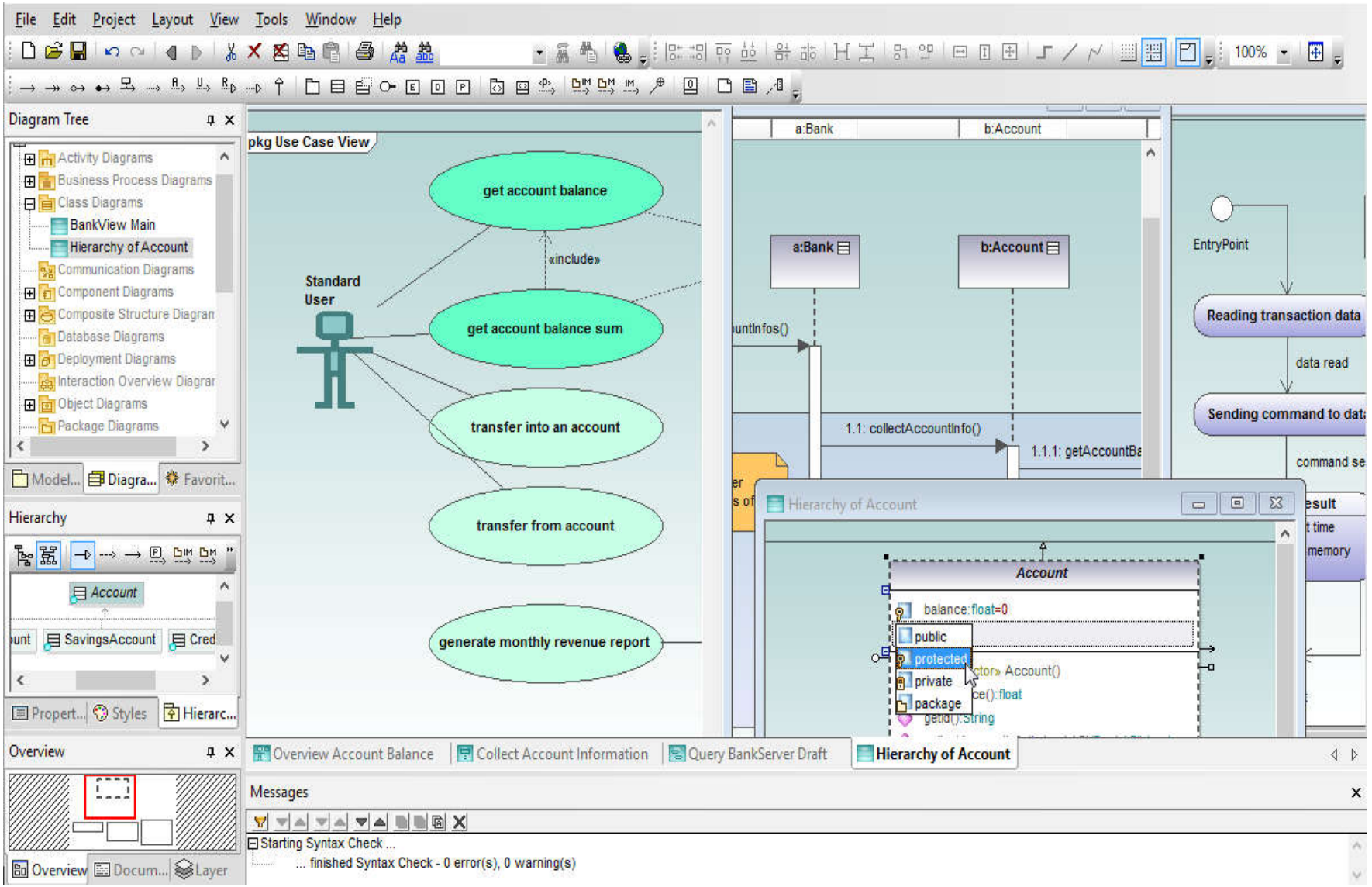
- A collection of tools used to support the software development process

In other words,

1. Software that is used to support software process activities.

2. Provides software process support by
   - •automating some process activities
   - •providing information about the software being developed

3. Currently used in every phase/workflow of life cycle


File Edit Project Layout View Tools Window Help
Diagram Tree
Activity Diagrams
Business Process Diagrams
Class Diagrams
BankView Main
Hierarchy of Account
Communication Diagrams
Component Diagrams
Composite Structure Diagran
Database Diagrams
Deployment Diagrams
Interaction Overview Diagrar
Object Diagrams
Package Diagrams
Model... Diagra... Favorit...
Hierarchy
Account
SavingsAccount
Propert... Styles Hierarc...
Overview
Overview Docum... Layer
pkg Use Case View
get account balance
«include»
Standard User
get account balance sum
transfer into an account
transfer from account
generate monthly revenue report
a:Bank
b:Account
1.1: collectAccountInfo()
1.1.1: getAccountBa
EntryPoint
Reading transaction data
data read
Sending command to data
command se
Hierarchy of Account
Account
balance:float=0
public
protected
private
package
Overview Account Balance
Collect Account Information
Query BankServer Draft
Hierarchy of Account
Messages
Starting Syntax Check ...
... finished Syntax Check - 0 error(s), 0 warning(s)

Two types of tools used by software engineers:

1. Analytical tools
   - Stepwise refinement
   - Cost-benefit analysis
   - Software metrics

**2. CASE tools**

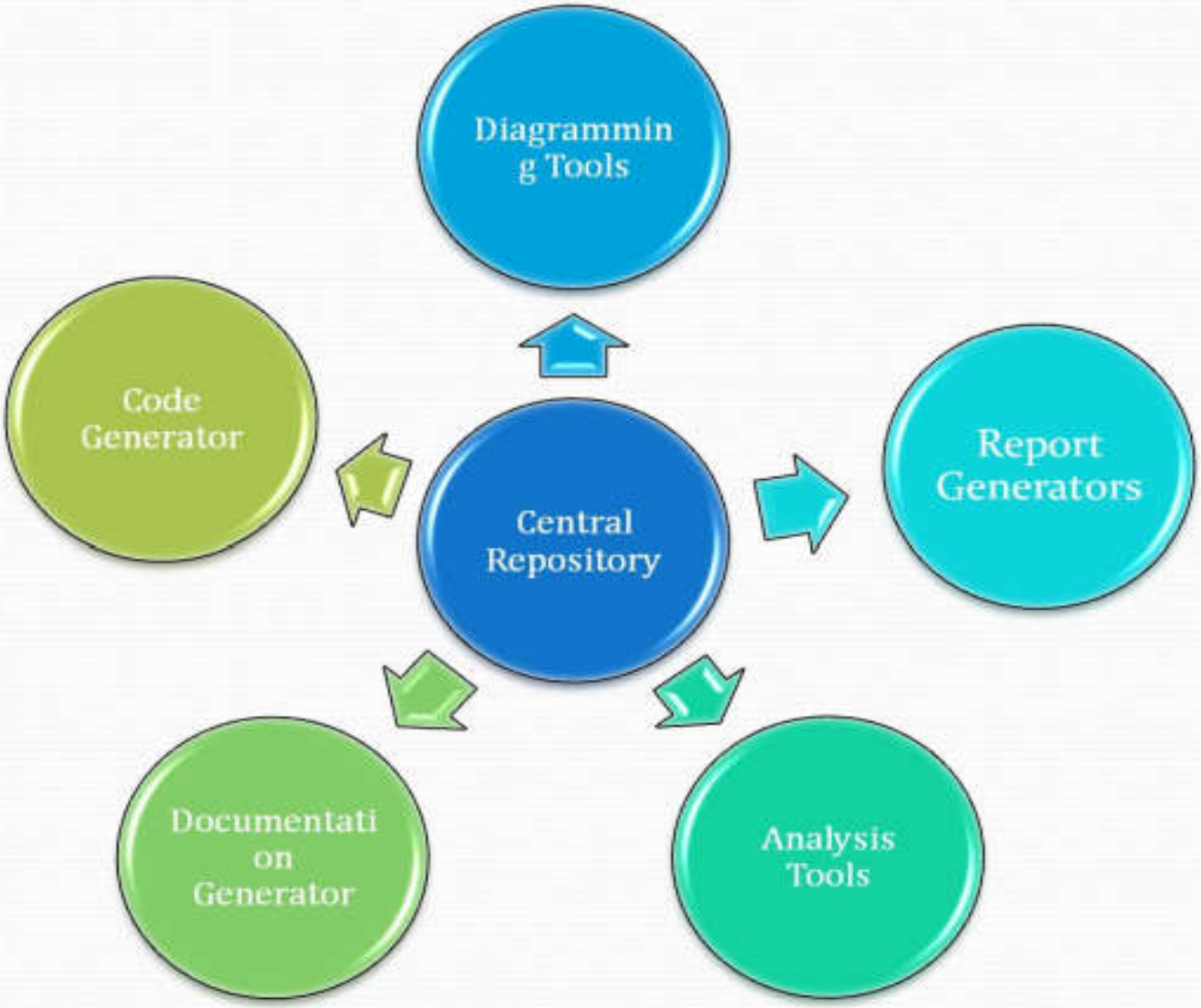



**Fig: component of Case Tools**

# 1. Central Repository

- Centralized Database.

- Used to store Graphical Diagrams & Prototype Forms and Reports of analysis and diagramming workflow

- Act as
  Information Repository

## 2. Report Generator

- Used to
  Create, modify and test prototypes of computer displays and reports.

  Identify which data items to display or collect for each screen or report

## 3. Diagramming Tool

- Allow you to represent a system and its components visually.

- Allows higher level processes to be easily decomposed.

- Can examine processes or data models at high or low level.

## 4. Analysis tools

- Generate reports that help identify possible inconsistencies, redundancies and omissions.

- Generally focus on
  - diagram completeness and consistency.
  - data structures and usage.

## 5. Documentation Tool

- Create standard reports based on contents of repository.
- Need textual descriptions of needs, solutions, trade-offs, diagrams of data and processes, prototype forms and reports, program specifications and user documentation.
- High-quality documentation leads to 80% reduction in system maintenance effort in comparison to average quality documentation

## 6. Code Generation Tool

- Create code for the custom feature in object model

- Code Generation Tool helps in:
  Connect to the Repository.
  Select the Model.
  Select the custom features to generate code for.
  Define properties for each custom feature.
  Specify the output of the project.

## 6. Code Generation Tool

- Create code for the custom feature in object model

- Code Generation Tool helps in:
  - Connect to the Repository.
  - Select the Model.
  - Select the custom features to generate code for.
  - Define properties for each custom feature.
  - Specify the output of the project.

# TYPES OF CASE TOOLS

## 1.Upper-CASE:-

Upper CASE is focused in supporting project identification and selection, project initiation, project planning, analysis and design.

1. Focuses on Analysis Phase
   - Diagramming Tools
   - Report Generator
   - Analysis Tool

## 2.Lower-CASE:-

Lower CASE provides support for the implementation and maintenance phases.

1. Supports Programming and Integration tasks.

2. Focuses on
   - Central Repository
   - Code Generator
   - Configuration Management

## 3. I-CASE (integrative case):-

support the entire SDLC.

1. Supports both Upper CASE Tools and Lower CASE Tools.
2. Focuses on
   - Analysis
   - Code
   - Design
   - Database

## CASE Usage Within the SDLC

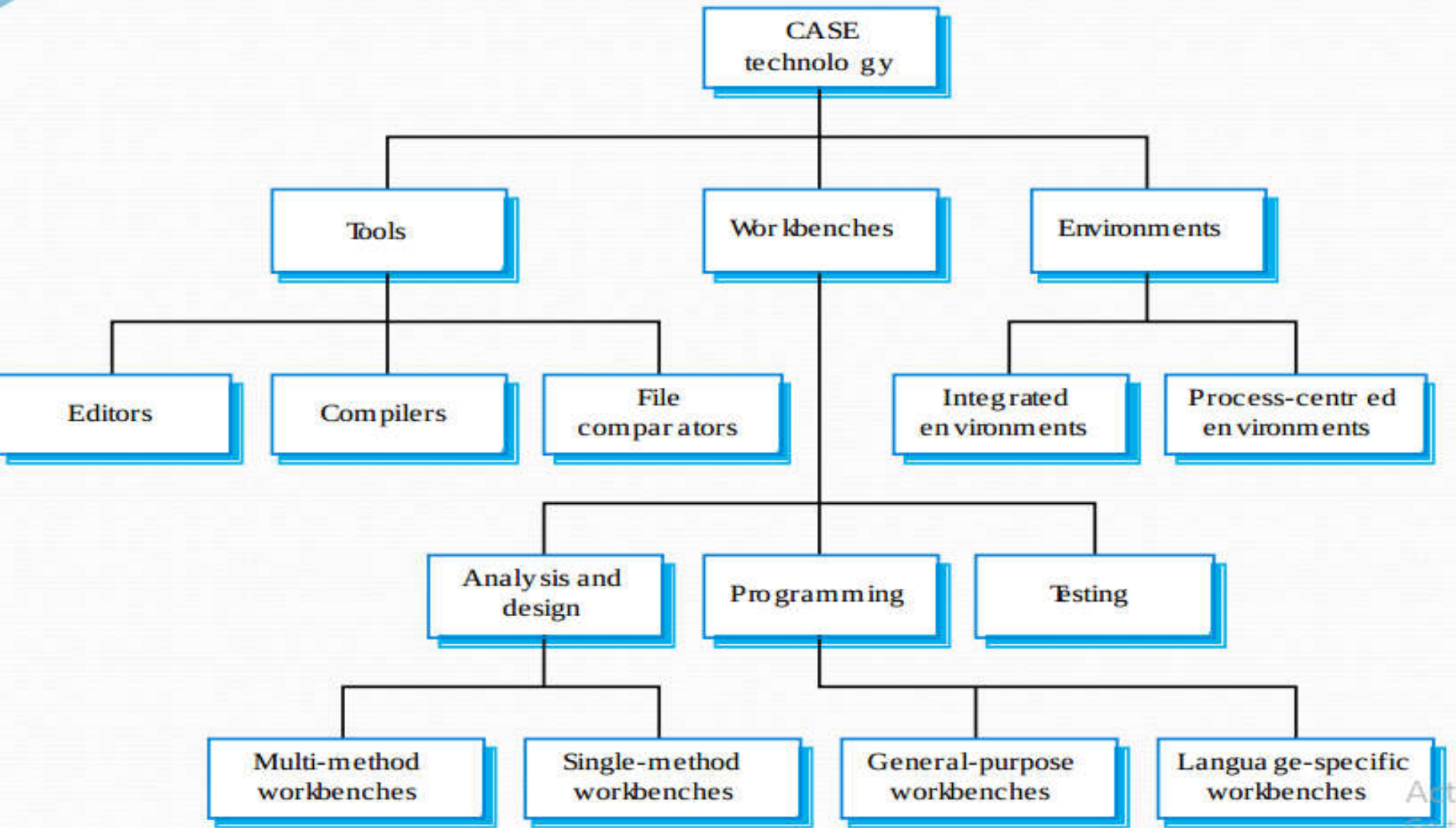
| SDLC Phase | Key Activities | CASE Tool Usage |
|---|---|---|
| Project identification and selection | Display and structure high-level organizational information | Diagramming and matrix tools to create and structure information |
| Project initiation and planning | Develop project scope and feasibility | Repository and documentation generators to develop project plans |
| Analysis | Determine and structure system requirements | Diagramming to create process, logic and data models |

| SDLC Phase | Key Activities | CASE Tool Usage |
|---|---|---|
| Design | Create new system designs | Form and report generators to prototype designs; analysis and documentation generators to define specifications |
| Implementation | Translate designs into an information system | Code generators and analyzers, form and report generators; documentation generators to develop system and user documentation |
| Maintenance | Evolve information systems | All tools are used . |

# Categories Of CASE Tools

- Tools
- Workbench
- Environment

# Categories Of CASE Tools



## CASE tools can be classified based on functionality

- Management tools: PERT tools, estimation tools
- Editing tools: Test editors, diagram editors, word processors
- Configuration management tools: Version management system, change management systems
- Prototyping tools: Very high-level languages, user interface generators
- Method supports tools: Design editors, data dictionaries, code generators
- Language processing tools: Compiler, Interpreters

# CASE Workbenches

- CASE workbenches are collections of integrated tools, which support a specific software life cycle stage.
- CASE tools to support analysis & design, programming and testing are widely used in industry.

**i) Programming Workbenches**: Is a set of integrated tools to support the process of program development. A typical programming workbench includes the following tools:

- Language compiler
- Structured editor: to create and edit source programs Linker: to link object code components
- Loader: to load an executable program into the computer memory
- Interactive debugger: to trace and control the execution of user program

**ii) Analysis & Design Workbenches**: Typical components of an analysis & design workbench are:

- Diagramming editors: Are used to create DFDs, ERDS charts etc. The editor is not just a drawing tool but understands the meaning of different symbols/objects. It captures information about these objects and saves this information in a central repository.
- Checking facilities: Are used to ensure that models created by analysts/designers are internally consistent and complete and all models are consistent with one another (externally consistent)

- Query language facilities: Is used to browse the repository and examine completed designs
- Report generation facilities: Are used to automatically generate system documentation
- Form generation tools: Are used to specify screen and document formats
- Skeleton code generators: Are used to generate code or code segments automatically from the design stored in the central repository
- Import/Export facilities: Allow the interchange of information with other development tools

**iii) Testing Workbenches:** Testing workbenches are open collections of testing tools. A typical testing workbench consists of:

- Test manager: to record testing
- Test data generator: to generate test cases
- Oracle: to generate predictions of expected results
- File comparators: to compare results of program test
- Report generator: to automatically generate system documentation

## CASE tool environment may be expected to provide the following:

- Automated graphics facilities for producing charts and diagrams in support of modeling activities
- Screen and report generators to assist in designing the dialogue/presentation of the proposed information systems
- Data dictionaries providing a central repository for all definitions used by the development team

- Reporting facilities ranging from project management statistics through to design inconsistencies.

- Analysis and cross-referencing tools to ensure integration of model views.

- Code generators and documentation generators utilizing the deliverables of the modeling activities to develop working systems.

- **CHARACTERISTICS OF THE CASE TOOLS:**

1. Graphical interface to draw diagram, models, etc.
2. An information repository, data dictionary for official information management, selection, usage, application, storage.
3. Common GUI for integrated multiple tools.
4. Automatic code generator.
5. Automatic testing tools

## Advantages of Case:

The various advantages are:

- **Improvement in productivity:** by reducing the time of design and development.

- **Improvement in information system quality:** by providing automation.

- **Improvement of effectiveness:** performing the right task with minimal effort.

## Disadvantages Of Case:

1. High cost of purchase and use.
2. High cost of training the user.
3. It is not easy to share information between tools.
4. Software people often see CASE as a theft to their job security

**Chapter 2 finished(5 marks)**

**Upto chapter 2 : 25 marks covered**

# !! Attendance please !!

# Thank You!!!