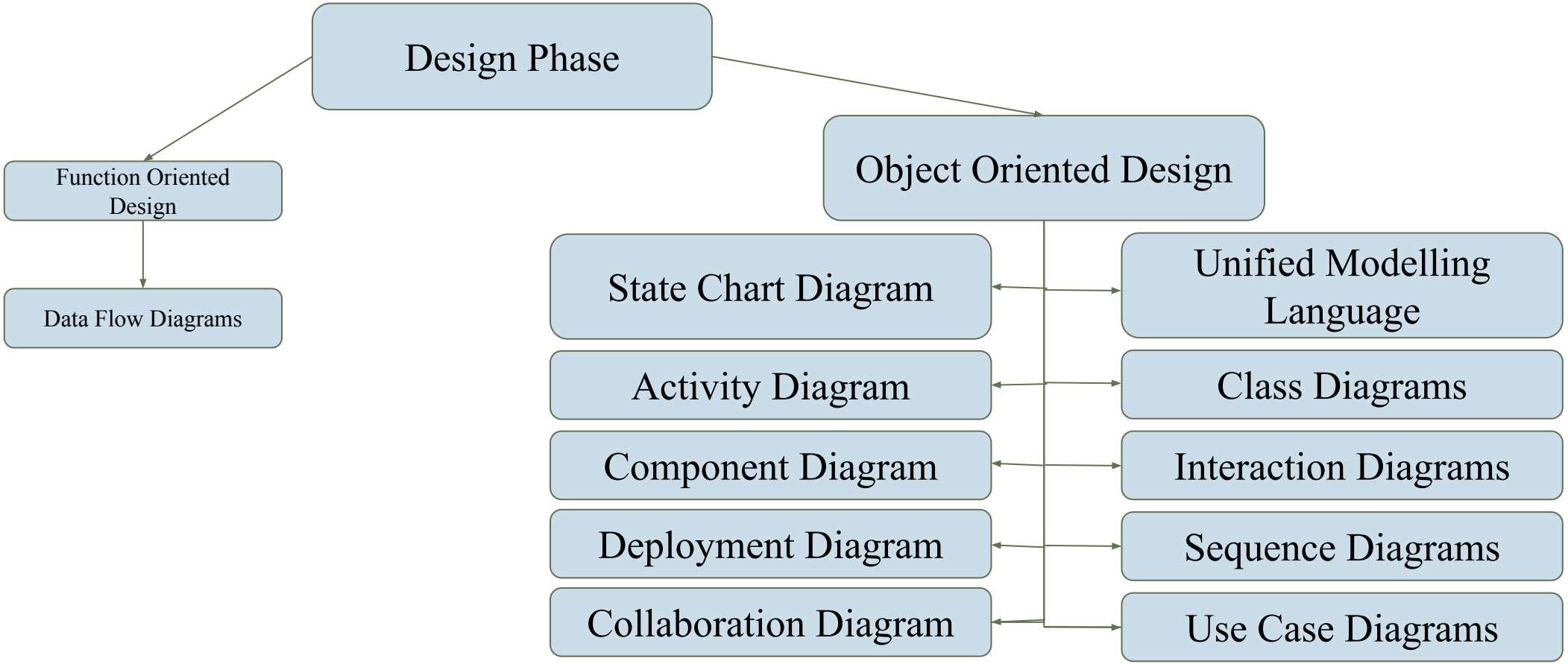
UML: Unified Modelling Language

Outline

- What is UML and why we use UML?
- How to use UML diagrams to design software system?
- What UML Modeling tools we use today?

Design Phase



What is UML and Why we use UML?

- UML → "**Unified Modeling Language**"
- Language: **express idea, not a methodology**
- Modeling: Describing a **software system at a high level of abstraction**
- Unified: **UML has become a world standard**

What is UML and Why we use UML?

- It is a **industry-standard graphical language** for specifying, visualizing, constructing, and documenting the artifacts of software systems

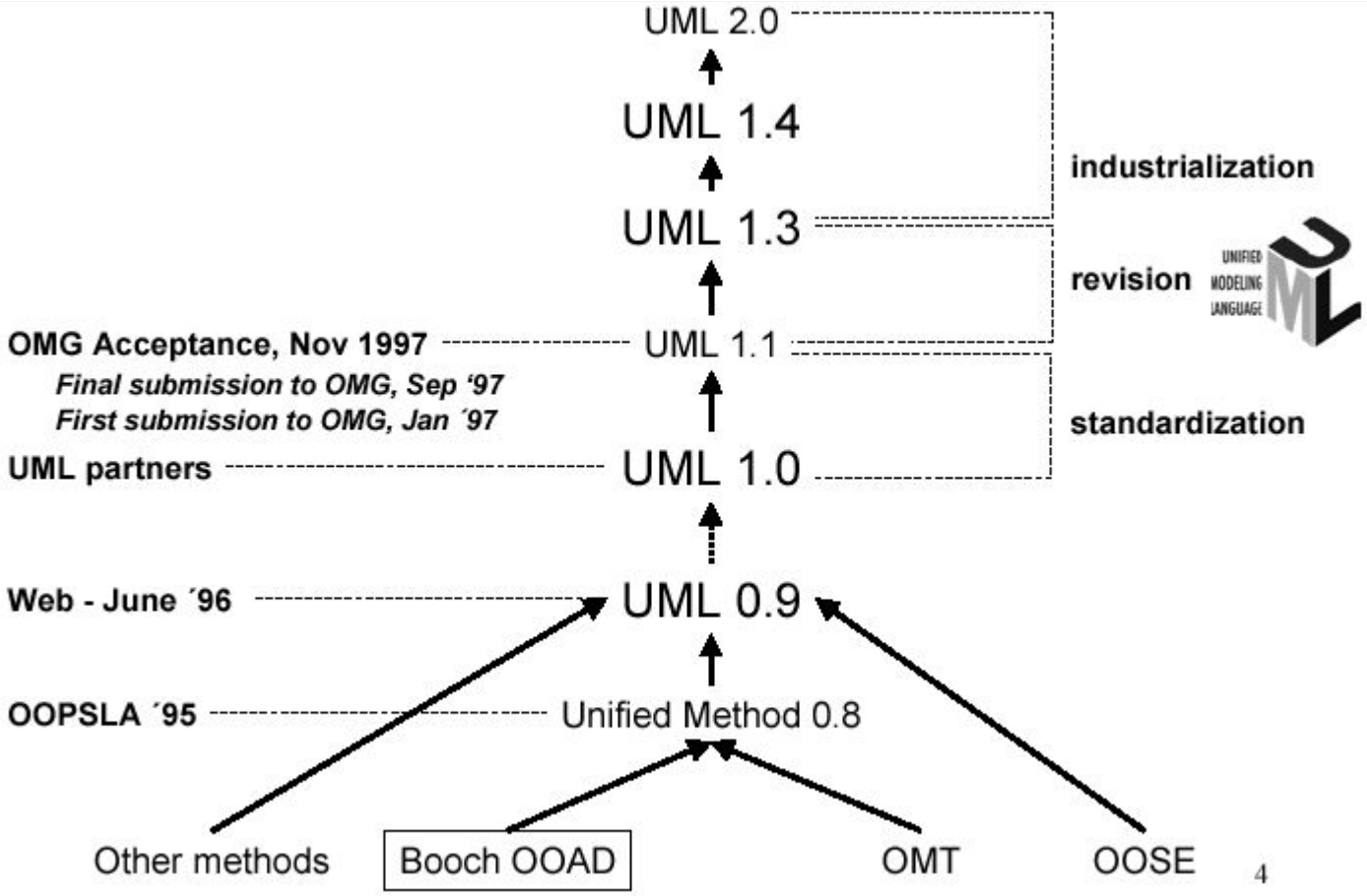
- The UML uses mostly **graphical notations** to express the OO analysis and design of software projects.

- **Simplifies the complex process** of software design

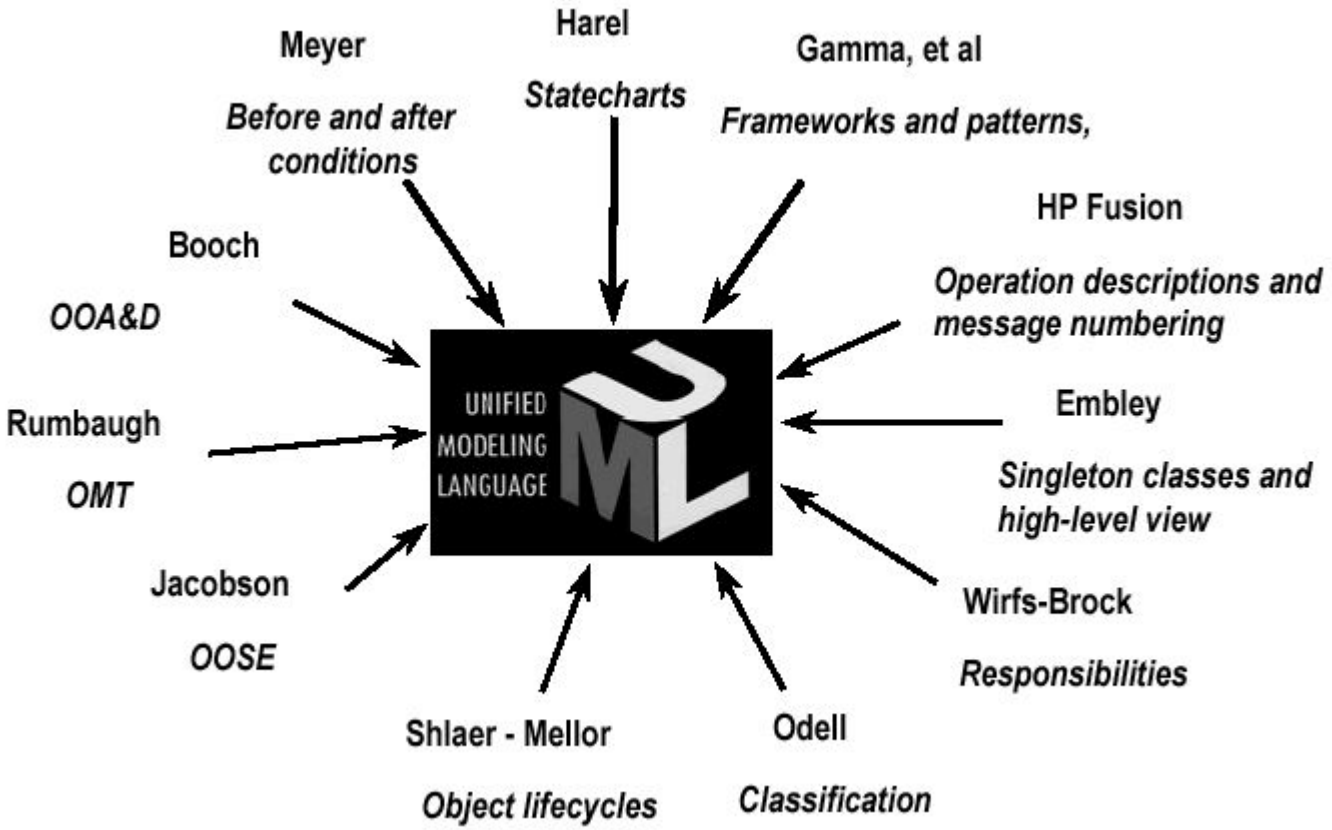
Why use UML

- Open Standard, Graphical notation for
  - ***Specifying, visualizing, constructing, and documenting software systems***
- Language can be used from **general initial design** to **very specific detailed design** across the entire software development lifecycle
- **Increase understanding/communication of product** to customers and developers
- **Support for diverse application areas**
- Support for UML in many software packages today (e.g. Rational, plugins for popular IDE's like NetBeans, Eclipse)
- **Based upon experience** and **needs of the user community**

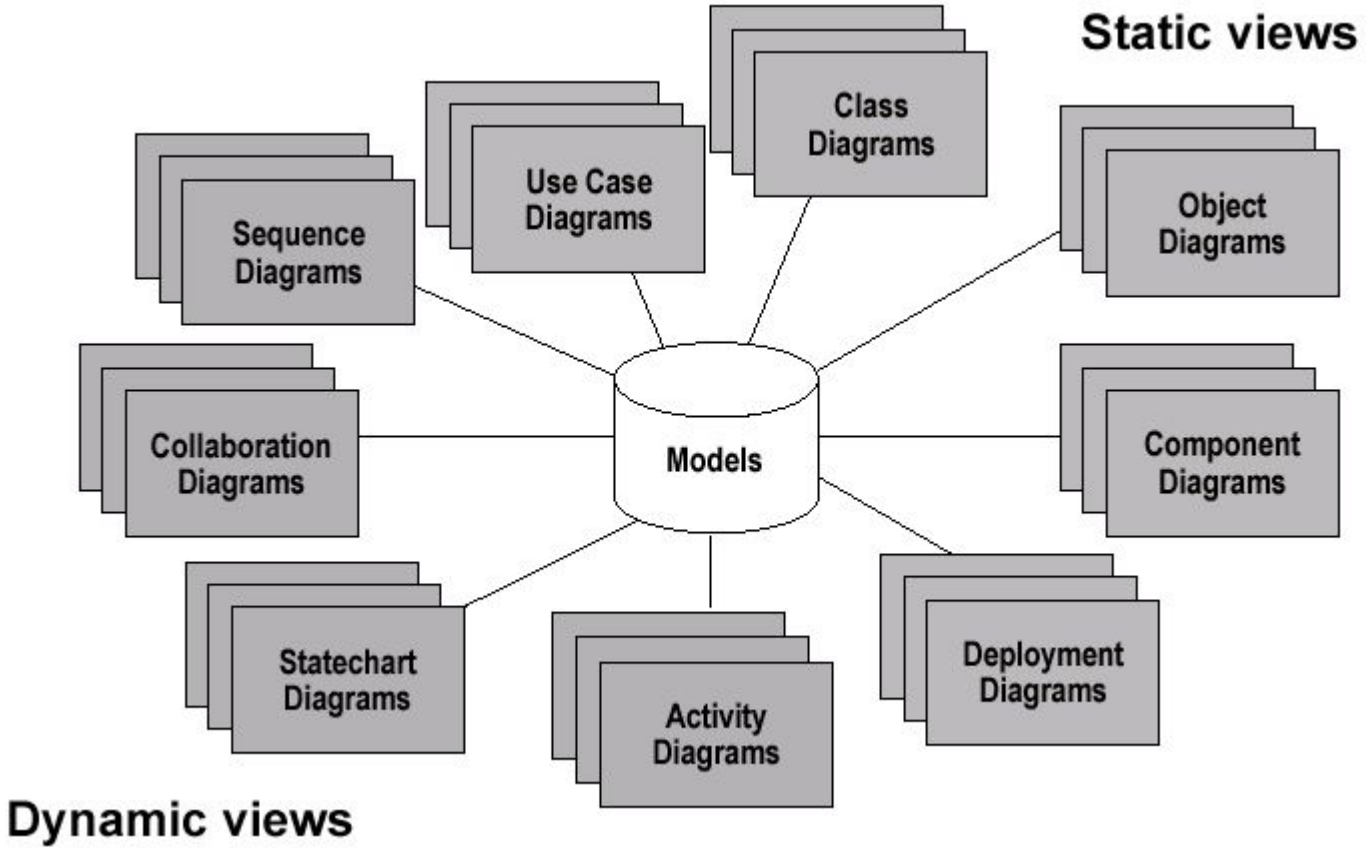
History of UML



Contributions to UML



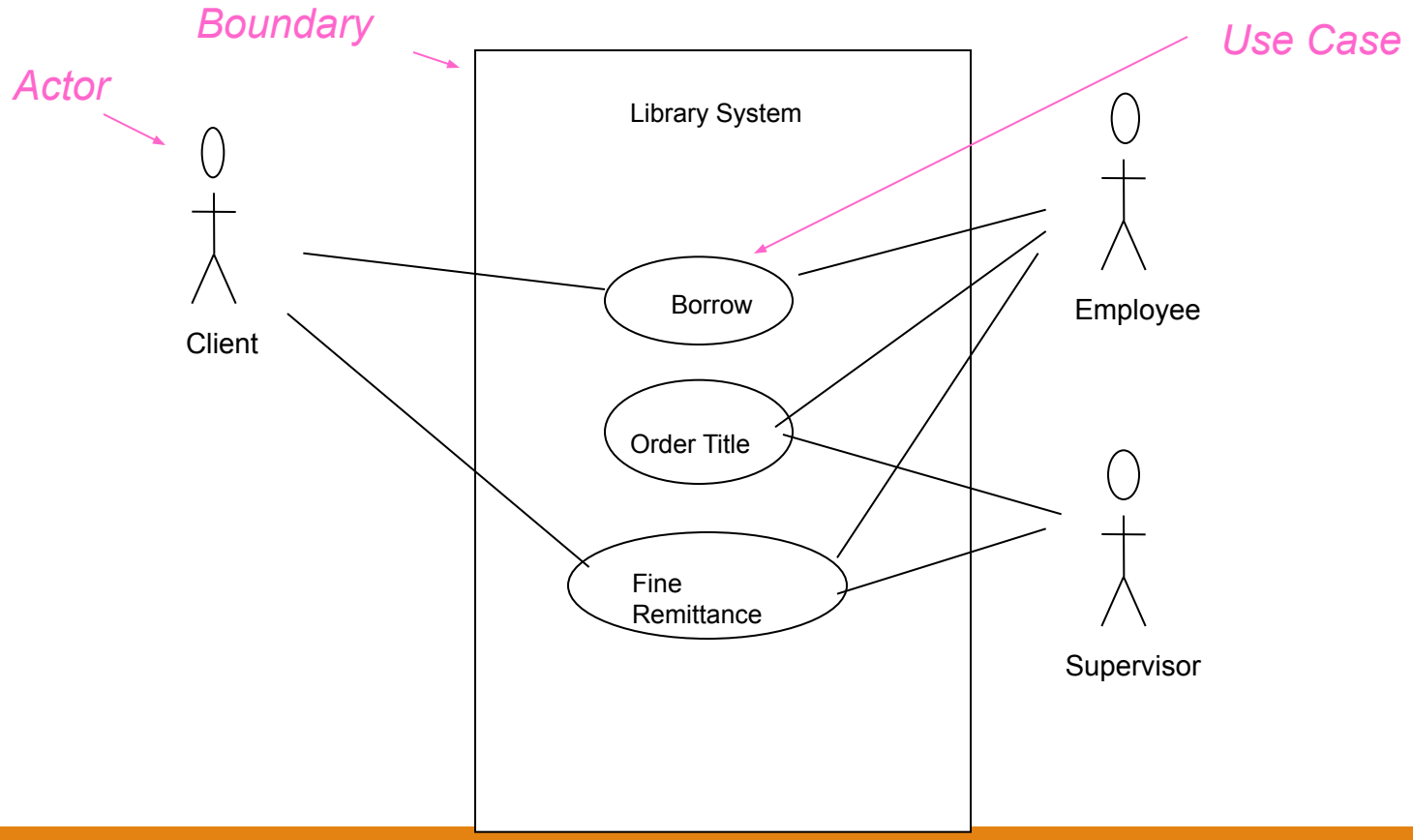
Models, Views, Diagrams



Use-Case Diagrams

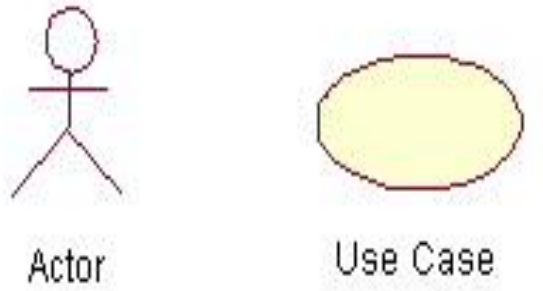
- A use-case diagram is a **set of use cases**
- A use case is a model of the interaction between
  - External users of a software product (actors) and
  - The software product itself
  - More precisely, an actor is a user playing a specific role
- describing a set of user **scenarios**
- **capturing user requirements**
- **contract** between end user and software developers

Use-Case Diagrams



Use-Case Diagrams

- **Actors:** A role that a user plays with respect to the system, including human users and other systems. e.g., inanimate physical objects (e.g. robot); an external system that needs some information from the current system.
- **Use case:** A set of scenarios that describing an interaction between a user and a system, including alternatives.
- **System boundary**: rectangle diagram representing the boundary between the actors and the system.

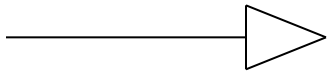


Use-Case Diagrams

- 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.



Use-Case Diagrams

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.

<<include>> ----------→

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".

<<extend>> ----------→

Use-Case Diagrams

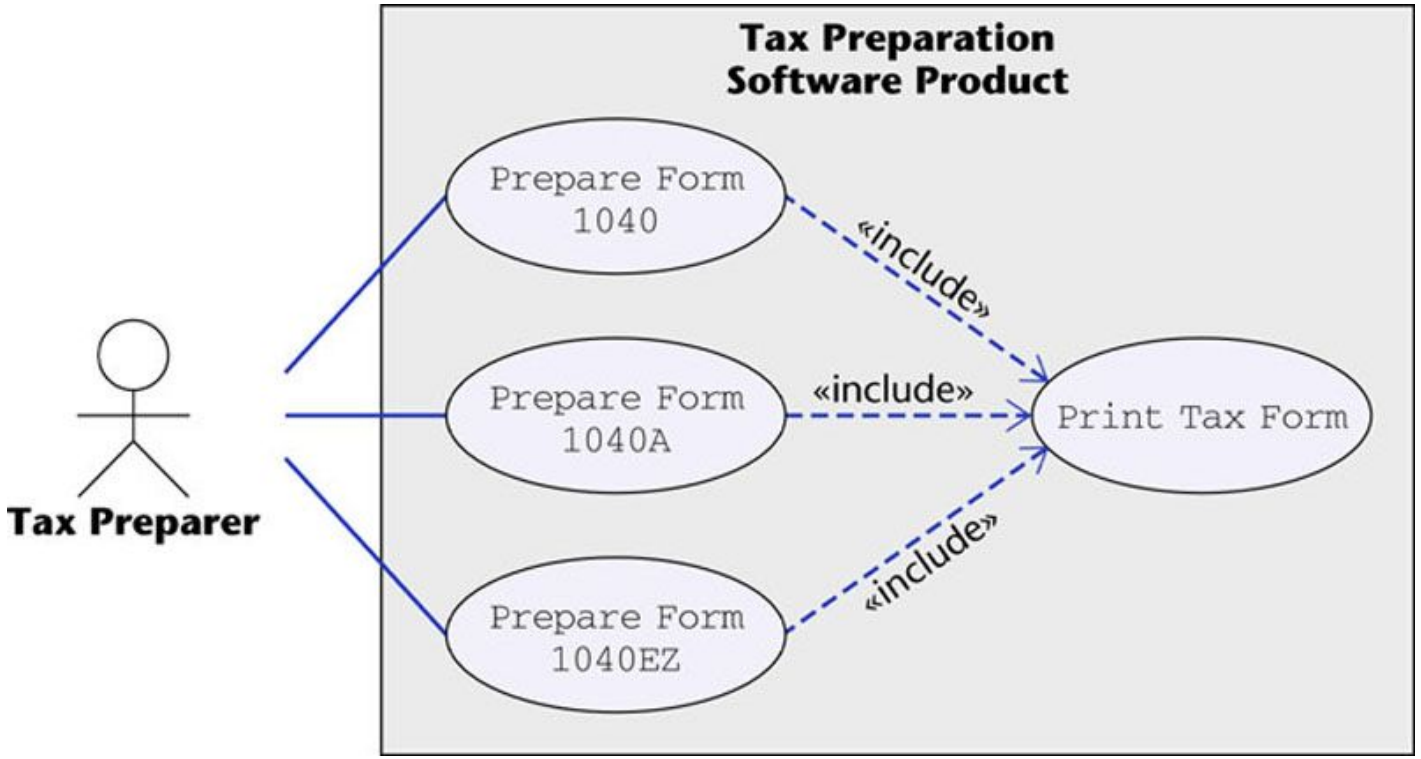
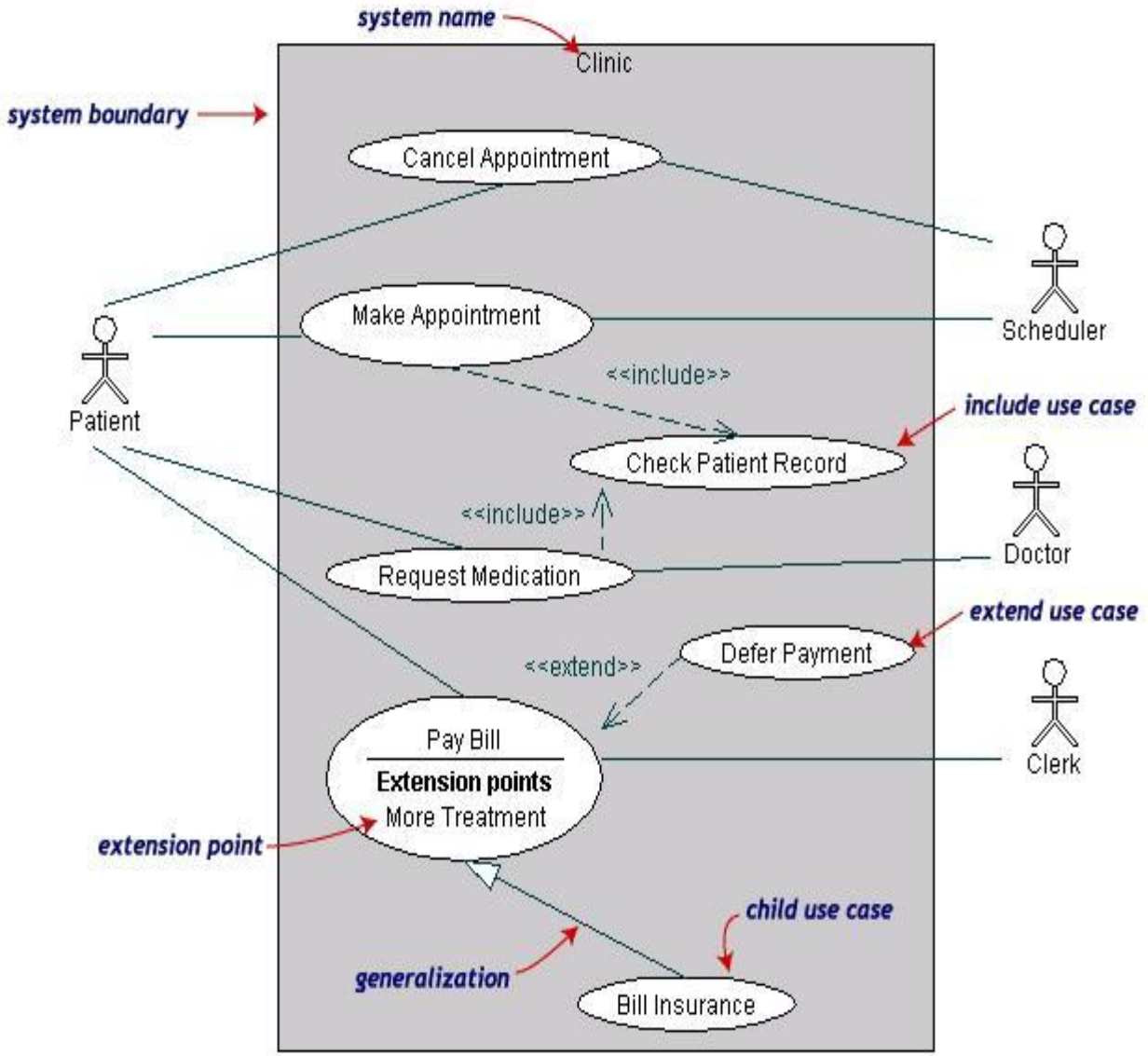


Figure 16.12

Use-Case Diagrams

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



(TogetherSoft, Inc)

Lab Practical 4: Use Case Diagram

- Group 1: E-commerce company
- Group 2: Online transport system
- Group 3: Online social media and social networking service company
- Group 4: Global online marketplace
- Group 5: Online marketplace and hospitality service

Norms

- All students will work in teams of 5.
- There are 5 systems available which can be alloted to students in sequential manner.
- The students need to make a Use Case Diagram.
- Assumptions regarding the systems are allowed.

Group 1: E-commerce company eg Flipkart etc.

Group 2: Online transport system eg. OLA, UBER

Group 3: Online social media and social networking service company

Group 4: Global online marketplace eg OLX

Group 5: Online marketplace and hospitality service eg. Airbnb

- Static diagrams describe the state of a system from a variety of perspectives. A static diagram describes what a piece of the system is.
- A dynamic diagram describes what a portion of the system is doing.

Class Diagram

- Class diagram is a static diagram.
- It represents the **static view of an application**.
- Class diagram is not only used for **visualizing, describing, and documenting different aspects of a system** but also for **constructing executable code of the software application.**
- Class diagram **describes the attributes and operations of a class** and also the **constraints imposed on the system.**

- The class diagrams are widely used in the **modeling of object-oriented systems** because they are the only UML diagrams, which can be mapped directly with object-oriented languages.
- Class diagram **shows a collection of classes, interfaces, associations, collaborations, and constraints**.
- It is also **known as a structural diagram.**

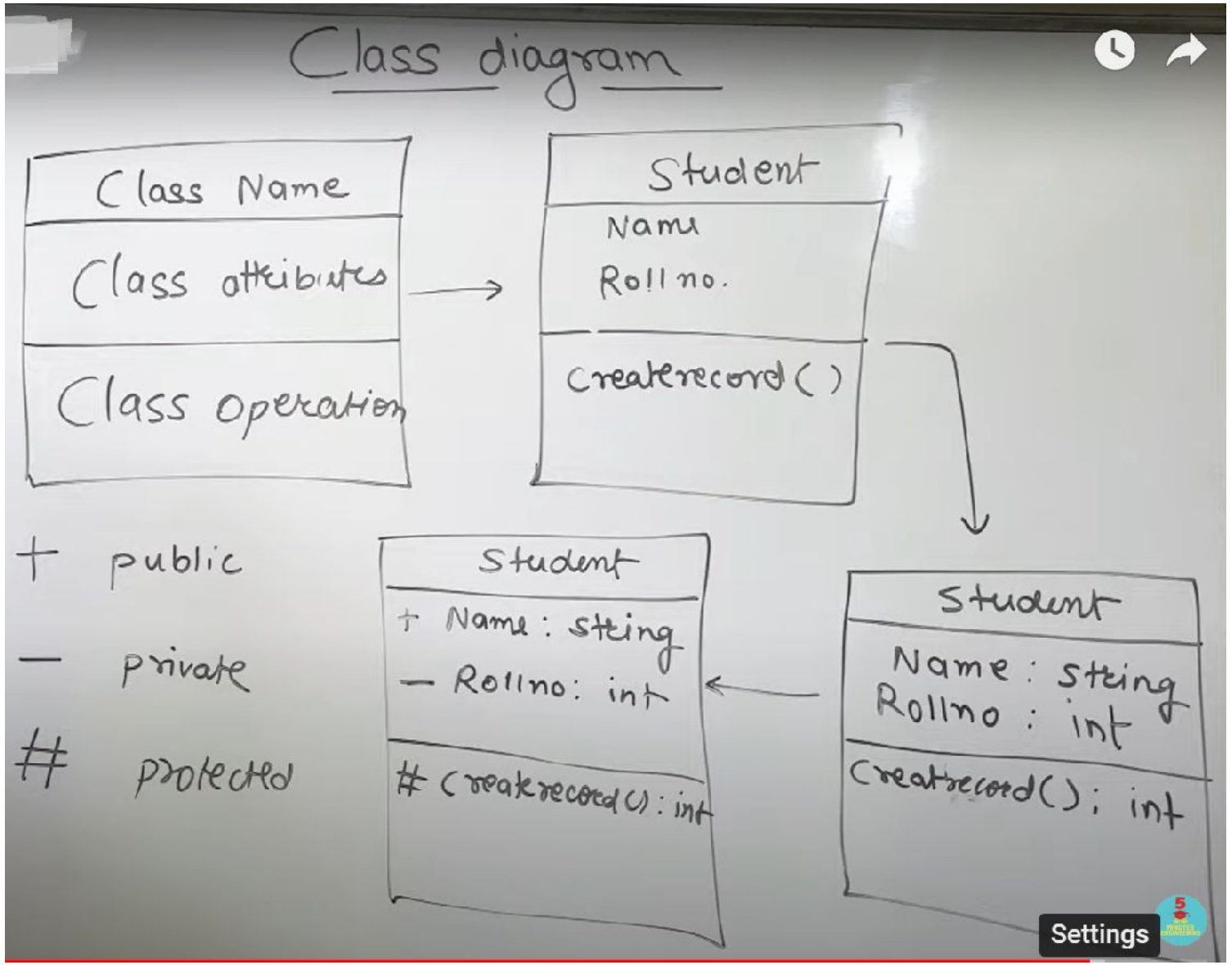
- The purpose of the class diagram can be summarized as −

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

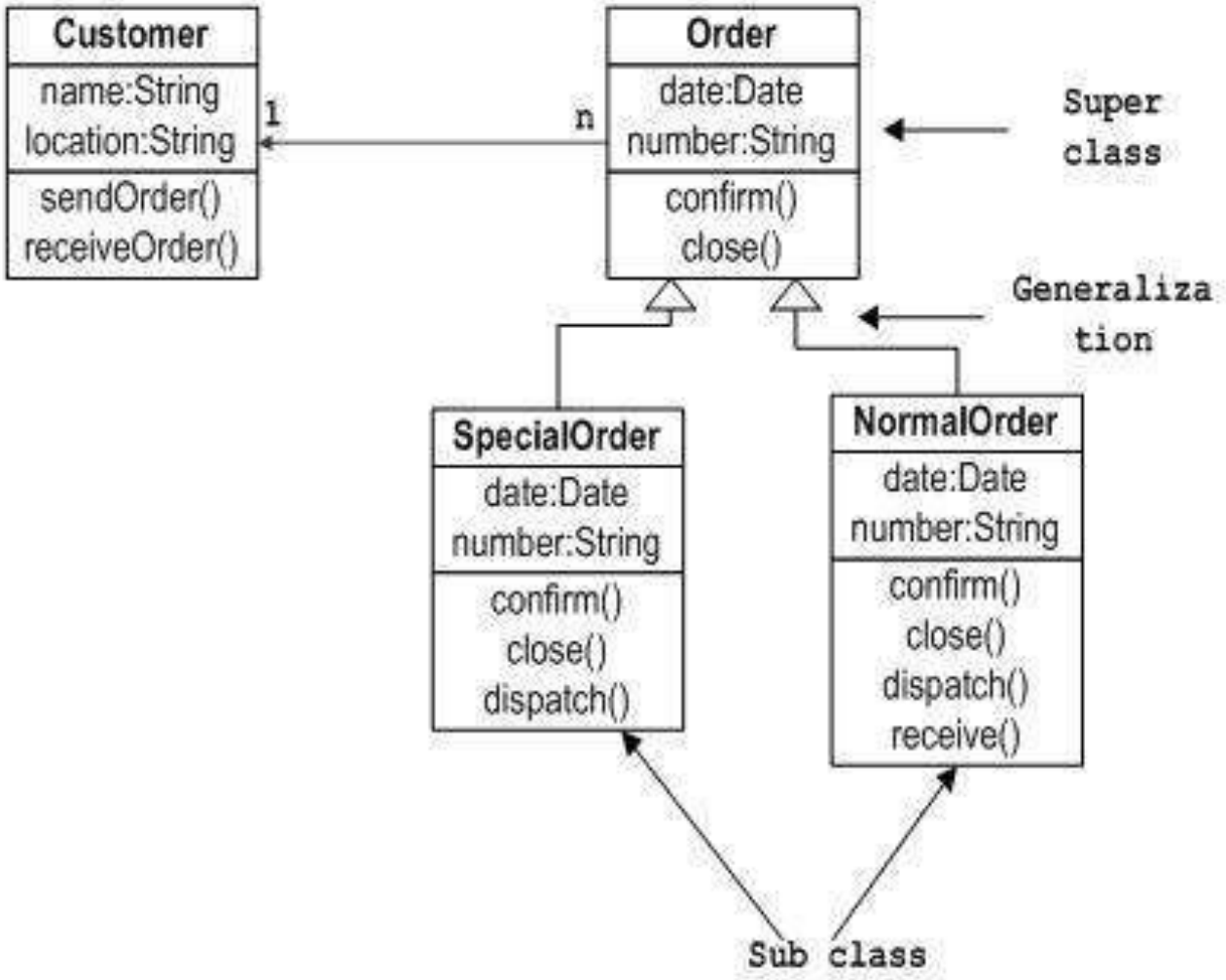
- The following points should be remembered while drawing a class diagram –
- The **name of the class diagram should be meaningful** to describe the aspect of the system.
- **Each element and their relationships should be identified in advance**.
- **Responsibility (attributes and methods) of each class should be clearly identified**

- For each class, **minimum number of properties should be specified**, as unnecessary properties will make the diagram complicated.
- **Use notes whenever required to describe some aspect of the diagram**. At the end of the drawing it should be understandable to the developer/coder.
- Finally, **before making the final version, the diagram should be drawn on plain paper and reworked** as many times as possible to make it correct.

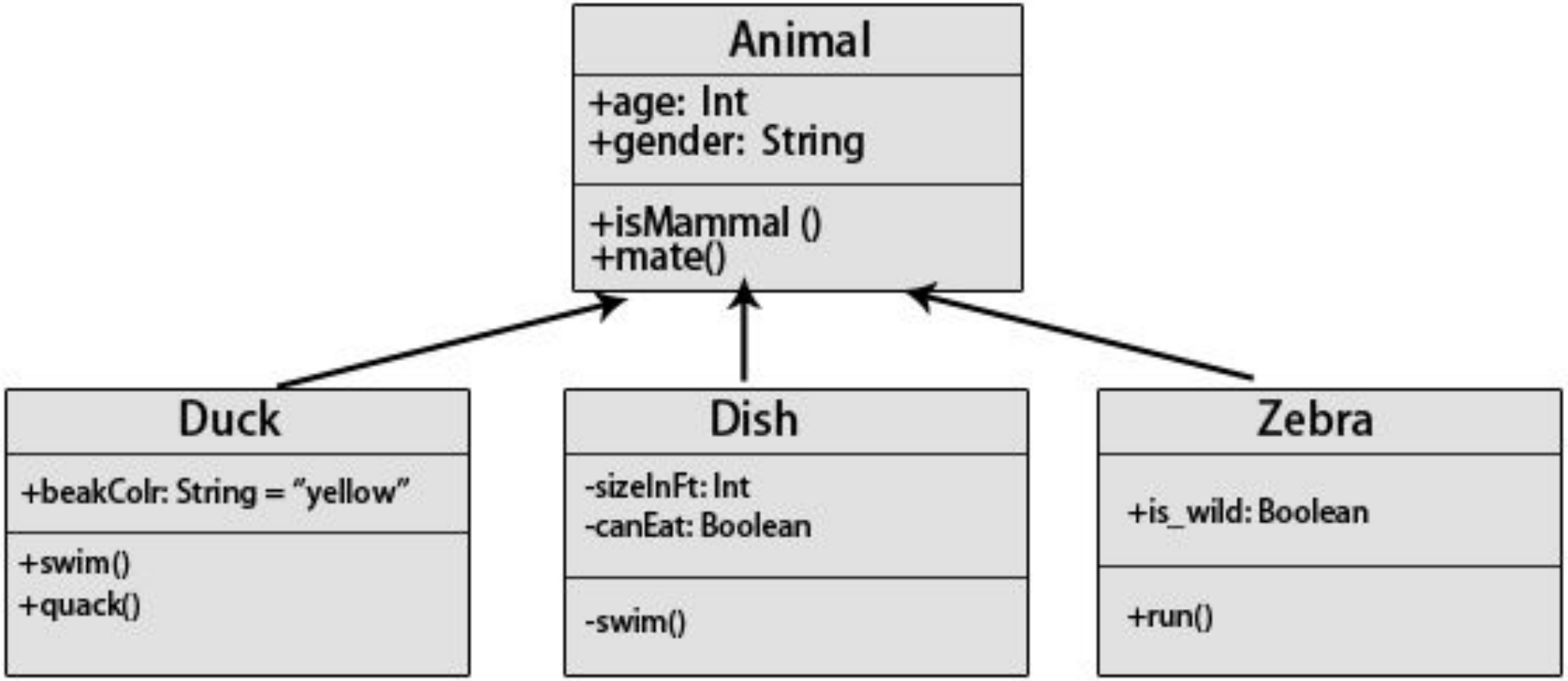
- The following diagram is an example of an Order System of an application. It describes a particular aspect of the entire application.
- First of all, **Order and Customer are identified as the two elements of the system.** They have a one-to-many relationship because a customer can have multiple orders.
- **Order class is an abstract class** and it has **two concrete classes (inheritance relationship) SpecialOrder and NormalOrder**.
- The **two inherited classes have all the properties as the Order class**. In addition, they have additional functions like **dispatch () and receive ()**.


Class diagram
Class Name
Class attributes
Class operation
Student
Name
Rollno.
Createrecord()
+ public
— private
protected
Student
+ Name : steing
— Rollno : int
Createrecord() : int
Student
Name : steing
Rollno : int
Createrecord() ; int
Settings

Sample Class Diagram



Class Diagram



Association

- Between two other classes in an association relationship, an association class forms a part of it. Additional information about the relationship could be obtained by attaching the association relationship with the association class. Various operations, attributes, etc., are present in the association class.
- The below diagram shows an association between bank and account.

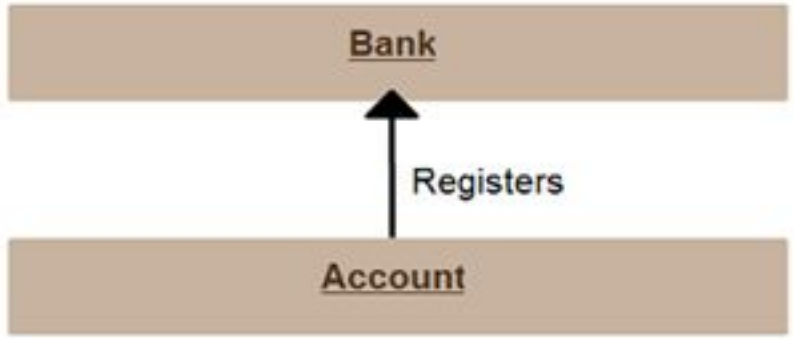
Multiplicity

The number of elements or cardinality could be defined by multiplicity. It is one of the most misunderstood relationships which describes the number of instances allowed for a particular element by providing an inclusive non-negative integers interval. It has both lower and upper bound. For example, **a bank would have many accounts registered to it. Thus near the account class, a star sign is present.**

Directed Association

This is a **one-directional relationship in a class diagram that ensures the flow of control from one to another classifier.** The navigability is specified by one of the association ends. The relationship between two classifiers could be described by naming any association. An arrow indicates the direction of navigation.

The below example shows an arrowhead relationship between the container and the contained.



Aggregation

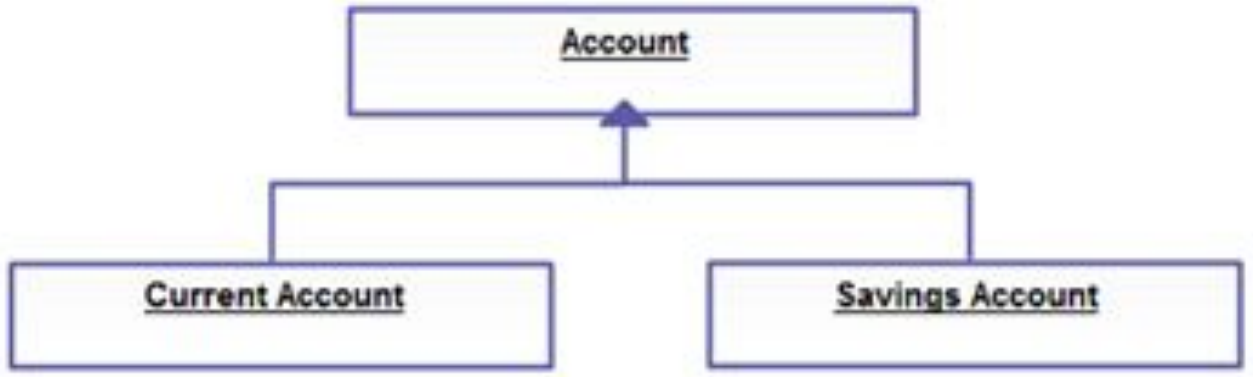
In this type of relationship, a **more complex object is created by assembling different objects together**. The interaction within the different groups of objects is defined by Aggregation. **The integrity of the objects is protected, and the response of the assembled objects is decided by the control object**. In aggregation, the classes nurture the ‘has a relationship.

Composition

It is a form of aggregation which represents the **whole-part relationship**. Here, the part classifier lifetime is dependent on the whole classifier lifetime. In a class, a strong life-cycle is represented by the composition relationship. There is usually a one-direction flow of data here. It is generally indicated by a solid line.

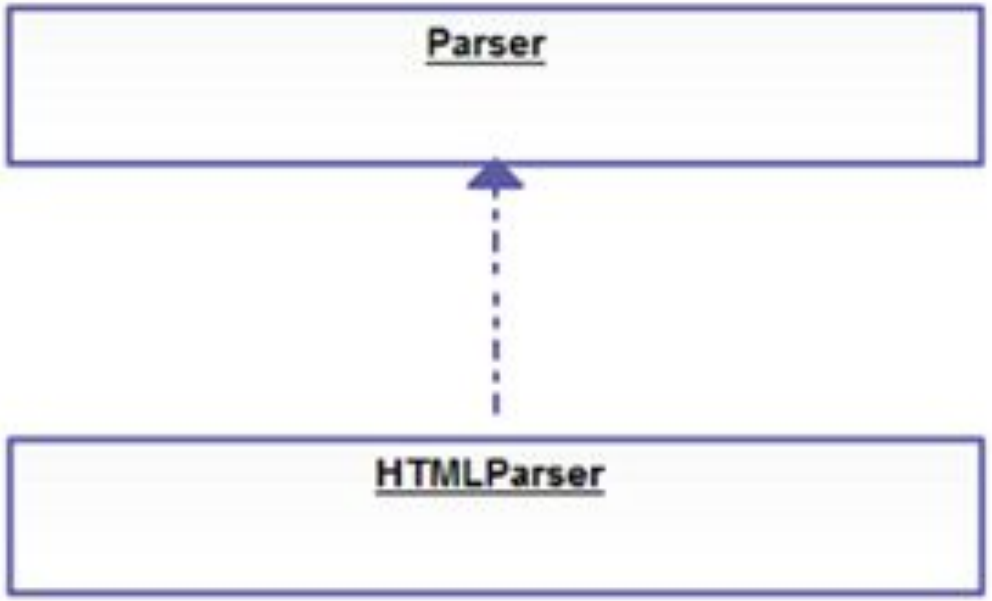
Generalization

- In this kind of relationship, the **child model is based on the parent model**. The relationship is used to describe various use-case diagrams and ensures that the child class receives the properties present in the parent. The child model could reuse the attributes of the parent model with the help of the generalization relationship.

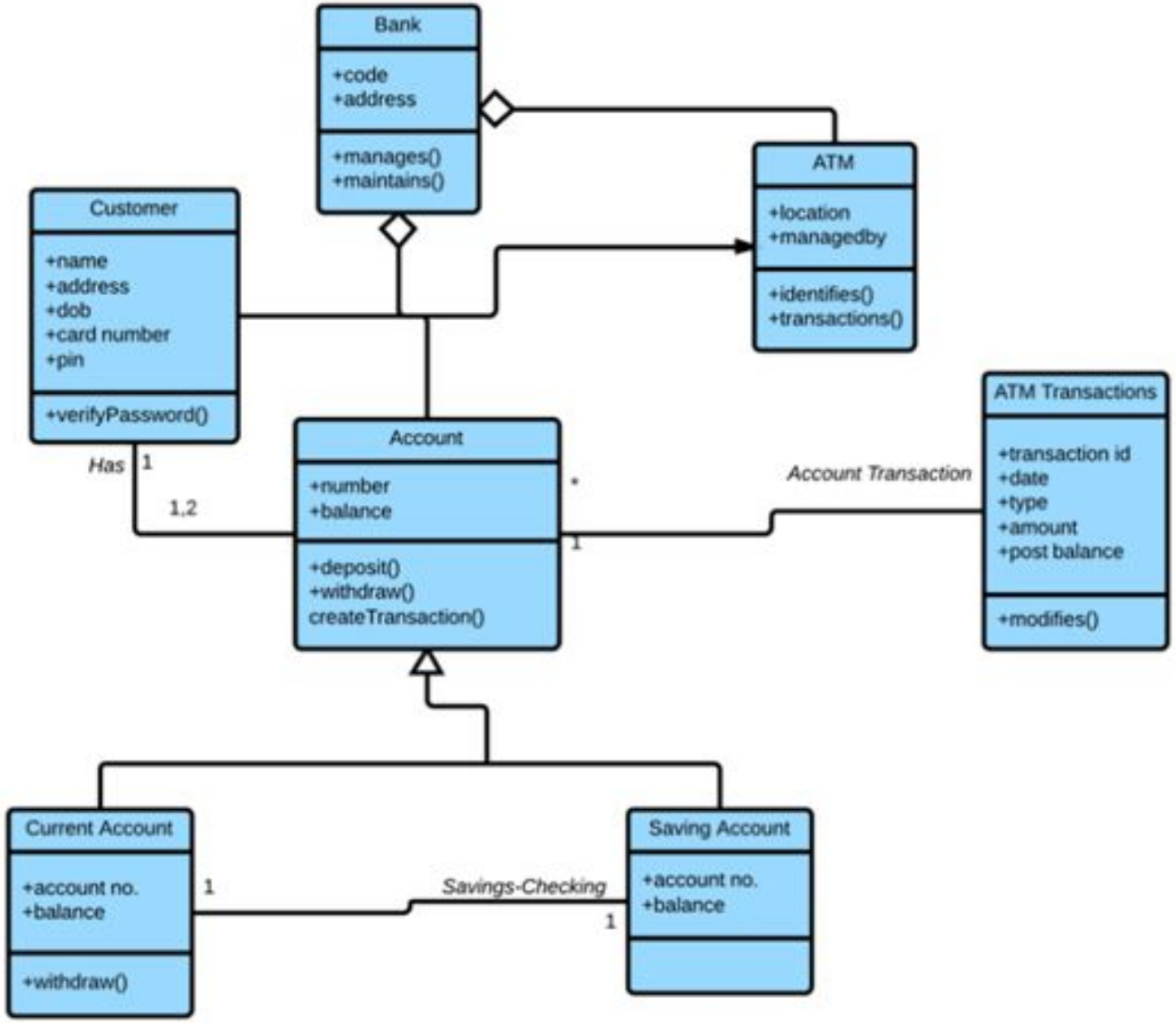


Realization

- **The behavior of one model element is realized by the specified behavior of another model element**. This type of relationship doesn't have any names.



- In a nutshell it can be said, class diagrams are used for –
- Describing the static view of the system.
- Showing the collaboration among the elements of the static view.
- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.


Bank
+code
+address
+manages()
+maintains()
ATM
+location
+managedby
+identifies()
+transactions()
Customer
+name
+address
+dob
+card number
+pin
+verifyPassword()
Has
1
1,2
Account
+number
+balance
+deposit()
+withdraw()
createTransaction()
*
1
Account Transaction
ATM Transactions
+transaction id
+date
+type
+amount
+post balance
+modifies()
Current Account
+account no.
+balance
+withdraw()
1
Savings-Checking
1
Saving Account
+account no.
+balance

Object diagram

- Object diagrams are **derived from class diagrams** so object diagrams are dependent upon class diagrams.
- Object diagrams **represent an instance of a class diagram**.
- The **basic concepts are similar for class diagrams and object diagrams**.
- Object diagrams also **represent the static view of a system** but this **static view is a snapshot of the system at a particular m**oment

- The purpose of a diagram should be understood clearly to implement it practically.
- The purposes of object diagrams are similar to class diagrams.
- The difference is that a **class diagram represents an abstract model** consisting of classes and their relationships.
- However, an **object diagram represents an instance at a particular moment, which is concrete in nature.**
- It means the **object diagram is closer to the actual system behavior**.
- The **purpose is to capture the static view of a system at a particular moment**.

- The purpose of the object diagram can be summarized as −
- Forward and reverse engineering.
- Object relationships of a system
- Static view of an interaction.
- Understand object behaviour and their relationship from practical perspective

How to Draw an Object Diagram?

- First, analyze the system and decide which instances have important data and association.
- Second, consider only those instances, which will cover the functionality.
- Third, make some optimization as the number of instances are unlimited.

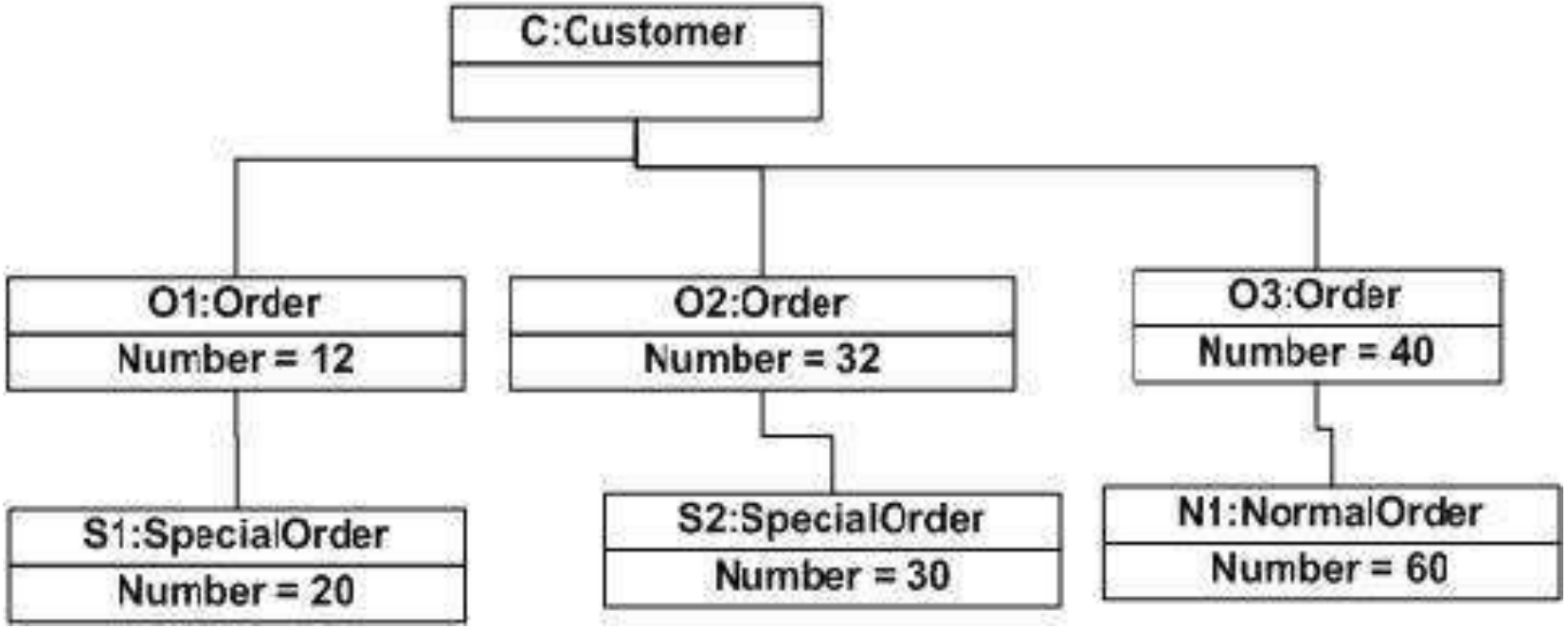
- Before drawing an object diagram, the following things should be remembered and understood clearly −

- Object diagrams consist of objects.
- The link in object diagram is used to connect objects.
- Objects and links are the two elements used to construct an object diagram

- After this, the following things are to be decided before starting the construction of the diagram −

- The object diagram should have a **meaningful name** to indicate its purpose.
- The **most important elements** are to be identified.
- The **association among objects** should be clarified.
- **Values of different elements need to be captured** to include in the object diagram.
- **Add proper notes at points** where more clarity is required.

Object diagram of an order management system



- The following diagram is an instance of the system at a particular time of purchase. It has the following objects.

- Customer
- Order
- SpecialOrder
- NormalOrder

- Now the customer object (C) is associated with three order objects (O1, O2, and O3).
- These order objects are associated with special order and normal order objects (S1, S2, and N1).
- The customer has the following three orders with different numbers (12, 32 and 40) for the particular time considered.

- The customer can increase the number of orders in future and in that scenario the object diagram will reflect that.
- If order, special order, and normal order objects are observed then you will find that they have some values.
- For orders, the values are 12, 32, and 40 which implies that the objects have these values for a particular moment (here the particular time when the purchase is made is considered as the moment) when the instance is captured

- The same is true for special order and normal order objects which have number of orders as 20, 30, and 60.
- If a different time of purchase is considered, then these values will change accordingly.

Where to Use Object Diagrams?

- Object diagrams can be imagined as the snapshot of a running system at a particular moment. Let us consider an example of a running train
- Now, if you take a snap of the running train then you will find a static picture of it having the following −

- A particular state which is running.
- A particular number of passengers. which will change if the snap is taken in a different time

- Here, we can imagine the snap of the running train is an object having the above values. And this is true for any real-life simple or complex system.

- In a nutshell, it can be said that object diagrams are used for −

- Making the prototype of a system.
- Reverse engineering.
- Modeling complex data structures.
- Understanding the system from practical perspective.

Component diagram

- Component diagrams are different in terms of nature and behavior.
- Component diagrams are **used to model the physical aspects of a system**.
- Physical aspects are the elements such as **executables, libraries, files, documents, etc. which reside in a node.**
- Component diagrams are used to **visualize the organization and relationships among components** in a system.
- These diagrams are also used to **make executable systems**

- The purpose of the component diagram can be summarized as −

- Visualize the components of a system.
- Construct executables by using forward and reverse engineering.
- Describe the organization and relationships of the components.

How to Draw a Component Diagram?

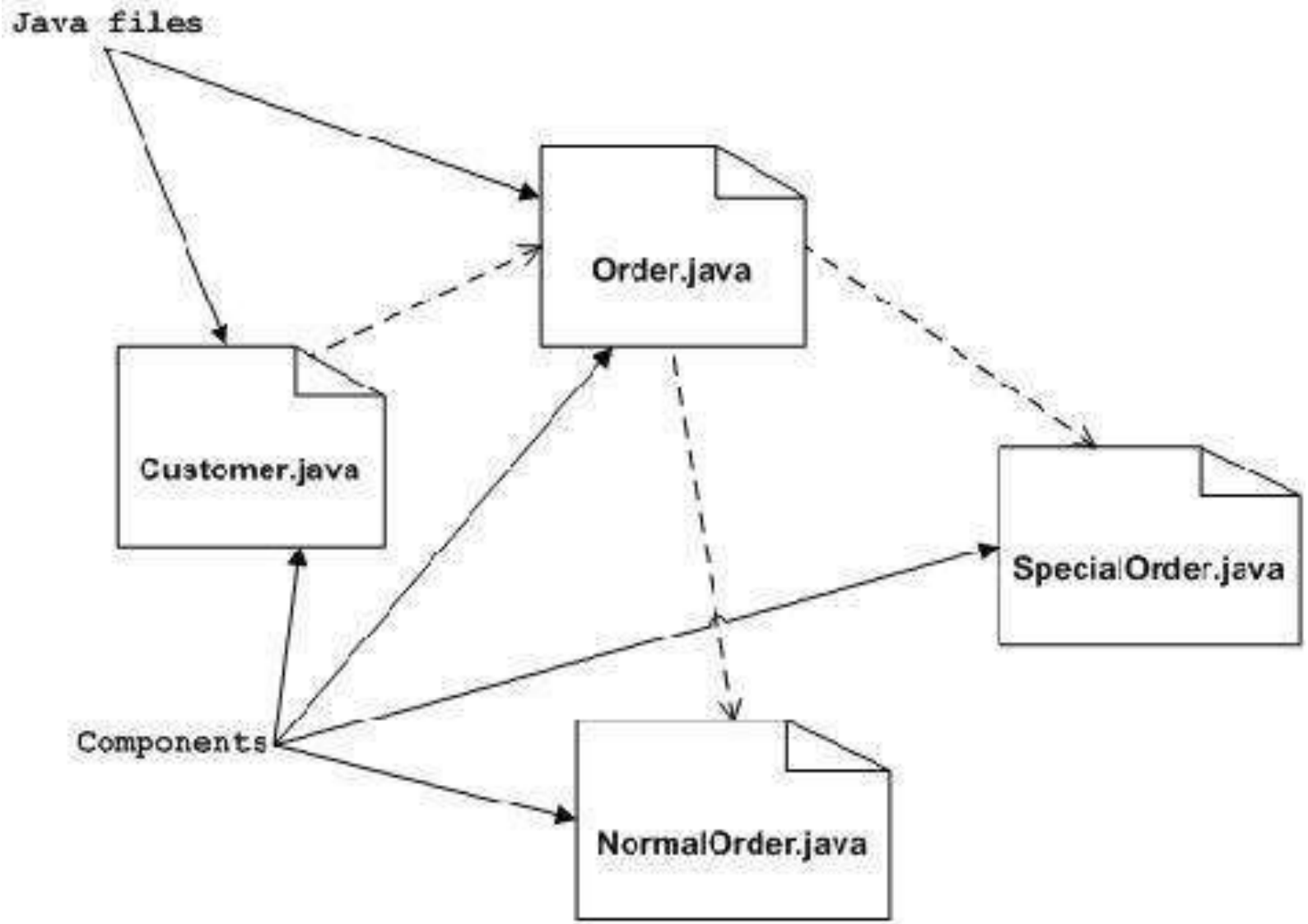
- Before drawing a component diagram, the following artifacts are to be identified clearly −

- **Files used** in the system.
- **Libraries and other artifacts** relevant to the application.
- **Relationships among the artifacts**.

- After identifying the artifacts, the following points need to be kept in mind.

- Use a **meaningful name to identify the component** for which the diagram is to be drawn.
- **Prepare a mental layout** before producing the using tools.
- **Use notes for clarifying important points**.

Component diagram of an order management system

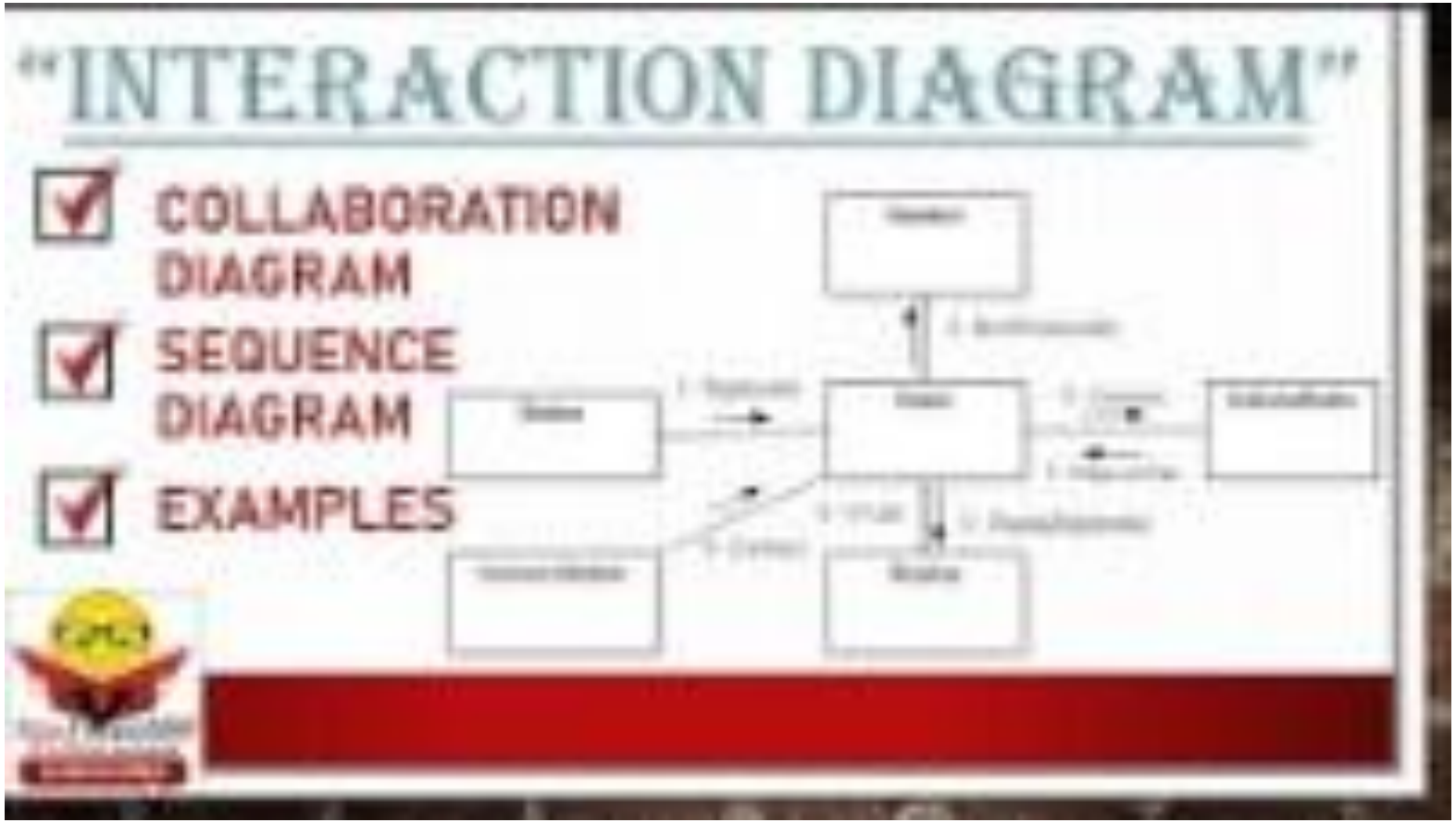


- Following is a component diagram for order management system.
- Here, the artifacts are files. The diagram shows the files in the application and their relationships.
- In actual, the component diagram also contains dlls, libraries, folders, etc.
- In the following diagram, four files are identified and their relationships are produced.

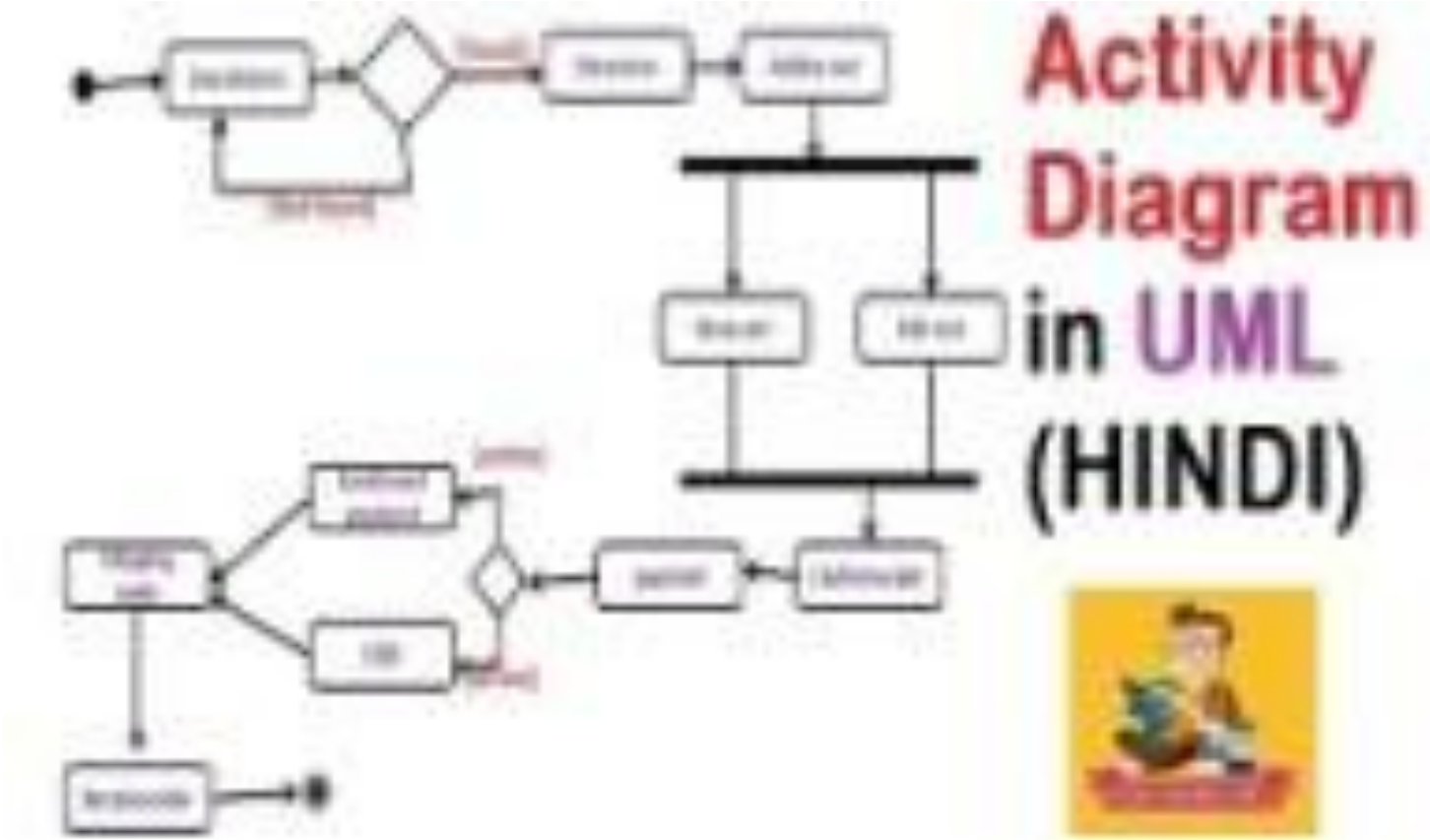
- Component diagrams can be used to −

- Model the components of a system.
- Model the database schema.
- Model the executables of an application.
- Model the system's source code.

Interaction diagrams



Activity Diagram



References

S No.	Link	Description
1	Use case Diagram (https://www.youtube.com/watch?v=zid-MVo7M-E&t=640s)	Details and making of Use Case Diagrams
2	Class Diagram (https://www.youtube.com/watch?v=xiUFTLIU-lw)	Details and making of Class Diagrams
3	Sequence Diagram (https://www.youtube.com/watch?v=pCK6prSq8aw&t=1s)	Details and making of Sequence Diagrams
4	Activity Diagrams (https://www.youtube.com/watch?v=XFTAIj2N2Lc)	Details and making of Activity Diagrams

Thank You