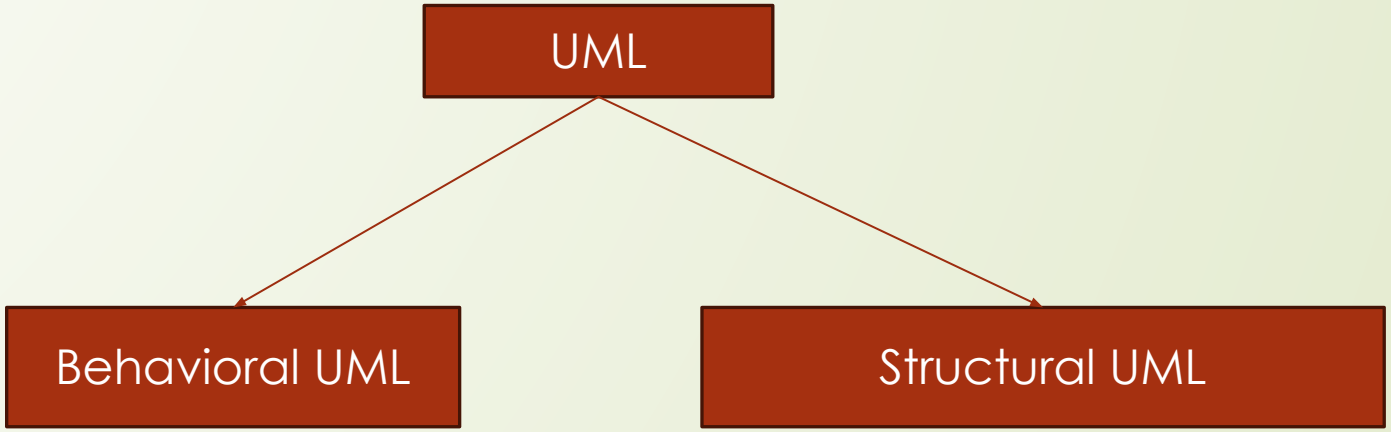
UML

UNIT 2


UML
Behavioral UML
Structural UML

What is UML?

- The Unified Modelling Language is a standard graphical language for modelling object oriented software
  - At the end of the 1980s and the beginning of 1990s, the first object-oriented development processes appeared
  - The proliferation of methods and notations tended to cause considerable confusion
  - Two important methodologists Rumbaugh and Booch decided to merge their approaches in 1994.
    - They worked together at the Rational Software Corporation
  - In 1995, another methodologist, Jacobson, joined the team
    - His work focused on use cases
  - In 1997 the Object Management Group (OMG) started the process of UML standardization

UML diagrams

- Class diagrams
  - describe classes and their relationships
- Interaction diagrams
  - show the behaviour of systems in terms of how objects interact with each other
- State diagrams and activity diagrams
  - show how systems behave internally
- Component and deployment diagrams
  - show how the various components of systems are arranged logically and physically

UML features

- It has detailed *semantics*
- It has *extension* mechanisms
- It has an associated textual language
  - *Object Constraint Language* (OCL)

- The objective of UML is to assist in software development
  - It is not a *methodology*

What constitutes a good model?

- A model should
  - use a standard notation
  - be understandable by clients and users
  - lead software engineers to have insights about the system
  - provide abstraction

- Models are used:
  - to help create designs
  - to permit analysis and review of those designs.
  - as the core documentation describing the system.

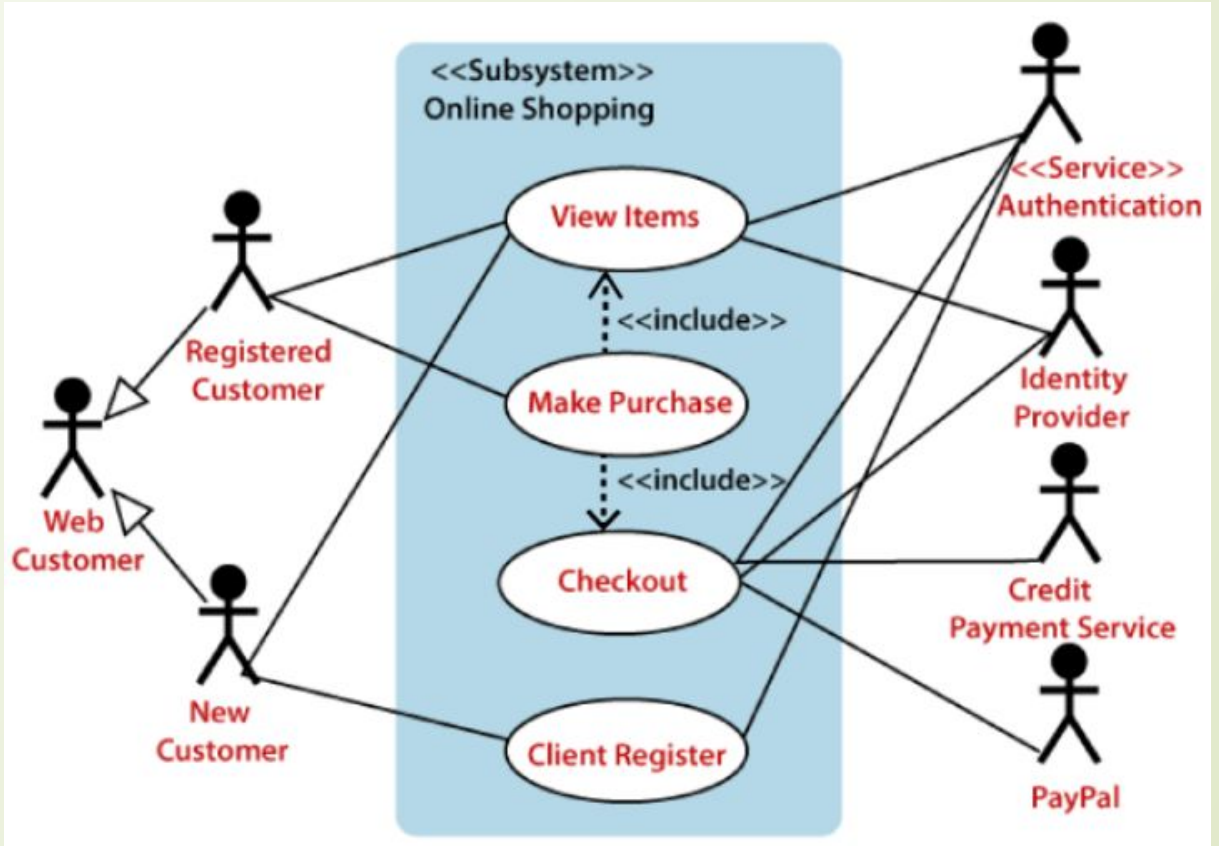
UML Use Case Diagram

- A use case diagram is used to represent the dynamic behavior of a system. It encapsulates the system's functionality by incorporating use cases, actors, and their relationships. It models the tasks, services, and functions required by a system/subsystem of an application.
- Purpose of Use Case Diagrams
- The main purpose of a use case diagram is to portray the dynamic aspect of a system. It accumulates the system's requirement, which includes both internal as well as external influences. It invokes persons, use cases, and several things that invoke the actors and elements accountable for the implementation of use case diagrams.
- Following are the purposes of a use case diagram given below:

1. It gathers the system's needs.
2. It depicts the external view of the system.
3. It recognizes the internal as well as external factors that influence the system.
4. It represents the interaction between the actors.

Example of a Use Case Diagram

- A use case diagram depicting the Online Shopping website is given below.
- Here the Web Customer actor makes use of any online shopping website to purchase online. The top-level uses are as follows; View Items, Make Purchase, Checkout, Client Register. The **View Items** use case is utilized by the customer who searches and view products. The **Client Register** use case allows the customer to register itself with the website for availing gift vouchers, coupons, or getting a private sale invitation. It is to be noted that the **Checkout** is an included use case, which is part of **Making Purchase,** and it is not available by itself.

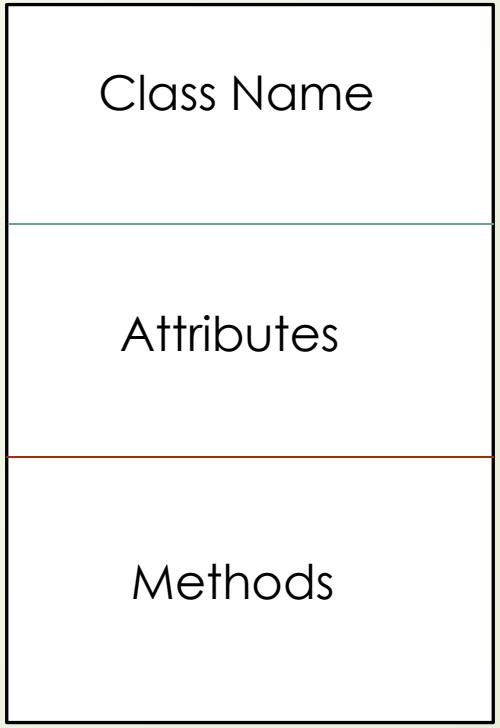


UML Class Diagrams

- The class diagram depicts a static view of an application.
- It represents the types of objects residing in the system and the relationships between them. A class consists of its objects, and also it may inherit from other classes.
- A class diagram is used to visualize, describe, document various different aspects of the system, and also construct executable software code.
- Purpose of Class Diagrams

1. It analyses and designs a static view of an application.
2. It describes the major responsibilities of a system.
3. It is a base for component and deployment diagrams.
4. It incorporates forward and reverse engineering.

Vital components of a Class Diagram



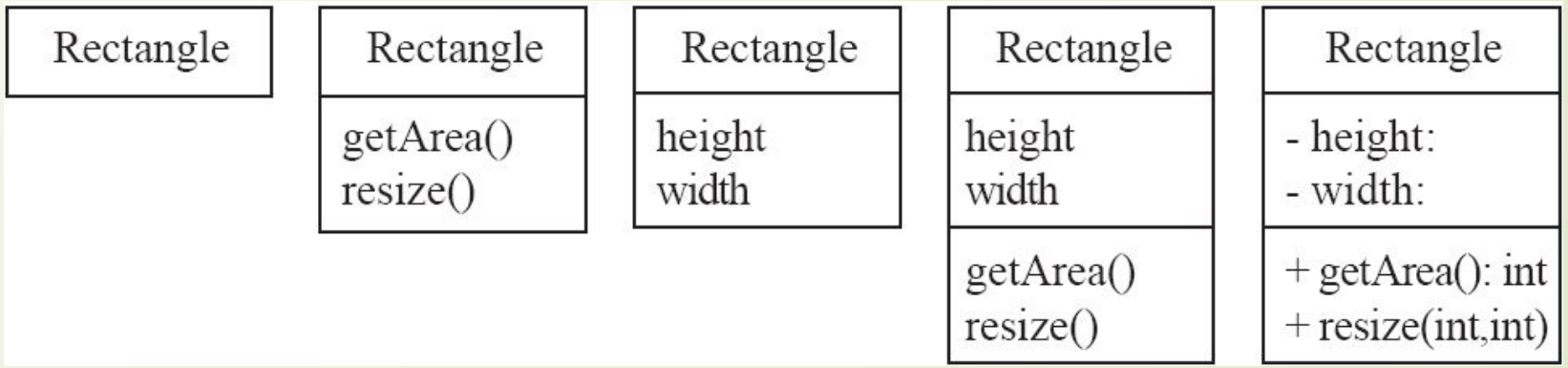
Essentials of UML Class Diagrams

- *The main symbols shown on class diagrams are:*
  - *Classes*
    - represent the types of data themselves
  - *Associations*
    - represent linkages between instances of classes
  - *Attributes*
    - are simple data found in classes and their instances
  - *Operations*
    - represent the functions performed by the classes and their instances
  - *Generalizations*
    - group classes into inheritance hierarchies

Classes

- A class is simply represented as a box with the name of the class inside
  - The diagram may also show the attributes and operations
  - The complete signature of an operation is:

    operationName(parameterName: parameterType …): returnType

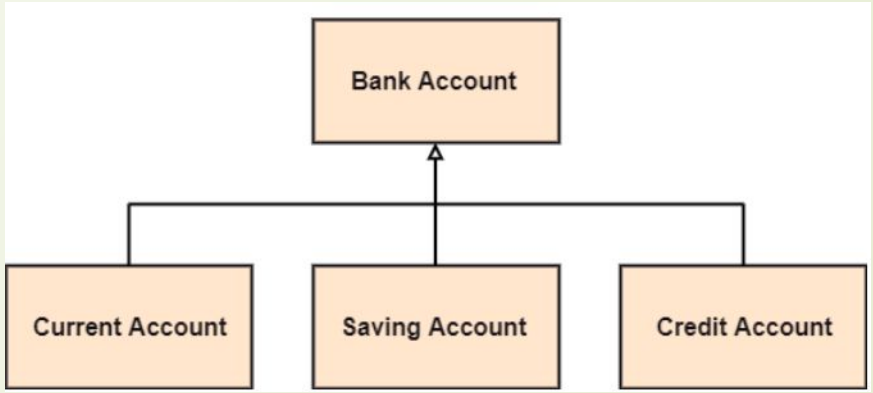


Relationships

- In UML, relationships are of three types:

- **Dependency:** A dependency is a semantic relationship between two or more classes where a change in one class cause changes in another class. It forms a weaker relationship.
  In the following example, Student_Name is dependent on the Student_Id.

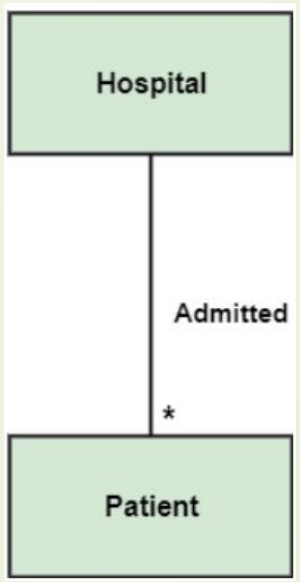
- **Generalization:** A generalization is a relationship between a parent class (superclass) and a child class (subclass). In this, the child class is inherited from the parent class.
  For example, The Current Account, Saving Account, and Credit Account are the generalized form of Bank Account.



- **Association:** It describes a static or physical connection between two or more objects. It depicts how many objects are there in the relationship.
  For example, a department is associated with the college.

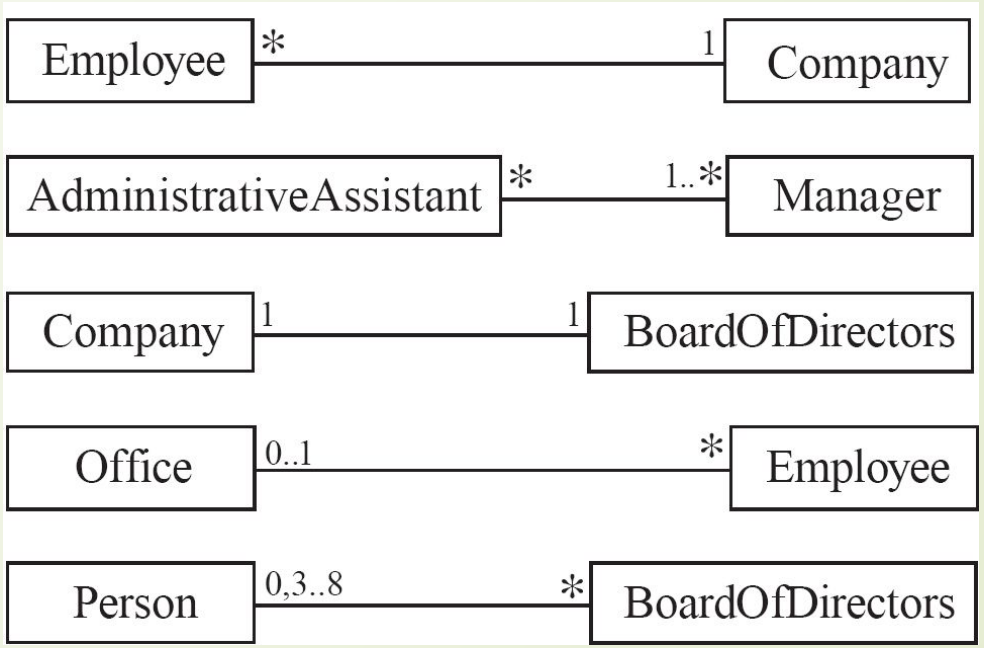
Multiplicity

- It defines a specific range of allowable instances of attributes. In case if a range is not specified, one is considered as a default multiplicity.
- For example, multiple patients are admitted to one hospital.



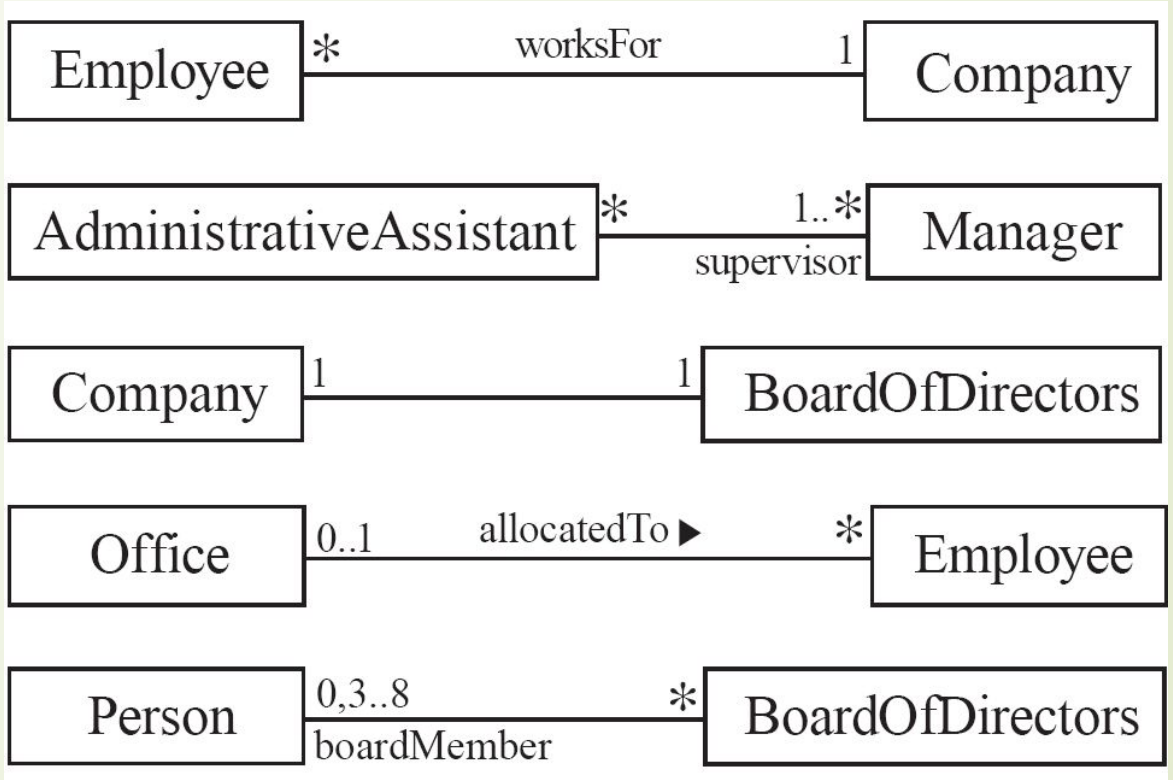
Association and Multiplicity

- Symbols indicating *multiplicity* are shown at each end of the association



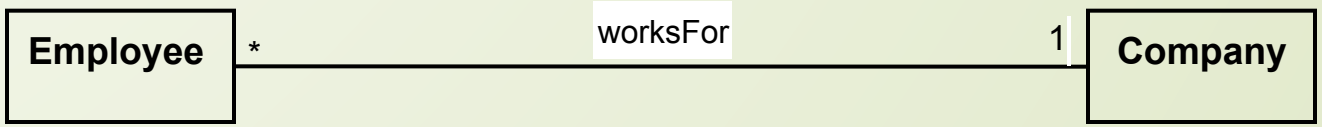
Labelling associations

- Each association can be labelled, to make explicit the nature of the association



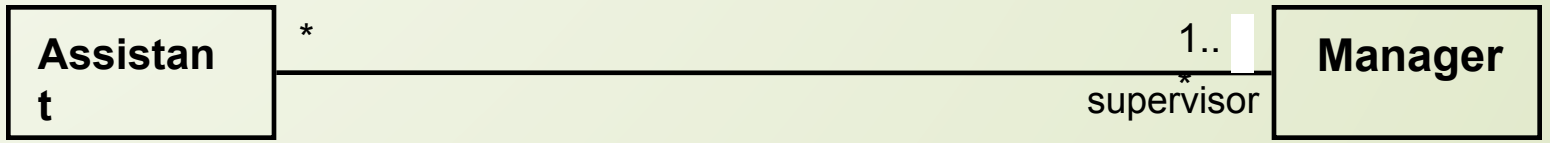
Analyzing and validating associations

- **Many-to-one**
  - A company has many employees,
  - An employee can only work for one company.
    - This company will not store data about the moonlighting activities of employees!
  - A company can have zero employees
    - E.g. a ‘shell’ company
  - It is not possible to be an employee unless you work for a company



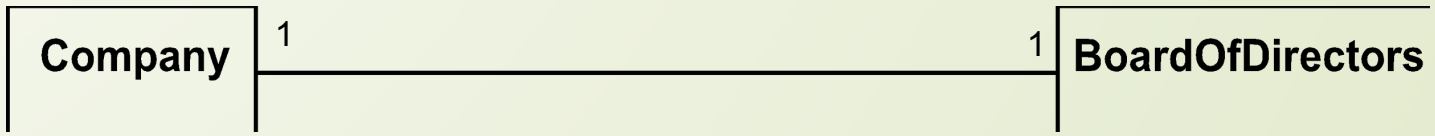
Analyzing and validating associations

- **Many-to-many**
  - An assistant can work for many managers
  - A manager can have many assistants
  - Assistants can work in pools
  - Managers can have a group of assistants
  - Some managers might have zero assistants.
  - Is it possible for an assistant to have, perhaps temporarily, zero managers?



Analyzing and validating associations

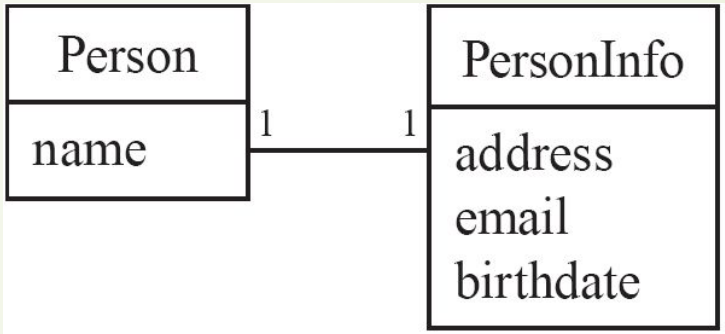
- **One-to-one**
  - For each company, there is exactly one board of directors
  - A board is the board of only one company
  - A company must always have a board
  - A board must always be of some company



Analyzing and validating associations

- Avoid unnecessary one-to-one associations

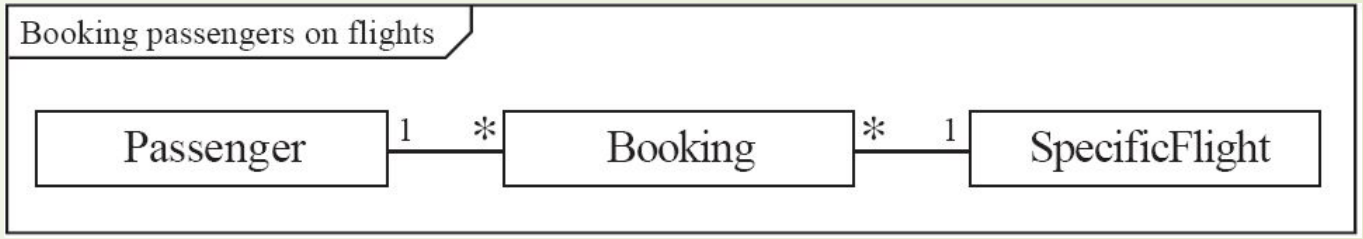
- Avoid this



do this

A more complex example

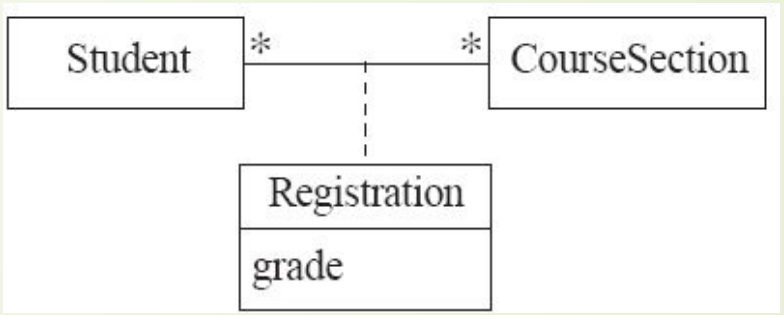
- A booking is always for exactly one passenger
  - no booking with zero passengers
  - a booking could *never* involve more than one passenger.
- A Passenger can have any number of Bookings
  - a passenger could have no bookings at all
  - a passenger could have more than one booking

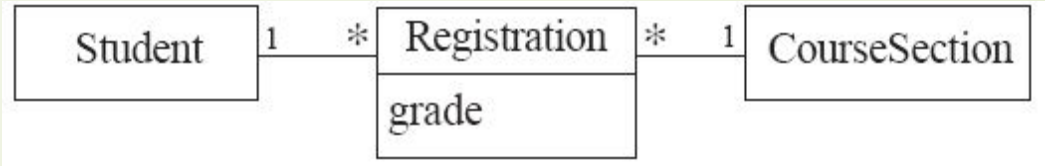


- The *frame* around this diagram is an optional feature that any UML 2.0 may possess.

Association classes

- Sometimes, an attribute that concerns two associated classes cannot be placed in either of the classes
- The following are equivalent





Directionality in associations

- Associations are by default *bi-directional*
- It is possible to limit the direction of an association by adding an arrow at one end

Aggregation & Composition

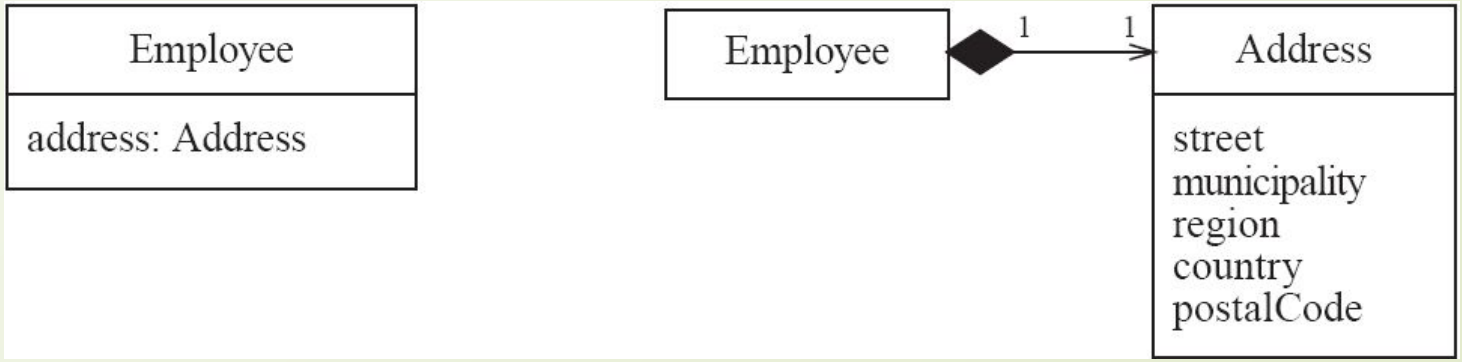
- An **aggregation** is a subset of association, which represents **has a relationship**. It is more specific then association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class.
- The company encompasses a number of employees, and even if one employee resigns, the company still exists.

- The **composition** is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the other part also gets discarded. It represents a **whole-part relationship**.
- A contact book consists of multiple contacts, and if you delete the contact book, all the contacts will be lost.

Composition

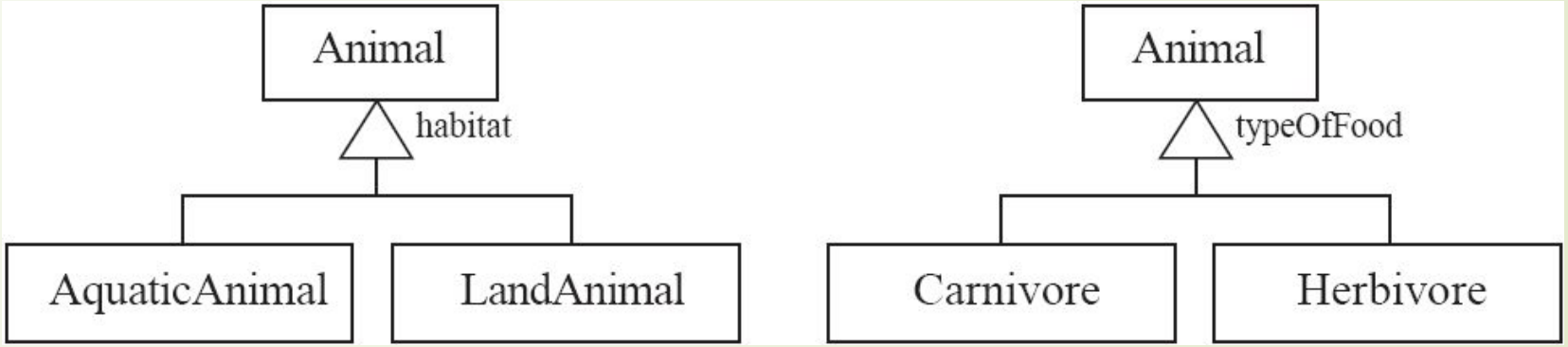
- A *composition* is a strong kind of aggregation
  - if the aggregate is destroyed, then the parts are destroyed as well

- Two alternatives for addresses



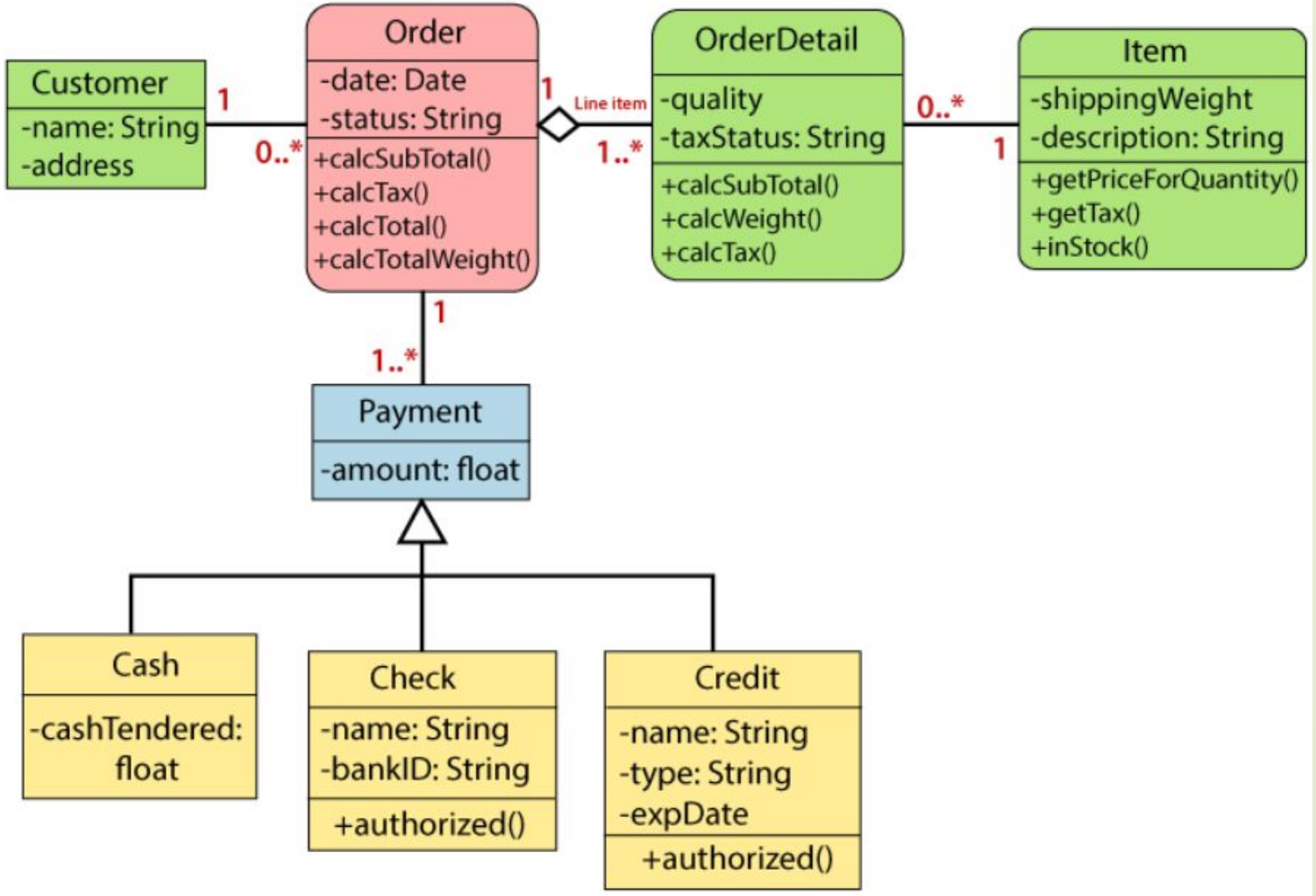
Generalization

- Specializing a superclass into two or more subclasses
  - A *generalization set* is a labeled group of generalizations with a common superclass
  - The label (sometimes called the *discriminator*) describes the criteria used in the specialization



Class Diagram Example

A class diagram describing the sales order system is given below.



Suggested sequence of activities

- Identify a first set of candidate **classes**
- Add **associations** and **attributes**
- Find **generalizations**
- List the main **responsibilities** of each class
- Decide on specific **operations**
- **Iterate** over the entire process until the model is satisfactory
  - Add or delete classes, associations, attributes, generalizations, responsibilities or operations
  - Identify interfaces
  - Apply design patterns

Don't be too disorganized. Don't be too rigid either.

Prototyping a class diagram on paper

- As you identify classes, you write their names on small cards
- As you identify attributes and responsibilities, you list them on the cards
  - If you cannot fit all the responsibilities on one card:
    - this suggests you should split the class into two related classes.
- Move the cards around on a whiteboard to arrange them into a class diagram.
- Draw lines among the cards to represent associations and generalizations.

UML State Transition diagram

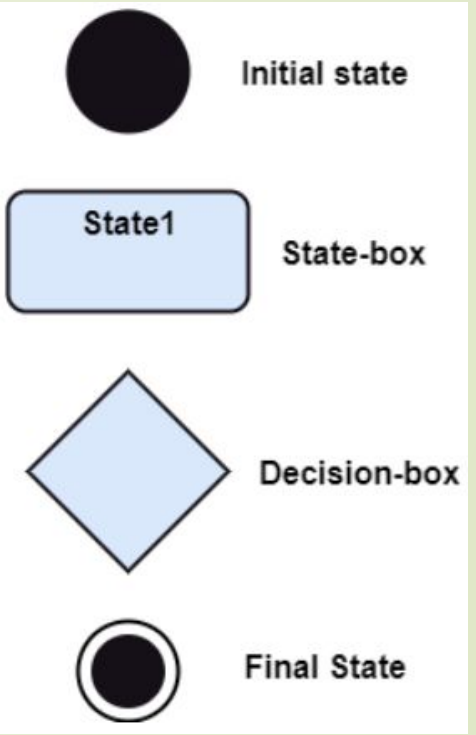
- The state transition diagram is also called the Statechart or, state machine diagram which shows the order of states underwent by an object within the system. It captures the software system's behavior.
- It models the behavior of a class, a subsystem, a package, and a complete system.
- It tends out to be an efficient way of modeling the interactions and collaborations in the external entities and the system. It models event-based systems to handle the state of an object. It also defines several distinct states of a component within the system. Each object/component has a specific state.
- Following are the types of a state machine diagram that are given below:

1. **Behavioral state machine**
   The behavioral state machine diagram records the behavior of an object within the system. It depicts an implementation of a particular entity. It models the behavior of the system.
2. **Protocol state machine**
   It captures the behavior of the protocol. The protocol state machine depicts the change in the state of the protocol and parallel changes within the system. But it does not portray the implementation of a particular component.

Notation of a State Machine Diagram

- Following are the notations of a state machine diagram enlisted below:

1. **initial state:** It defines the initial state (beginning) of a system, and it is represented by a black filled circle.
2. **Final state:** It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.
3. **Decision box:** It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.
4. **Transition:** A change of control from one state to another due to the occurrence of some event is termed as a transition. It is represented by an arrow labeled with an event due to which the change has ensued.
5. **State box:** It depicts the conditions or circumstances of a particular object of a class at a specific point of time. A rectangle with round corners is used to represent the state box.



Types of State

- The UML consist of three states:

1. **Simple state:** It does not constitute any substructure.
2. **Composite state:** It consists of nested states (substates), such that it does not contain more than one initial state and one final state. It can be nested to any level.
3. **Submachine state:** The submachine state is semantically identical to the composite state, but it can be reused.

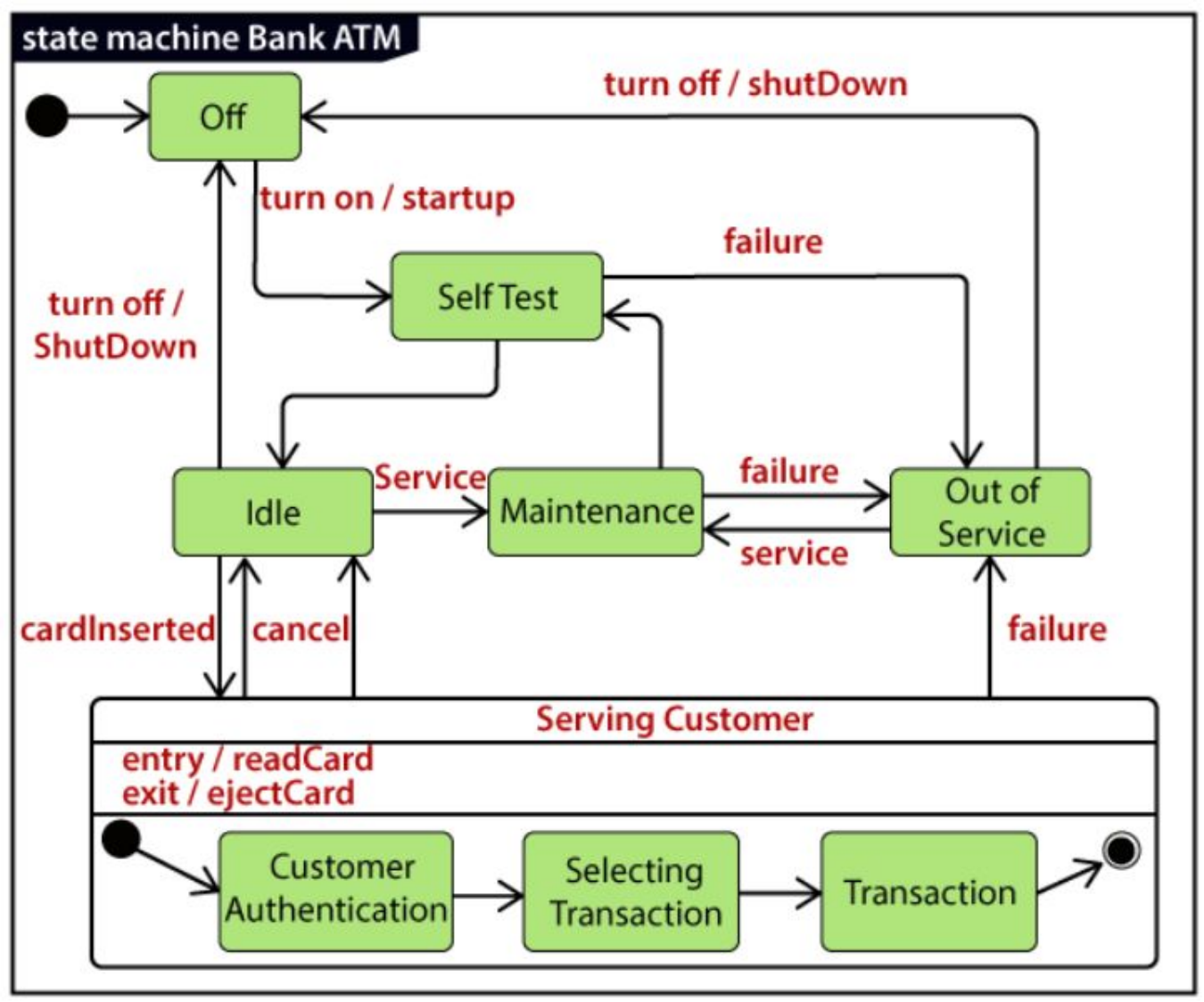
When to use a State Transition Diagram?

- The state machine diagram implements the real-world models as well as the object-oriented systems. It records the dynamic behavior of the system, which is used to differentiate between the dynamic and static behavior of a system.
- It portrays the changes underwent by an object from the start to the end. It basically envisions how triggering an event can cause a change within the system.
- State machine diagram is used for:

1. For modeling the object states of a system.
2. For modeling the reactive system as it consists of reactive objects.
3. For pinpointing the events responsible for state transitions.
4. For implementing forward and reverse engineering.

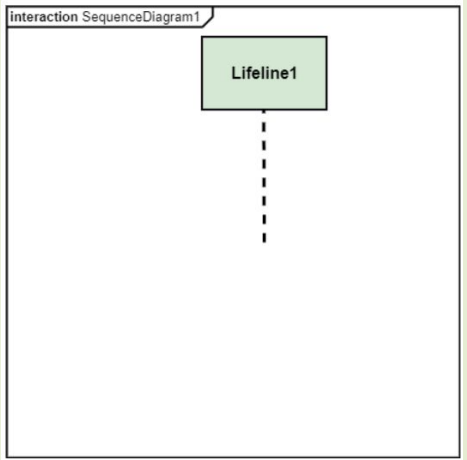
Example of a State Machine Diagram

- An example of a top-level state machine diagram showing Bank Automated Teller Machine (ATM) is given below.
- Initially, the ATM is turned off. After the power supply is turned on, the ATM starts performing the startup action and enters into the **Self Test** state. If the test fails, the ATM will enter into the **Out Of Service** state, or it will undergo **a triggerless transition** to the **Idle** state. This is the state where the customer waits for the interaction.
- Whenever the customer inserts the bank or credit card in the ATM's card reader, the ATM state changes from **Idle** to **Serving Customer**, the entry action **readCard** is performed after entering into **Serving Customer** state. Since the customer can cancel the transaction at any instant, so the transition from **Serving Customer** state back to the **Idle** state could be triggered by **cancel** event.
- Here the **Serving Customer** is a composite state with sequential substates that are **Customer Authentication, Selecting Transaction,** and **Transaction**.
- **Customer Authentication** and **Transaction** are the composite states itself is displayed by a hidden decomposition indication icon. After the transaction is finished, the **Serving Customer** encompasses a triggerless transition back to the **Idle** state. On leaving the state, it undergoes the exit action **ejectCard** that discharges the customer card.


state machine Bank ATM
Off
turn off / shutDown
turn on / startup
failure
Self Test
turn off /
ShutDown
Service
failure
Idle
Maintenance
Out of
Service
service
cardInserted
cancel
failure
Serving Customer
entry / readCard
exit / ejectCard
Customer
Authentication
Selecting
Transaction
Transaction

UML Interaction Diagram

- The interaction diagram portrays the interactions between distinct entities present in the model. It amalgamates both the **activity and sequence diagrams**. Communication is nothing but units of the behavior of a classifier that provides context for interactions.
- A set of messages that are interchanged between the entities to achieve certain specified tasks in the system is termed as interaction. It may incorporate any feature of the classifier of which it has access.
- In the interaction diagram, the critical component is the messages and the lifeline.
- In UML, the interaction overview diagram initiates the interaction between the objects utilizing message passing. While drawing an interaction diagram, the entire focus is to represent the relationship among different objects which are available within the system boundary and the message exchanged by them to communicate with each other.
- The message exchanged among objects is either to pass some information or to request some information. And based on the information, the interaction diagram is categorized into the sequence diagram, collaboration diagram, and timing diagram.
- The sequence diagram envisions the order of the flow of messages inside the system by depicting the communication between two lifelines, just like a time-ordered sequence of events.
- The collaboration diagram, which is also known as the communication diagram, represents how lifelines connect within the system, whereas the timing diagram focuses on that instant when a message is passed from one element to the other.
- Notation of an Interaction Diagram



Purpose of an Interaction Diagram

- The interaction diagram helps to envision the interactive (dynamic) behavior of any system. It portrays how objects residing in the system communicates and connects to each other. It also provides us with a context of communication between the lifelines inside the system.
- Following are the purpose of an interaction diagram given below:

1. To visualize the dynamic behavior of the system.
2. To envision the interaction and the message flow in the system.
3. To portray the structural aspects of the entities within the system.
4. To represent the order of the sequenced interaction in the system.
5. To visualize the real-time data and represent the architecture of an object-oriented system.

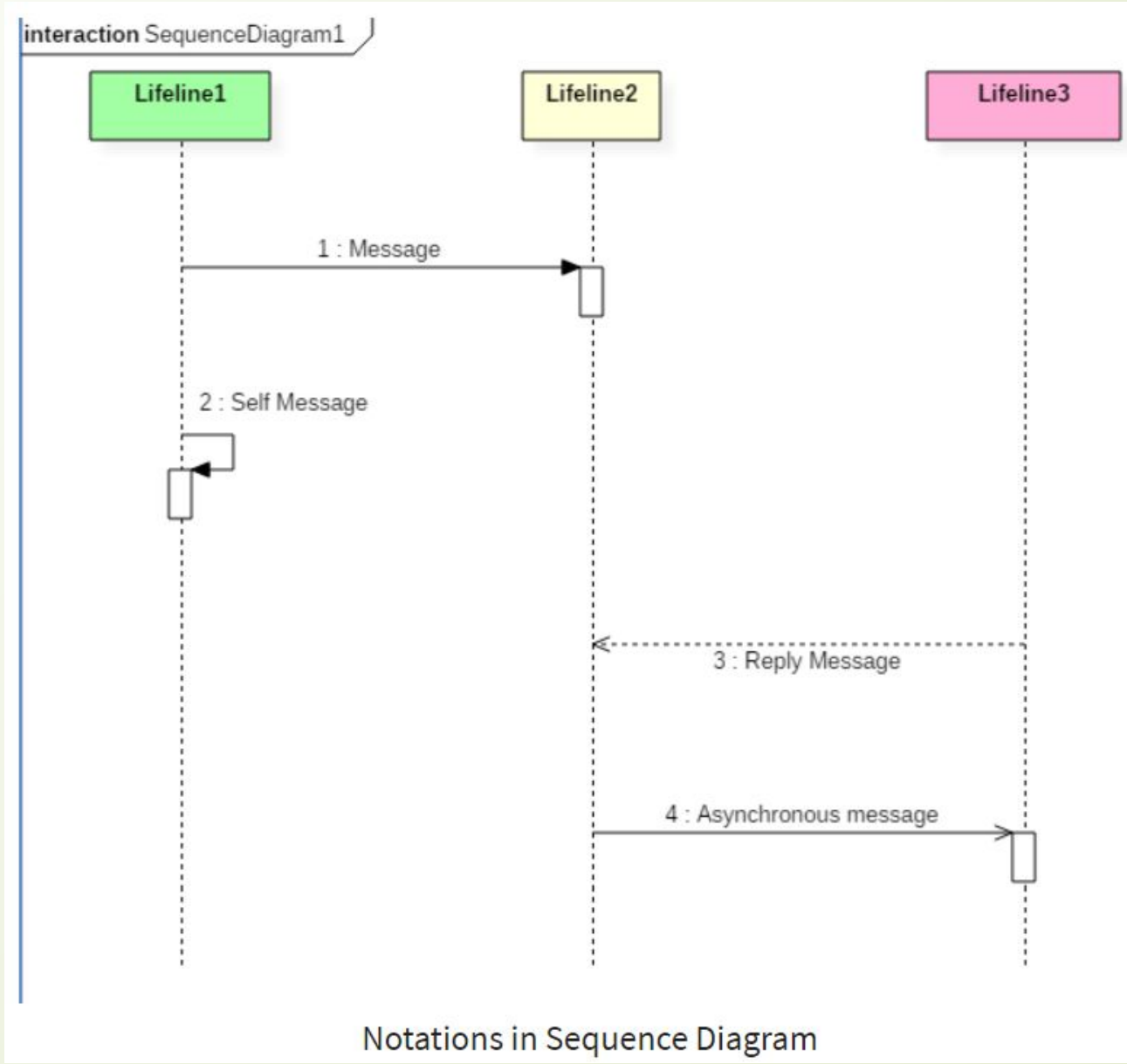
Use of an Interaction Diagram

- The interaction diagram can be used for:

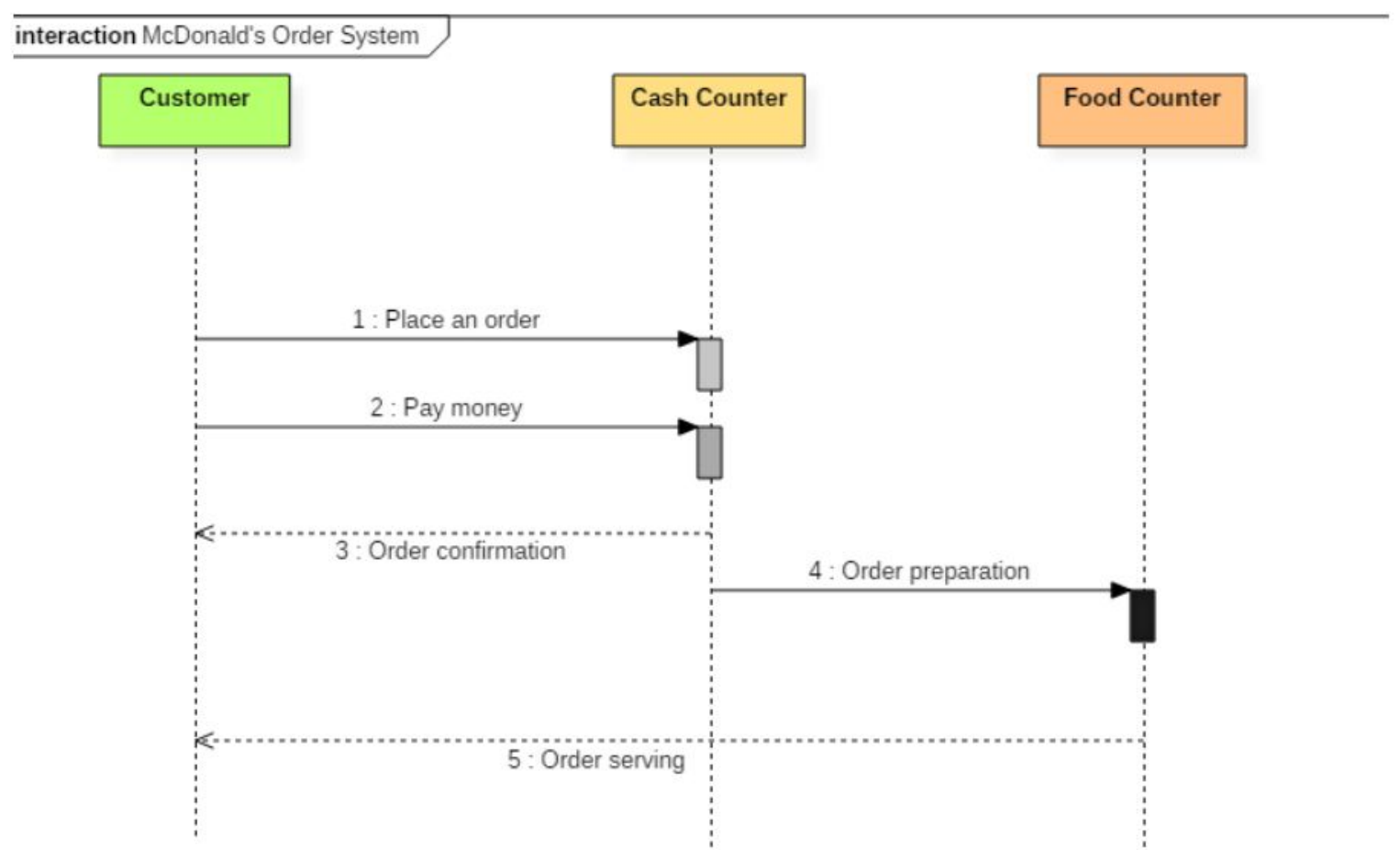
1. The sequence diagram is employed to investigate a new application.
2. The interaction diagram explores and compares the use of the collaboration diagram sequence diagram and the timing diagram.
3. The interaction diagram represents the interactive (dynamic) behavior of the system.
4. The sequence diagram portrays the order of control flow from one element to the other elements inside the system, whereas the collaboration diagrams are employed to get an overview of the object architecture of the system.
5. The interaction diagram models the system as a time-ordered sequence of a system.
6. The interaction diagram systemizes the structure of the interactive elements.

Sequence diagram

- A **Sequence Diagram** simply depicts interaction between **objects** in a sequential order. The purpose of a sequence diagram in UML is to visualize the sequence of a message flow in the system. The sequence diagram shows the interaction between two lifelines as a **time-ordered sequence of events**.
- A sequence diagram shows an implementation of a scenario in the system. Lifelines in the system take part during the execution of a system.
- In a sequence diagram, a lifeline is represented by a vertical bar.
- A message flow between two or more objects is represented using a vertical dotted line which extends across the bottom of the page.
- In a sequence diagram, different types of messages and operators are used which are described above.
- In a sequence diagram, iteration and branching are also used.



Notations in Sequence Diagram



Sequence diagram of Mcdonald's ordering system

Object: Rectangle
Request: Solid line
Response: Dotted line
Activation time: Vertical Rectangle