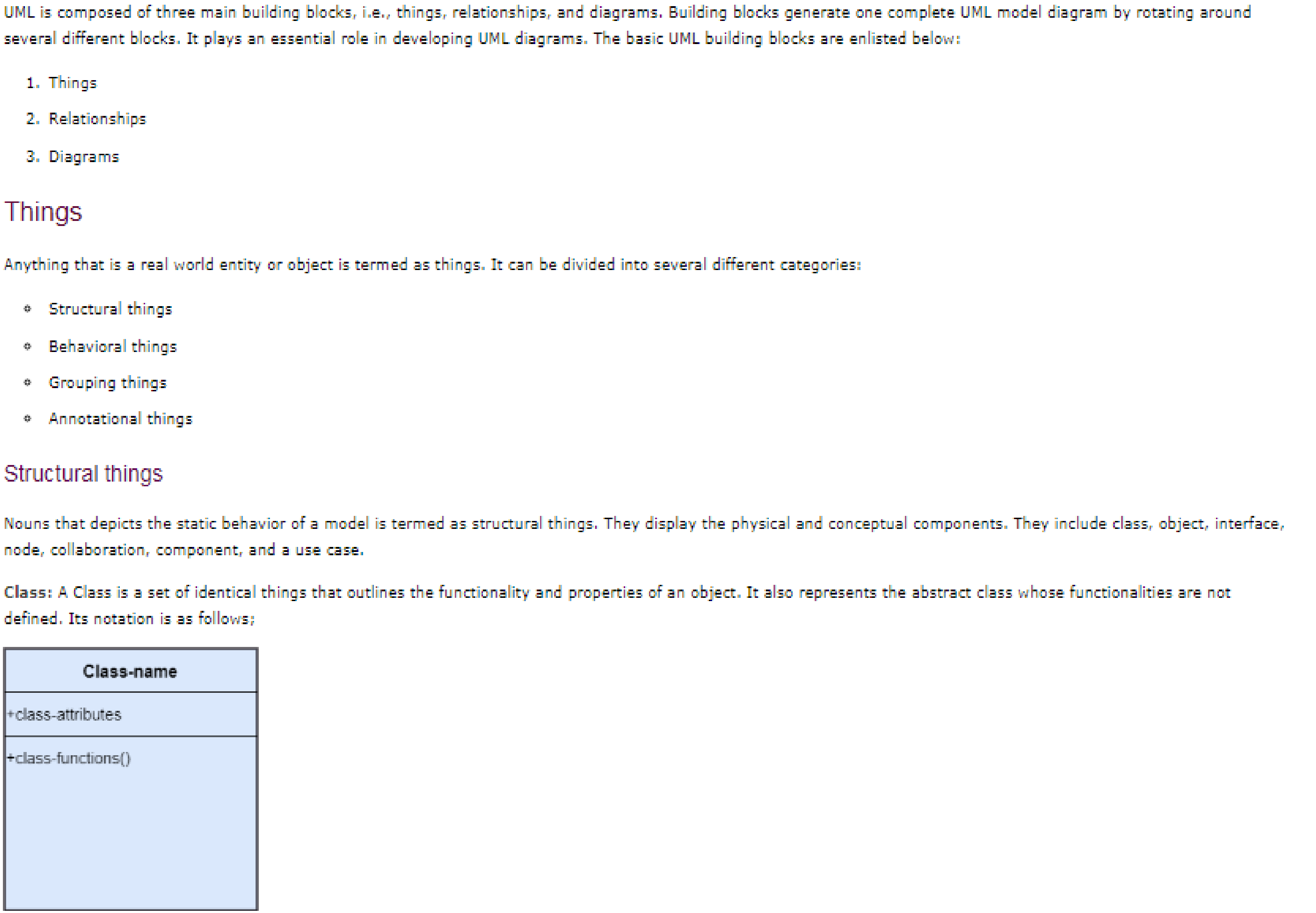
UML Building Blocks



UML is composed of three main building blocks, i.e., things, relationships, and diagrams. Building blocks generate one complete UML model diagram by rotating around several different blocks. It plays an essential role in developing UML diagrams. The basic UML building blocks are enlisted below:

1. Things
2. Relationships
3. Diagrams

Things

Anything that is a real world entity or object is termed as things. It can be divided into several different categories:

- Structural things
- Behavioral things
- Grouping things
- Annotational things

Structural things

Nouns that depicts the static behavior of a model is termed as structural things. They display the physical and conceptual components. They include class, object, interface, node, collaboration, component, and a use case.

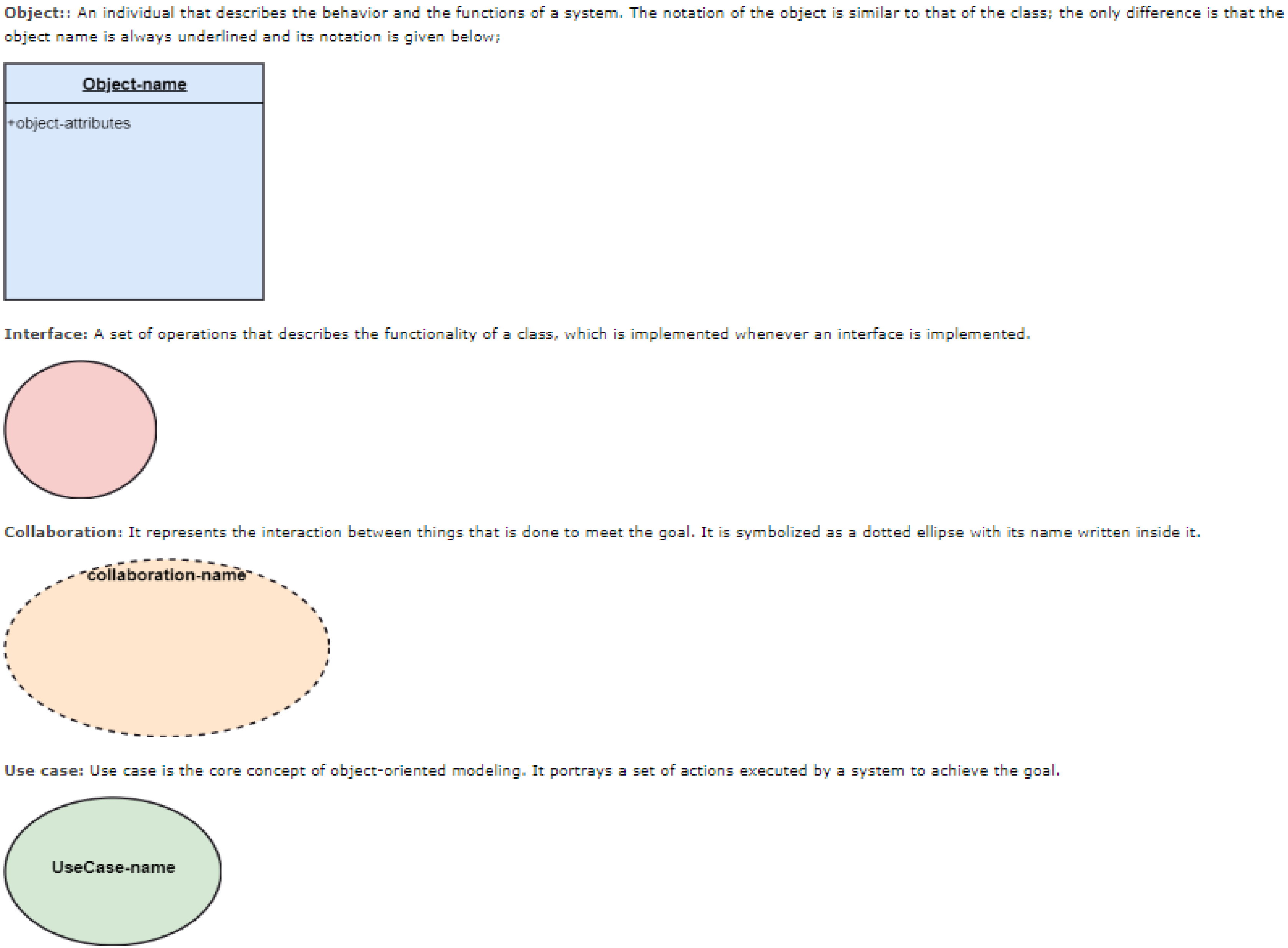
Class: A Class is a set of identical things that outlines the functionality and properties of an object. It also represents the abstract class whose functionalities are not defined. Its notation is as follows;

Class-name

+class-attributes

+class-functions()

UML Building Blocks (contd..)



Object:: An individual that describes the behavior and the functions of a system. The notation of the object is similar to that of the class; the only difference is that the object name is always underlined and its notation is given below;

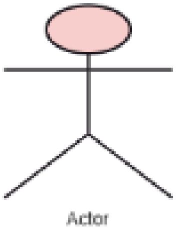
Interface: A set of operations that describes the functionality of a class, which is implemented whenever an interface is implemented.

Collaboration: It represents the interaction between things that is done to meet the goal. It is symbolized as a dotted ellipse with its name written inside it.

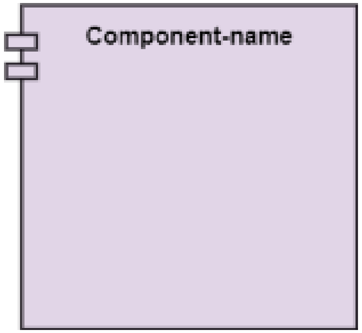
Use case: Use case is the core concept of object-oriented modeling. It portrays a set of actions executed by a system to achieve the goal.

UML Building Blocks (contd..)

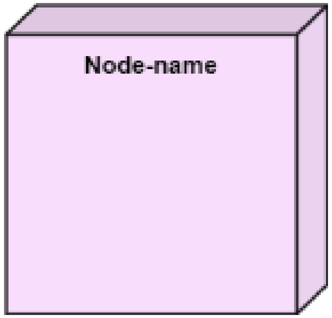
Actor: It comes under the use case diagrams. It is an object that interacts with the system, for example, a user.



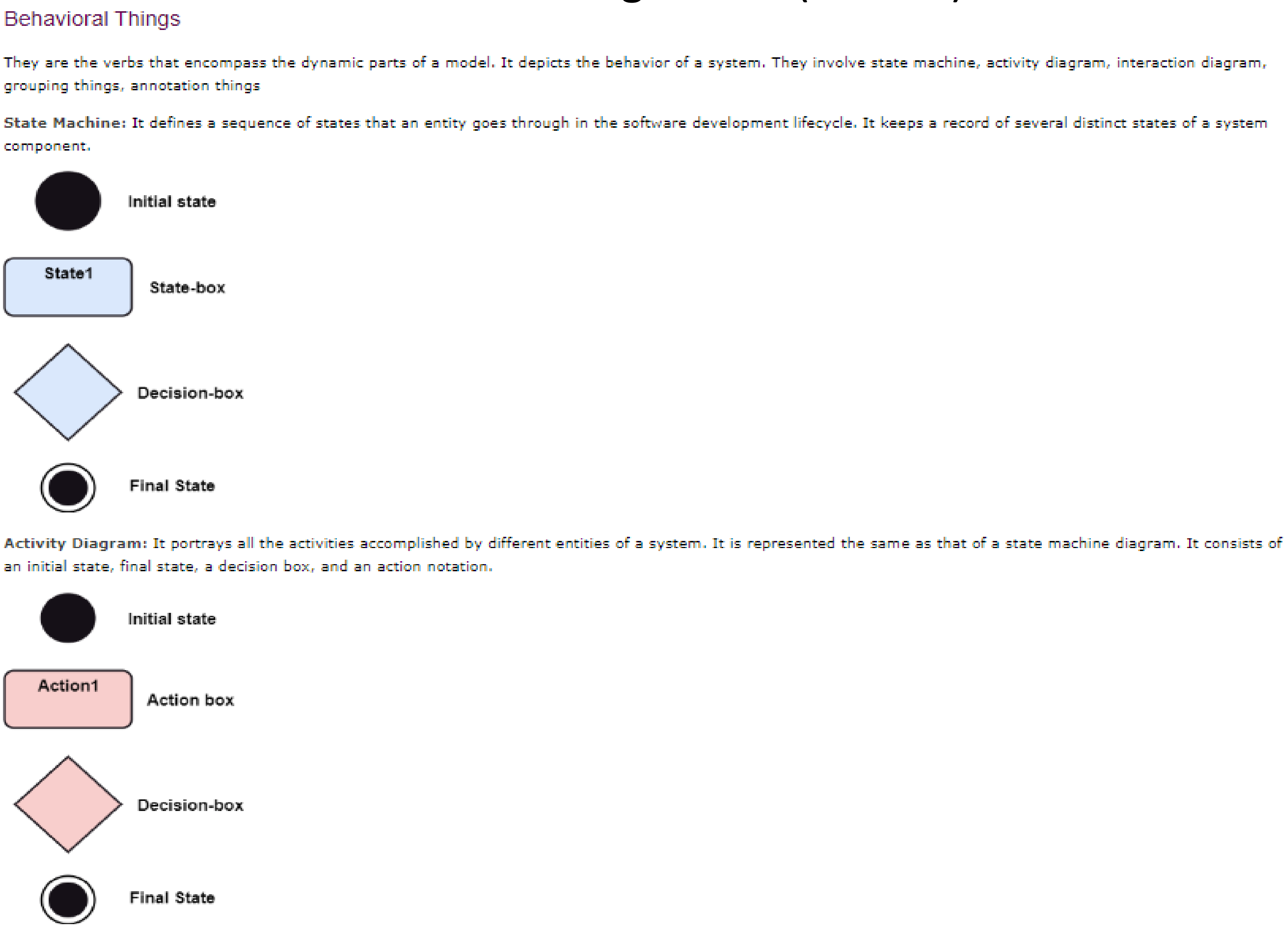
Component: It represents the physical part of the system.



Node: A physical element that exists at run time.



UML Building Blocks (contd..)



Behavioral Things

They are the verbs that encompass the dynamic parts of a model. It depicts the behavior of a system. They involve state machine, activity diagram, interaction diagram, grouping things, annotation things

State Machine: It defines a sequence of states that an entity goes through in the software development lifecycle. It keeps a record of several distinct states of a system component.

Initial state

State1

State-box

Decision-box

Final State

Activity Diagram: It portrays all the activities accomplished by different entities of a system. It is represented the same as that of a state machine diagram. It consists of an initial state, final state, a decision box, and an action notation.

Initial state

Action1

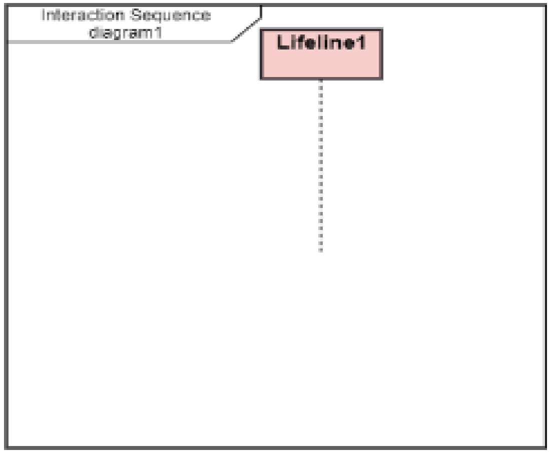
Action box

Decision-box

Final State

UML Building Blocks (contd..)

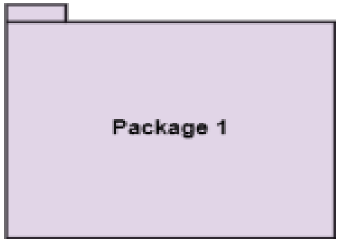
Interaction Diagram: It is used to envision the flow of messages between several components in a system.



Grouping Things

It is a method that together binds the elements of the UML model. In UML, the package is the only thing, which is used for grouping.

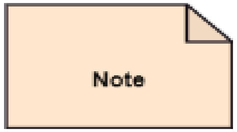
Package: Package is the only thing that is available for grouping behavioral and structural things.



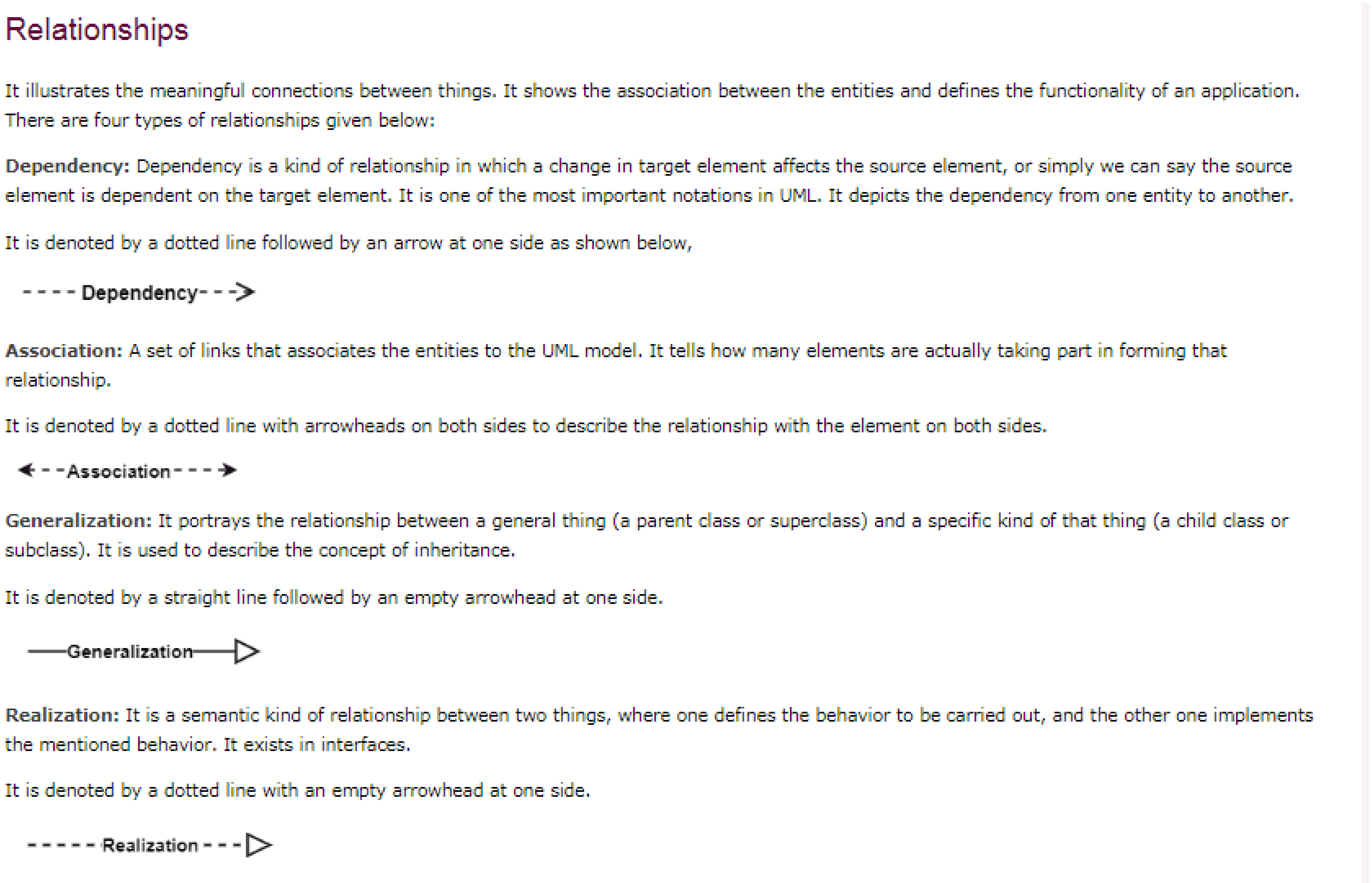
Annotation Things

It is a mechanism that captures the remarks, descriptions, and comments of UML model elements. In UML, a note is the only Annotational thing.

Note: It is used to attach the constraints, comments, and rules to the elements of the model. It is a kind of yellow sticky note.



UML Building Blocks (contd..)



Relationships

It illustrates the meaningful connections between things. It shows the association between the entities and defines the functionality of an application. There are four types of relationships given below:

Dependency: Dependency is a kind of relationship in which a change in target element affects the source element, or simply we can say the source element is dependent on the target element. It is one of the most important notations in UML. It depicts the dependency from one entity to another.

It is denoted by a dotted line followed by an arrow at one side as shown below,

- - - - Dependency- - ->

Association: A set of links that associates the entities to the UML model. It tells how many elements are actually taking part in forming that relationship.

It is denoted by a dotted line with arrowheads on both sides to describe the relationship with the element on both sides.

← - - Association - - - →

Generalization: It portrays the relationship between a general thing (a parent class or superclass) and a specific kind of that thing (a child class or subclass). It is used to describe the concept of inheritance.

It is denoted by a straight line followed by an empty arrowhead at one side.

——Generalization——▷

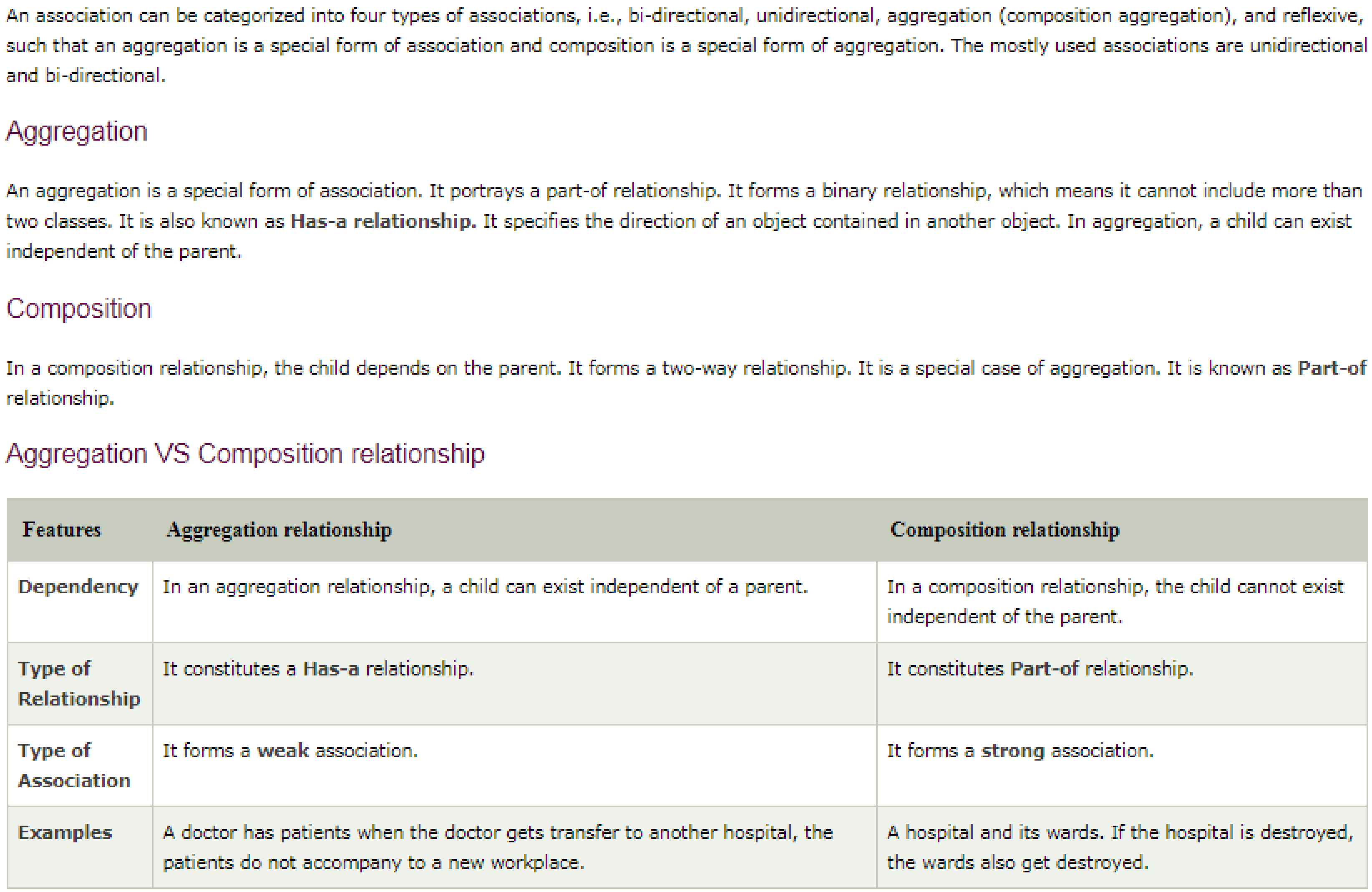
Realization: It is a semantic kind of relationship between two things, where one defines the behavior to be carried out, and the other one implements the mentioned behavior. It exists in interfaces.

It is denoted by a dotted line with an empty arrowhead at one side.

- - - - - Realization - - - ▷

UML Building Blocks (Association)

Association is a structural relationship that represents how two entities are linked or connected to each other within a system. It can form several types of associations, such as **one-to-one, one-to-many, many-to-one,** and **many-to-many.** A ternary association is one that constitutes three links. It portrays the static relationship between the entities of two classes.



An association can be categorized into four types of associations, i.e., bi-directional, unidirectional, aggregation (composition aggregation), and reflexive, such that an aggregation is a special form of association and composition is a special form of aggregation. The mostly used associations are unidirectional and bi-directional.

Aggregation

An aggregation is a special form of association. It portrays a part-of relationship. It forms a binary relationship, which means it cannot include more than two classes. It is also known as **Has-a relationship.** It specifies the direction of an object contained in another object. In aggregation, a child can exist independent of the parent.

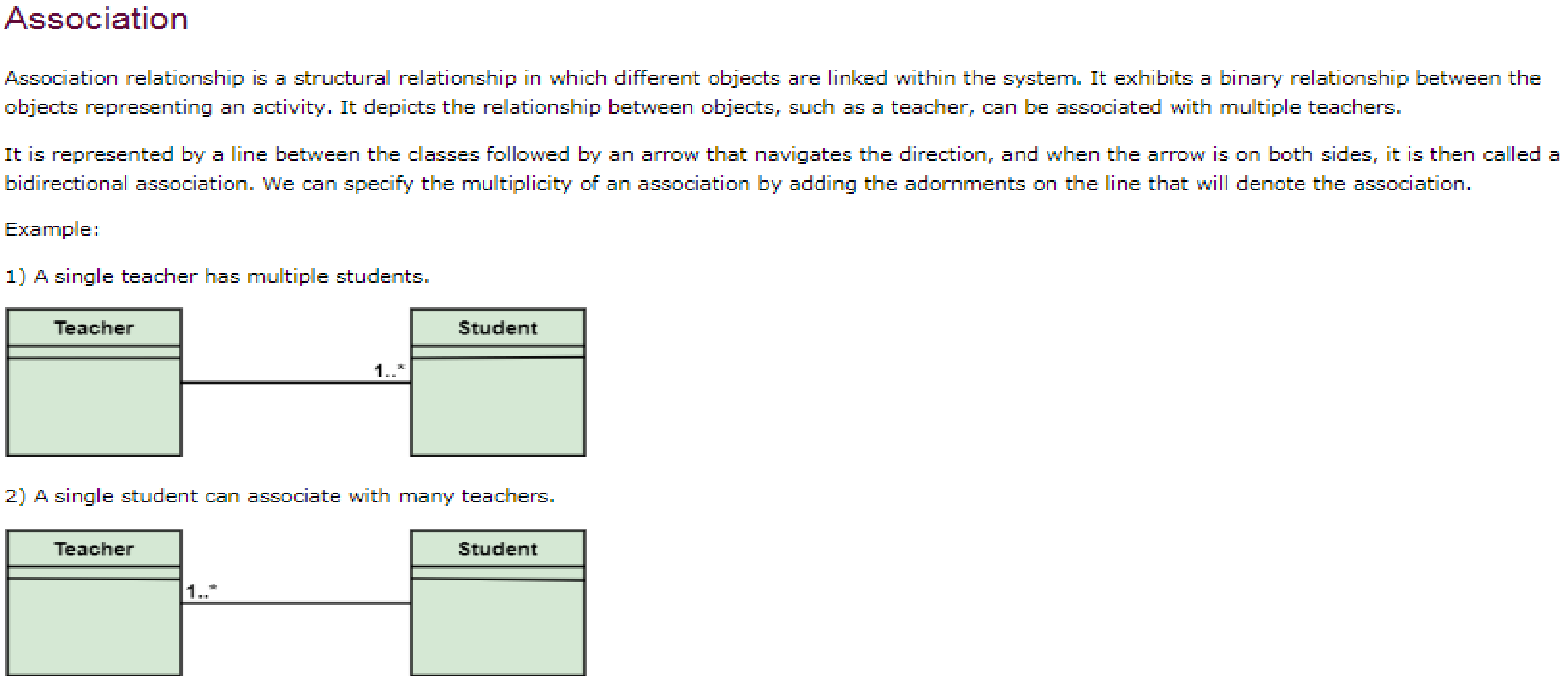
Composition

In a composition relationship, the child depends on the parent. It forms a two-way relationship. It is a special case of aggregation. It is known as **Part-of** relationship.

Aggregation VS Composition relationship

Features	Aggregation relationship	Composition relationship
Dependency	In an aggregation relationship, a child can exist independent of a parent.	In a composition relationship, the child cannot exist independent of the parent.
Type of Relationship	It constitutes a **Has-a** relationship.	It constitutes **Part-of** relationship.
Type of Association	It forms a **weak** association.	It forms a **strong** association.
Examples	A doctor has patients when the doctor gets transfer to another hospital, the patients do not accompany to a new workplace.	A hospital and its wards. If the hospital is destroyed, the wards also get destroyed.

UML Building Blocks (Association: Examples)



Association

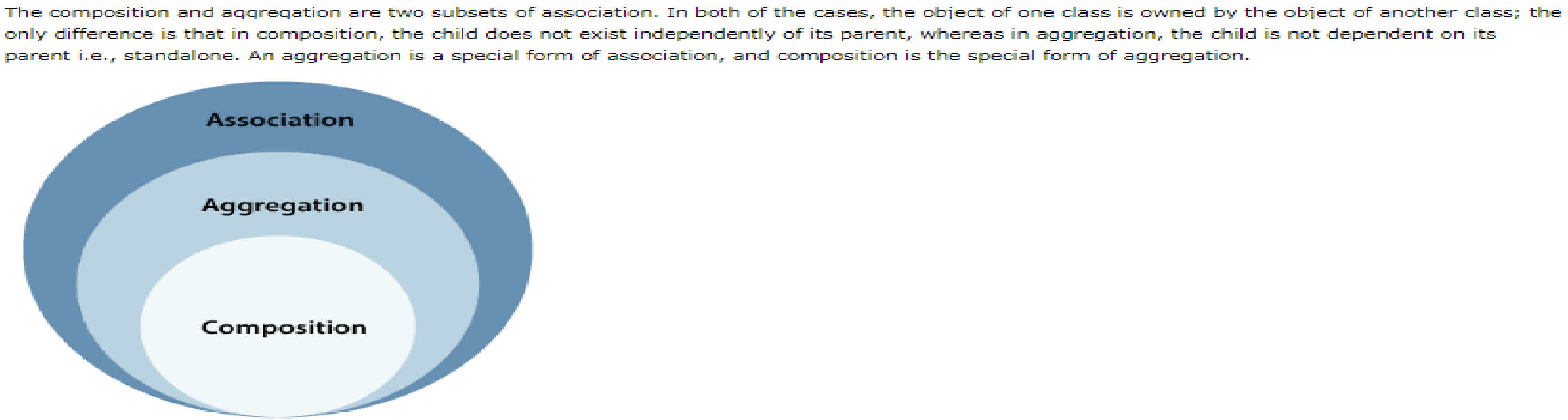
Association relationship is a structural relationship in which different objects are linked within the system. It exhibits a binary relationship between the objects representing an activity. It depicts the relationship between objects, such as a teacher, can be associated with multiple teachers.

It is represented by a line between the classes followed by an arrow that navigates the direction, and when the arrow is on both sides, it is then called a bidirectional association. We can specify the multiplicity of an association by adding the adornments on the line that will denote the association.

Example:

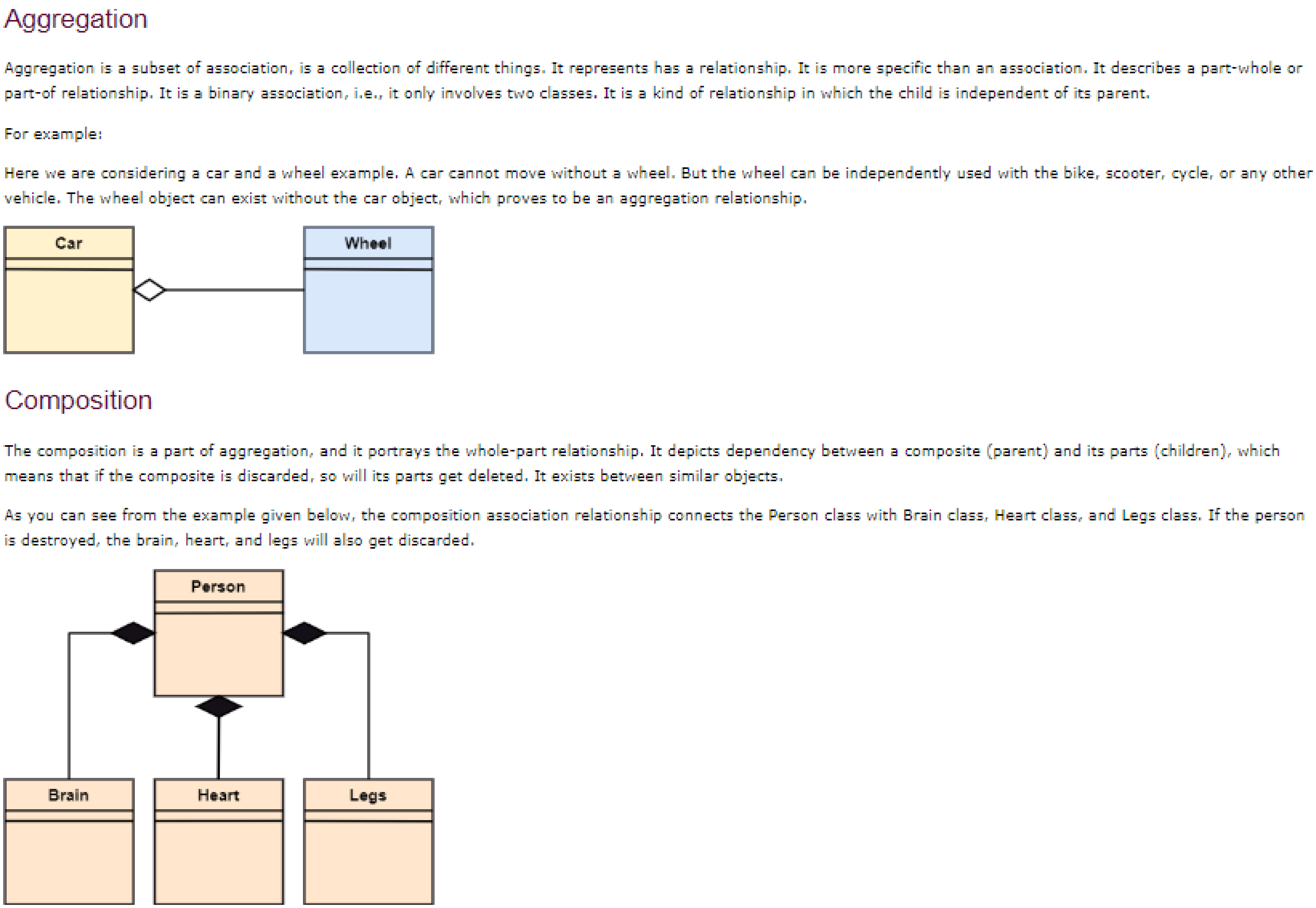
1) A single teacher has multiple students.

2) A single student can associate with many teachers.



The composition and aggregation are two subsets of association. In both of the cases, the object of one class is owned by the object of another class; the only difference is that in composition, the child does not exist independently of its parent, whereas in aggregation, the child is not dependent on its parent i.e., standalone. An aggregation is a special form of association, and composition is the special form of aggregation.

UML Building Blocks (Aggregation & Composition: Examples)



Aggregation

Aggregation is a subset of association, is a collection of different things. It represents has a relationship. It is more specific than an association. It describes a part-whole or part-of relationship. It is a binary association, i.e., it only involves two classes. It is a kind of relationship in which the child is independent of its parent.

For example:

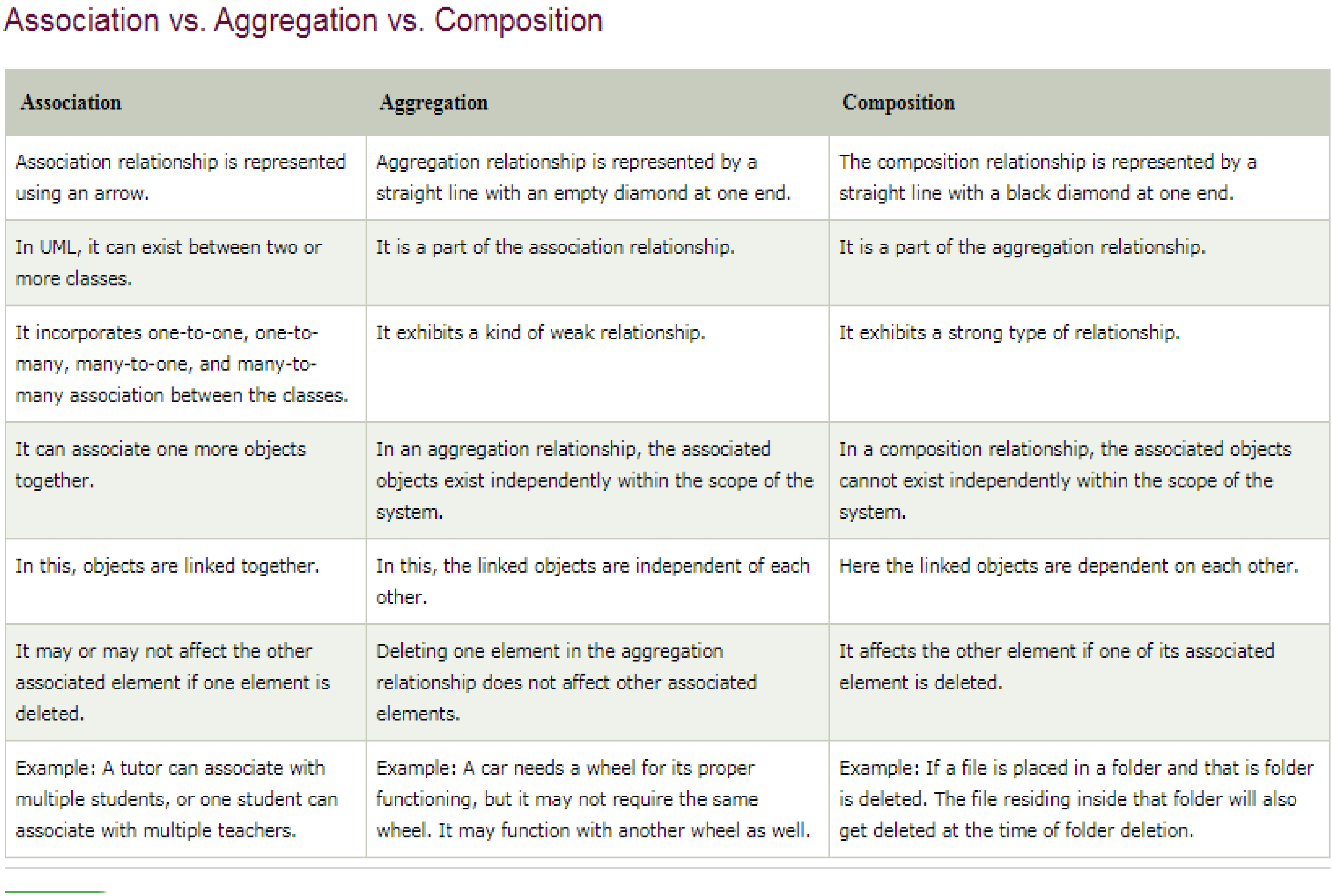
Here we are considering a car and a wheel example. A car cannot move without a wheel. But the wheel can be independently used with the bike, scooter, cycle, or any other vehicle. The wheel object can exist without the car object, which proves to be an aggregation relationship.

Composition

The composition is a part of aggregation, and it portrays the whole-part relationship. It depicts dependency between a composite (parent) and its parts (children), which means that if the composite is discarded, so will its parts get deleted. It exists between similar objects.

As you can see from the example given below, the composition association relationship connects the Person class with Brain class, Heart class, and Legs class. If the person is destroyed, the brain, heart, and legs will also get discarded.

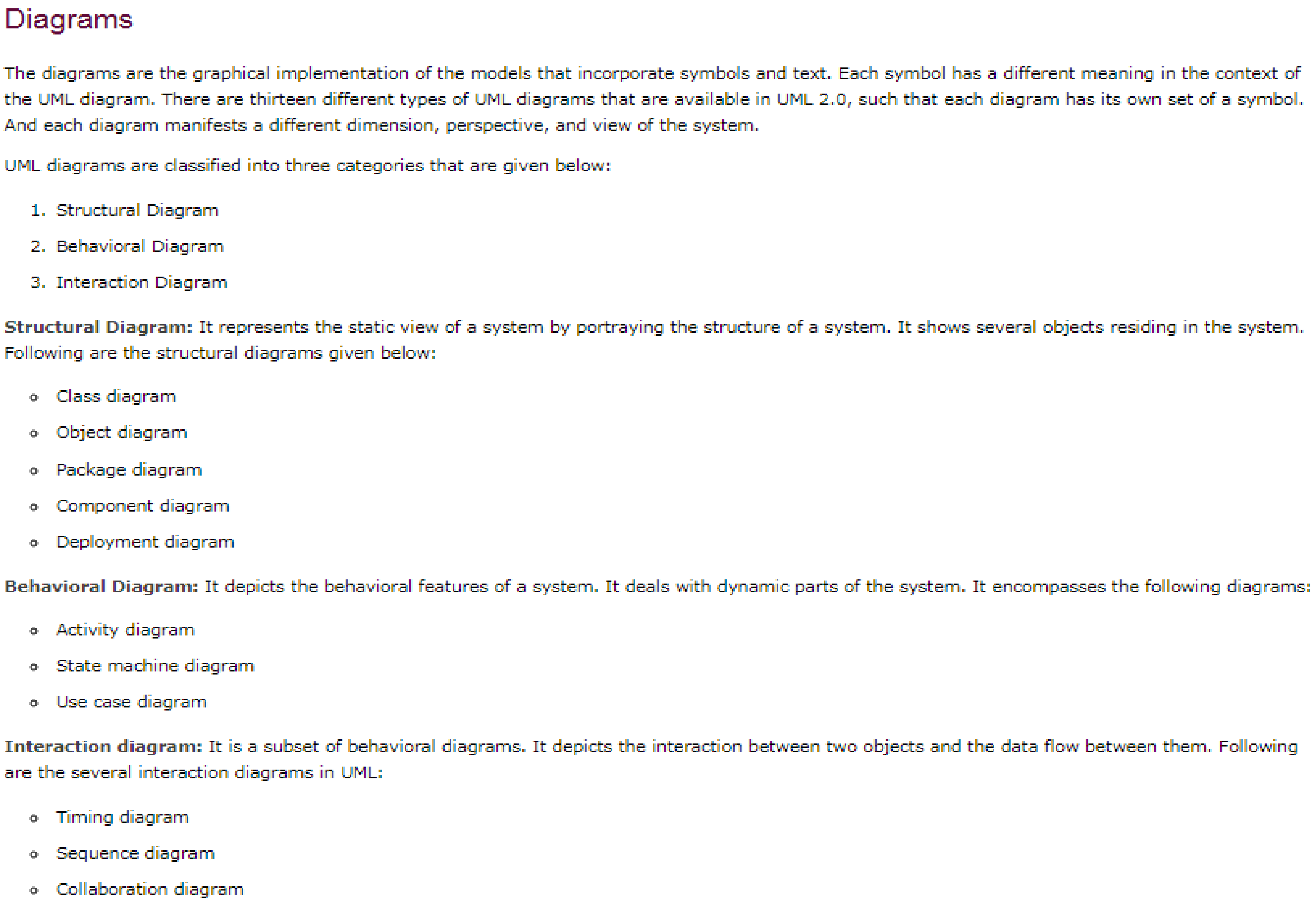
UML Building Blocks (Difference)



Association vs. Aggregation vs. Composition

Association	Aggregation	Composition
Association relationship is represented using an arrow.	Aggregation relationship is represented by a straight line with an empty diamond at one end.	The composition relationship is represented by a straight line with a black diamond at one end.
In UML, it can exist between two or more classes.	It is a part of the association relationship.	It is a part of the aggregation relationship.
It incorporates one-to-one, one-to-many, many-to-one, and many-to-many association between the classes.	It exhibits a kind of weak relationship.	It exhibits a strong type of relationship.
It can associate one more objects together.	In an aggregation relationship, the associated objects exist independently within the scope of the system.	In a composition relationship, the associated objects cannot exist independently within the scope of the system.
In this, objects are linked together.	In this, the linked objects are independent of each other.	Here the linked objects are dependent on each other.
It may or may not affect the other associated element if one element is deleted.	Deleting one element in the aggregation relationship does not affect other associated elements.	It affects the other element if one of its associated element is deleted.
Example: A tutor can associate with multiple students, or one student can associate with multiple teachers.	Example: A car needs a wheel for its proper functioning, but it may not require the same wheel. It may function with another wheel as well.	Example: If a file is placed in a folder and that is folder is deleted. The file residing inside that folder will also get deleted at the time of folder deletion.

UML Building Blocks (contd..)



Diagrams

The diagrams are the graphical implementation of the models that incorporate symbols and text. Each symbol has a different meaning in the context of the UML diagram. There are thirteen different types of UML diagrams that are available in UML 2.0, such that each diagram has its own set of a symbol. And each diagram manifests a different dimension, perspective, and view of the system.

UML diagrams are classified into three categories that are given below:

1. Structural Diagram
2. Behavioral Diagram
3. Interaction Diagram

Structural Diagram: It represents the static view of a system by portraying the structure of a system. It shows several objects residing in the system. Following are the structural diagrams given below:

- Class diagram
- Object diagram
- Package diagram
- Component diagram
- Deployment diagram

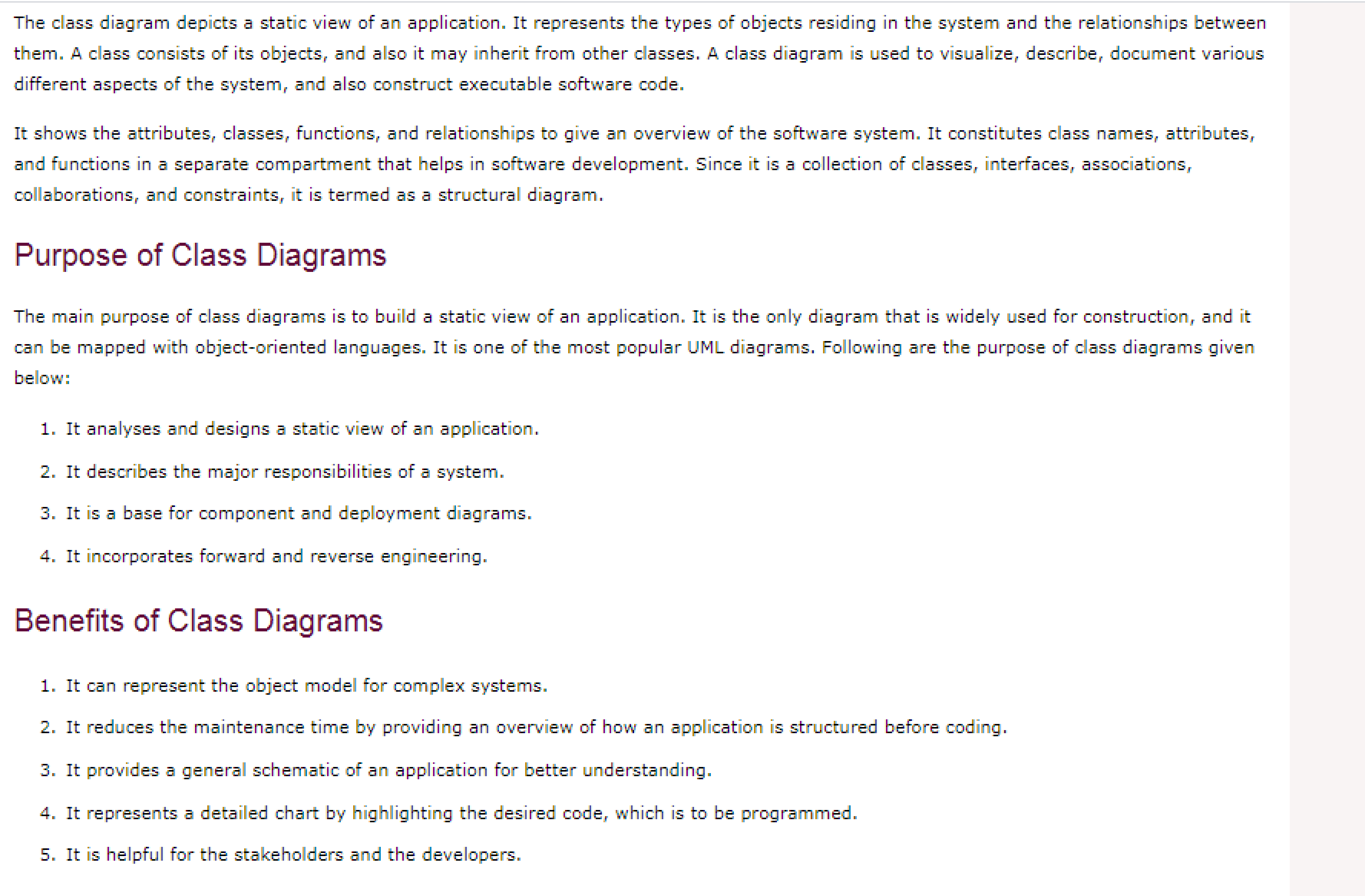
Behavioral Diagram: It depicts the behavioral features of a system. It deals with dynamic parts of the system. It encompasses the following diagrams:

- Activity diagram
- State machine diagram
- Use case diagram

Interaction diagram: It is a subset of behavioral diagrams. It depicts the interaction between two objects and the data flow between them. Following are the several interaction diagrams in UML:

- Timing diagram
- Sequence diagram
- Collaboration diagram

Class Diagram



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.

It shows the attributes, classes, functions, and relationships to give an overview of the software system. It constitutes class names, attributes, and functions in a separate compartment that helps in software development. Since it is a collection of classes, interfaces, associations, collaborations, and constraints, it is termed as a structural diagram.

Purpose of Class Diagrams

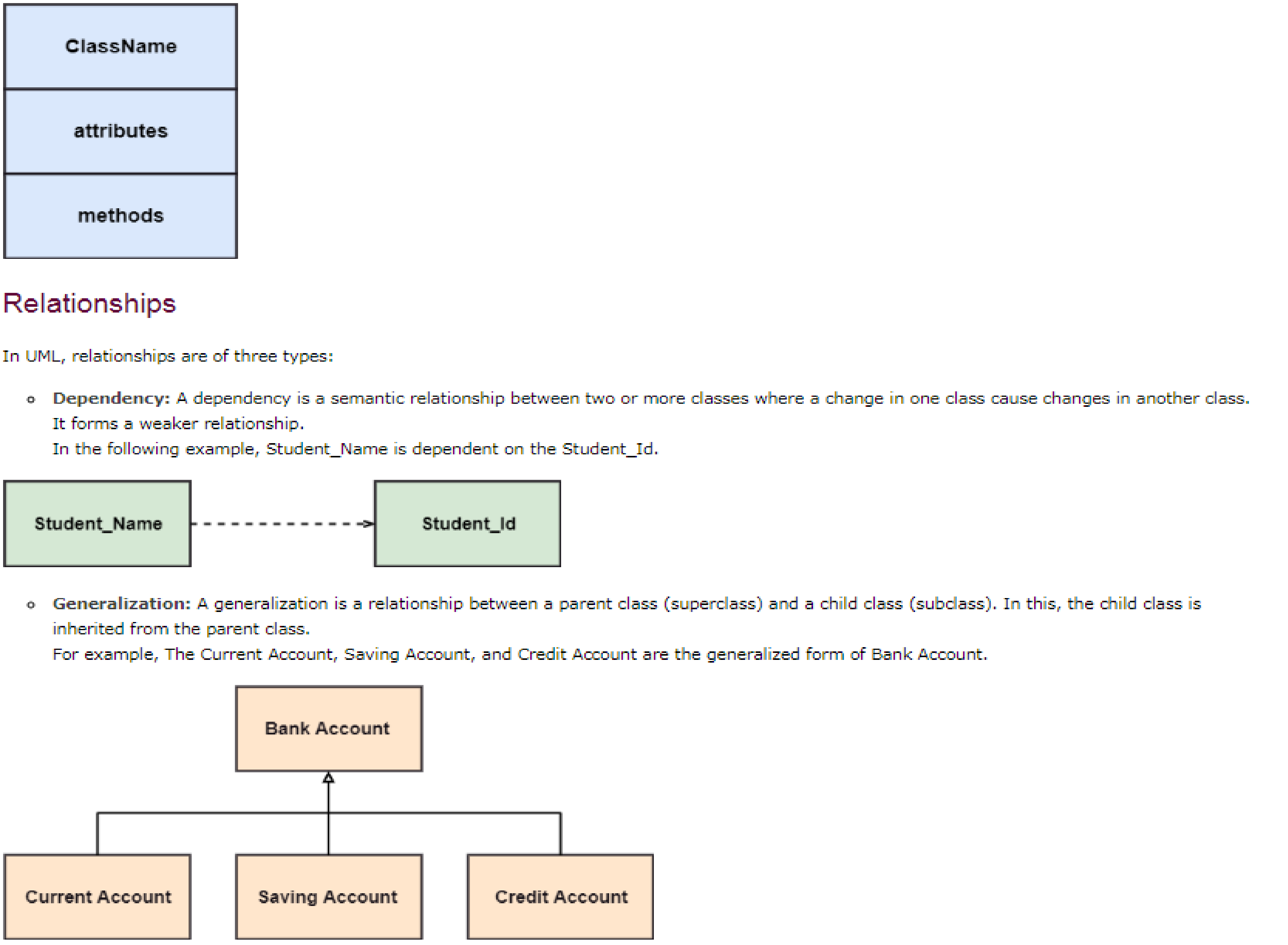
The main purpose of class diagrams is to build a static view of an application. It is the only diagram that is widely used for construction, and it can be mapped with object-oriented languages. It is one of the most popular UML diagrams. Following are the purpose of class diagrams given below:

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.

Benefits of Class Diagrams

1. It can represent the object model for complex systems.
2. It reduces the maintenance time by providing an overview of how an application is structured before coding.
3. It provides a general schematic of an application for better understanding.
4. It represents a detailed chart by highlighting the desired code, which is to be programmed.
5. It is helpful for the stakeholders and the developers.

Class Diagram (Components & Relationship)



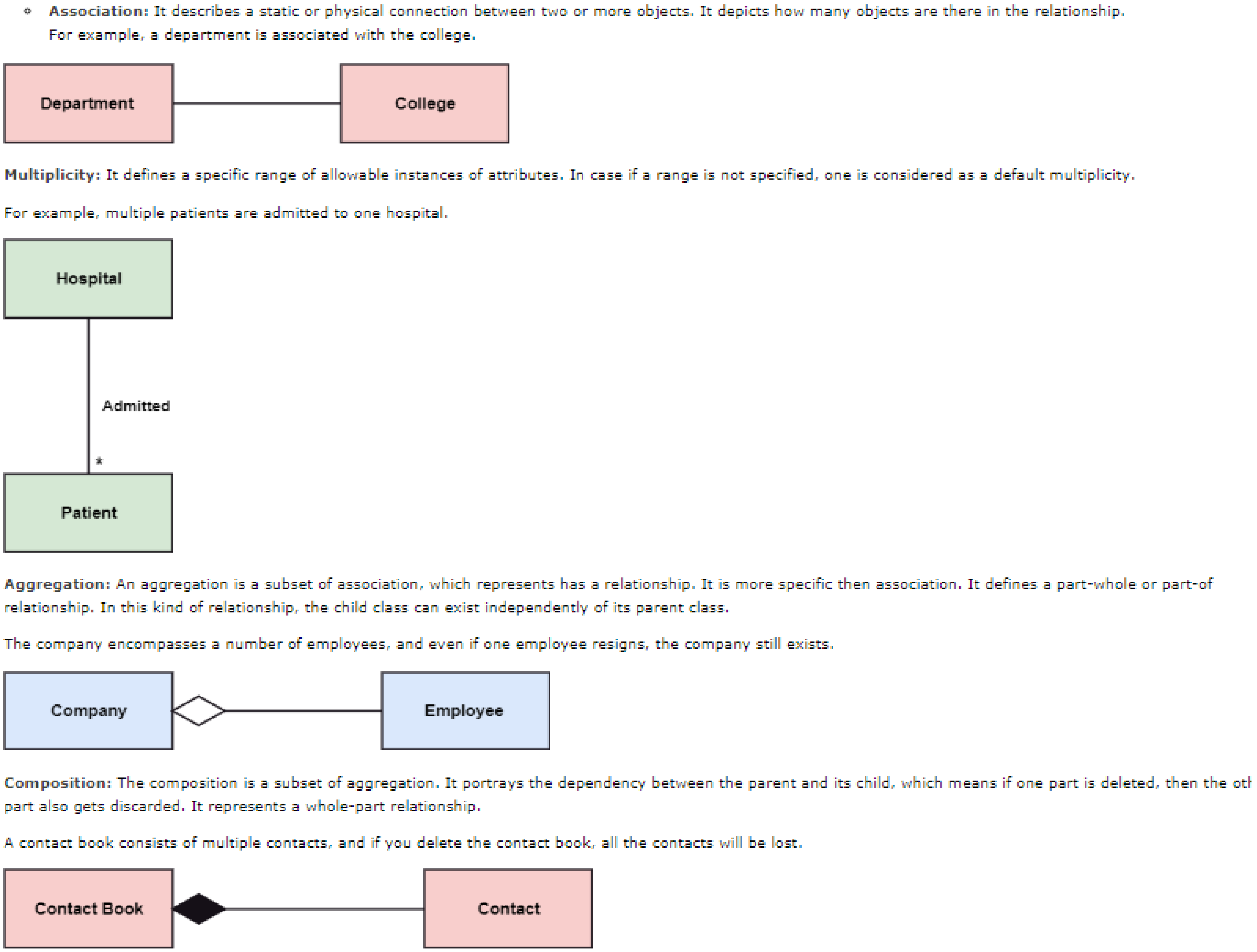
Relationships

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.

Class Diagram (Components & Relationship)



- **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.

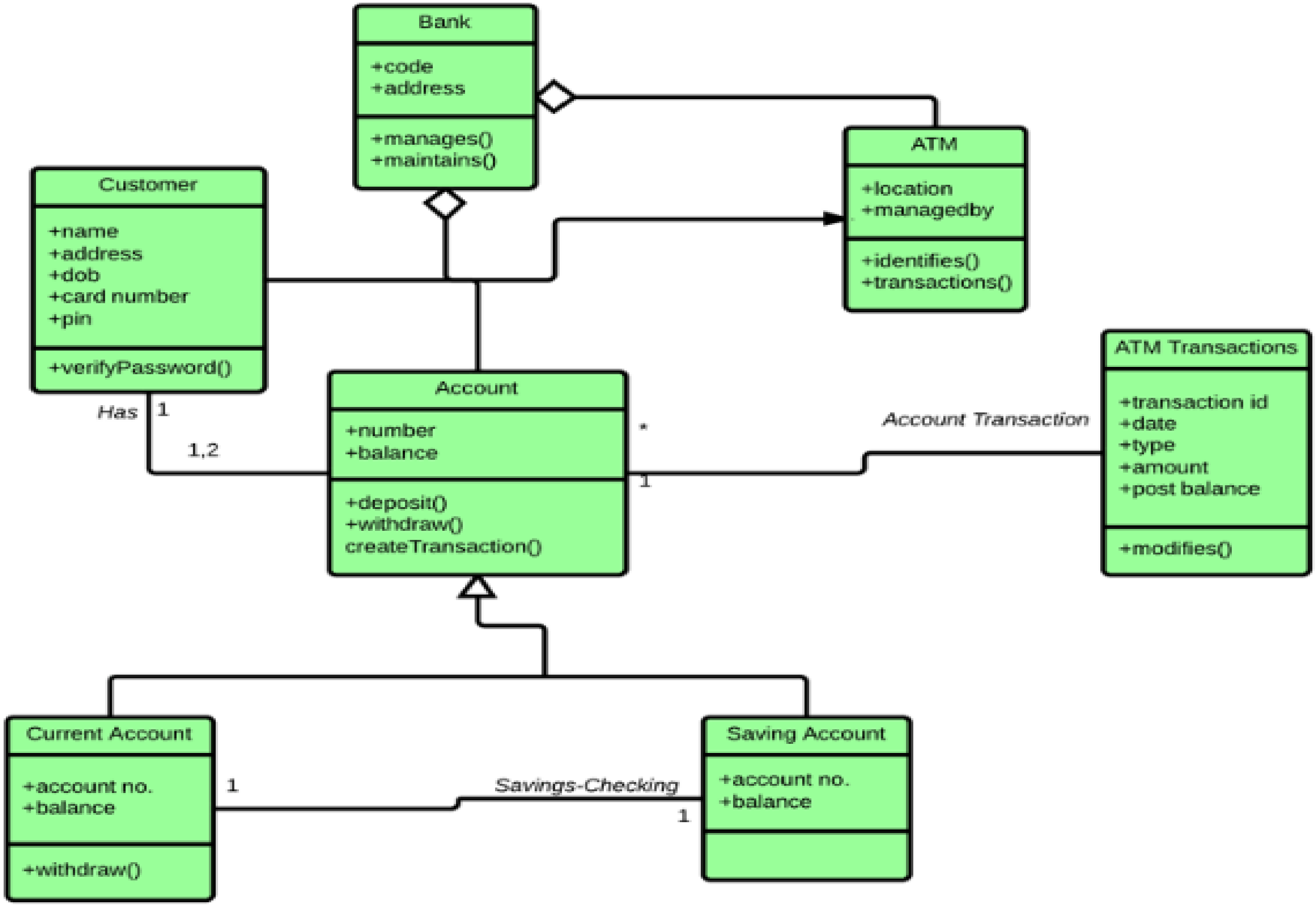
Aggregation: An aggregation is a subset of association, which represents has a relationship. It is more specific then association. It defines a part-whole or part-of relationship. In this kind of relationship, the child class can exist independently of its parent class.

The company encompasses a number of employees, and even if one employee resigns, the company still exists.

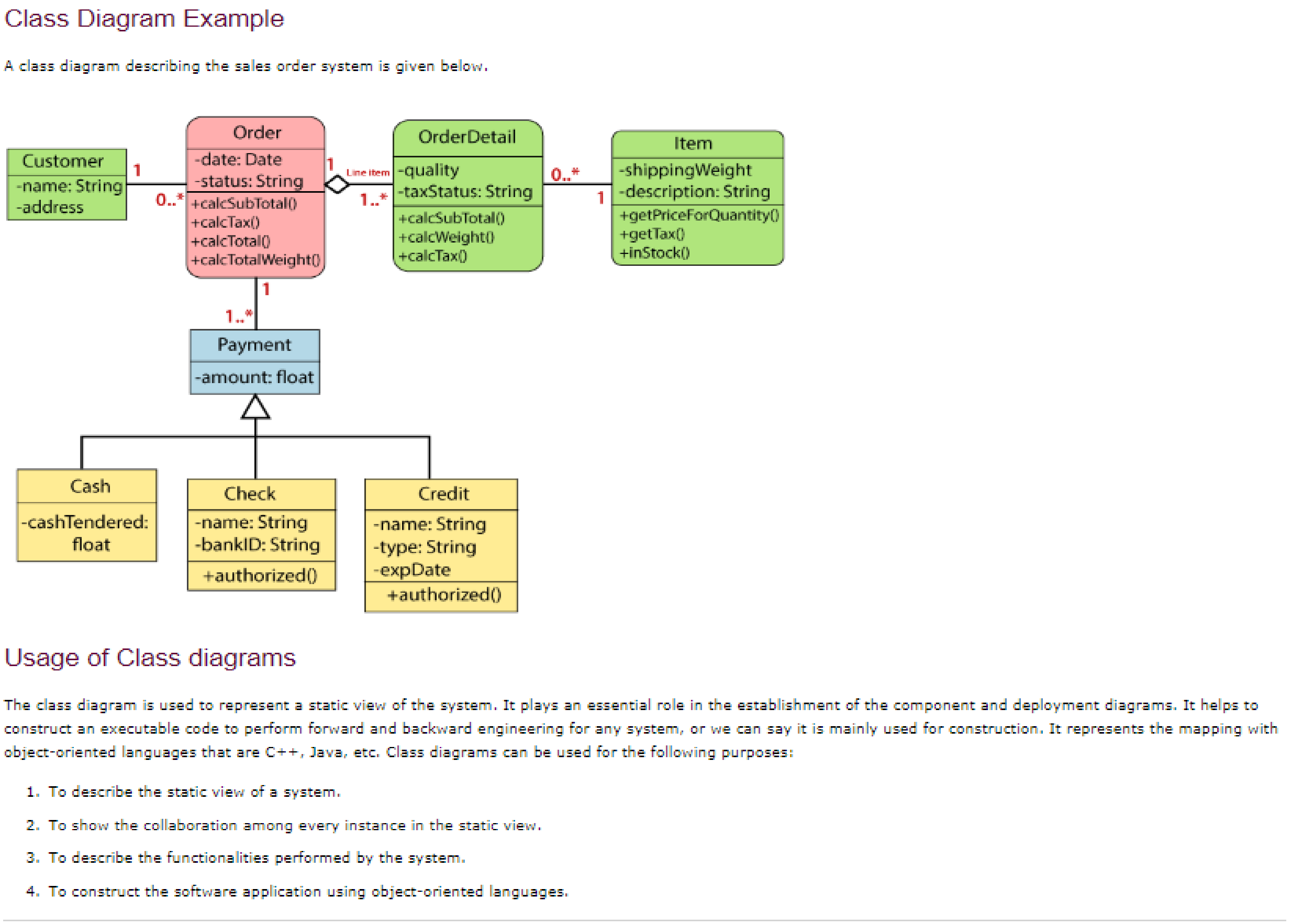
Composition: The composition is a subset of aggregation. It portrays the dependency between the parent and its child, which means if one part is deleted, then the oth part also gets discarded. It represents a whole-part relationship.

A contact book consists of multiple contacts, and if you delete the contact book, all the contacts will be lost.

Class Diagram (Example)



Class Diagram (Example & uses)



Class Diagram Example

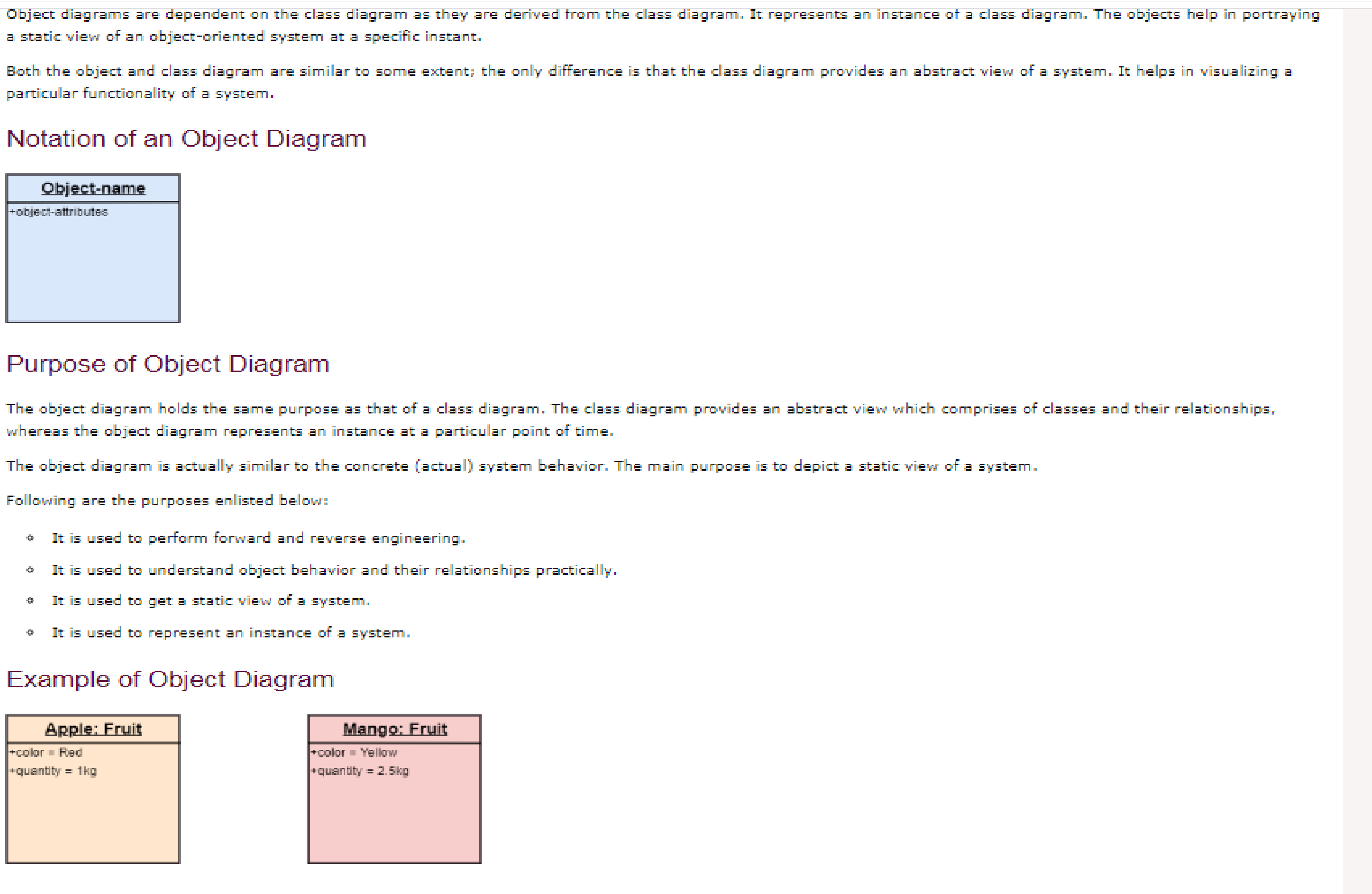
A class diagram describing the sales order system is given below.

Usage of Class diagrams

The class diagram is used to represent a static view of the system. It plays an essential role in the establishment of the component and deployment diagrams. It helps to construct an executable code to perform forward and backward engineering for any system, or we can say it is mainly used for construction. It represents the mapping with object-oriented languages that are C++, Java, etc. Class diagrams can be used for the following purposes:

1. To describe the static view of a system.
2. To show the collaboration among every instance in the static view.
3. To describe the functionalities performed by the system.
4. To construct the software application using object-oriented languages.

Object Diagram



Object diagrams are dependent on the class diagram as they are derived from the class diagram. It represents an instance of a class diagram. The objects help in portraying a static view of an object-oriented system at a specific instant.

Both the object and class diagram are similar to some extent; the only difference is that the class diagram provides an abstract view of a system. It helps in visualizing a particular functionality of a system.

Notation of an Object Diagram

Object-name

+object-attributes

Purpose of Object Diagram

The object diagram holds the same purpose as that of a class diagram. The class diagram provides an abstract view which comprises of classes and their relationships, whereas the object diagram represents an instance at a particular point of time.

The object diagram is actually similar to the concrete (actual) system behavior. The main purpose is to depict a static view of a system.

Following are the purposes enlisted below:

- It is used to perform forward and reverse engineering.
- It is used to understand object behavior and their relationships practically.
- It is used to get a static view of a system.
- It is used to represent an instance of a system.

Example of Object Diagram

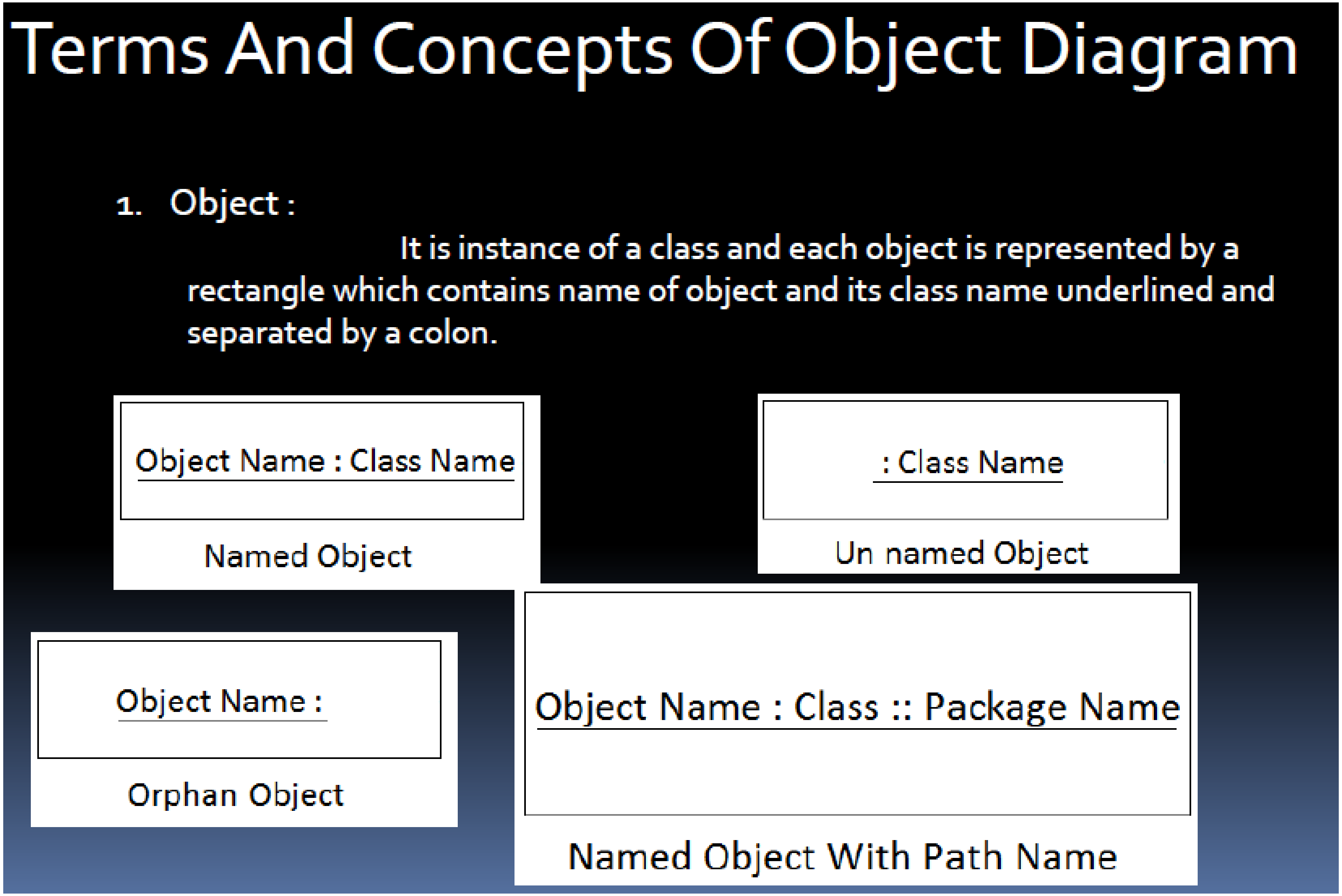
Apple: Fruit

+color = Red
+quantity = 1kg

Mango: Fruit

+color = Yellow
+quantity = 2.5kg

Object Diagram (contd..)



Terms And Concepts Of Object Diagram

1. Object :

It is instance of a class and each object is represented by a rectangle which contains name of object and its class name underlined and separated by a colon.

Object Name : Class Name

Named Object

: Class Name

Un named Object

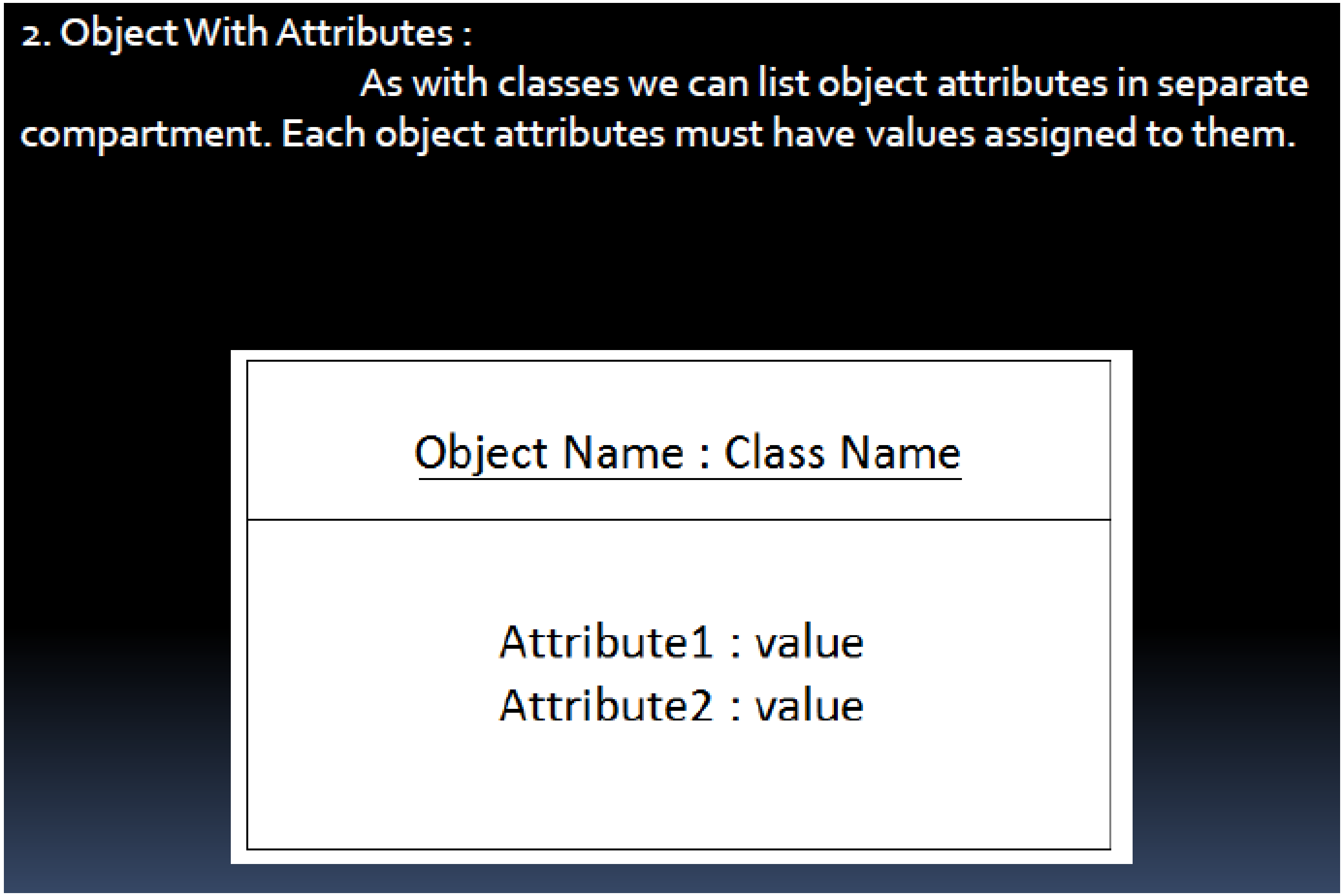
Object Name :

Orphan Object

Object Name : Class :: Package Name

Named Object With Path Name

Object Diagram (contd..)



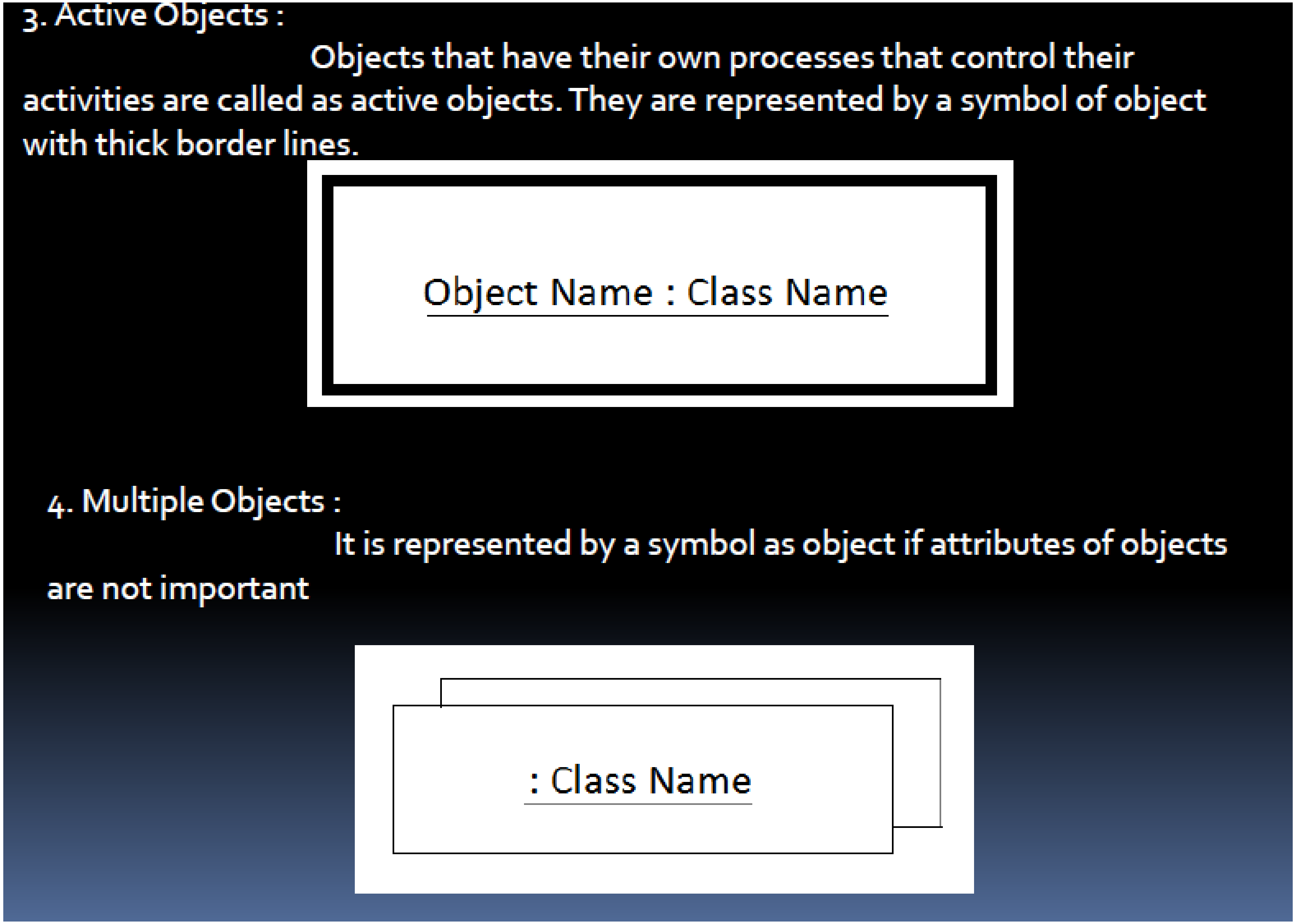
2. Object With Attributes :

As with classes we can list object attributes in separate compartment. Each object attributes must have values assigned to them.

Object Name : Class Name

Attribute1 : value
Attribute2 : value

Object Diagram (contd..)



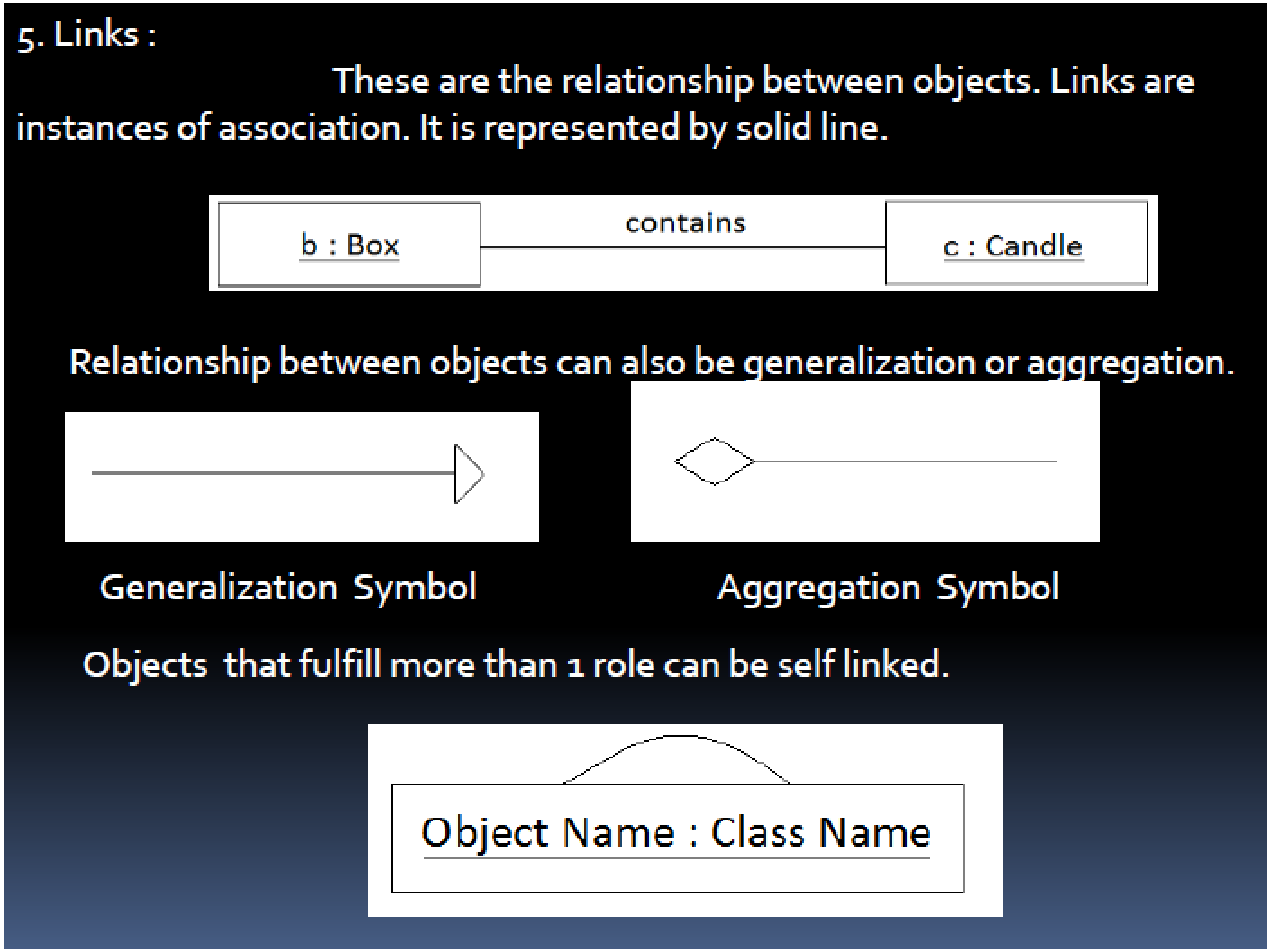
3. Active Objects :

Objects that have their own processes that control their activities are called as active objects. They are represented by a symbol of object with thick border lines.

4. Multiple Objects :

It is represented by a symbol as object if attributes of objects are not important

Object Diagram (contd..)



5. Links :

These are the relationship between objects. Links are instances of association. It is represented by solid line.

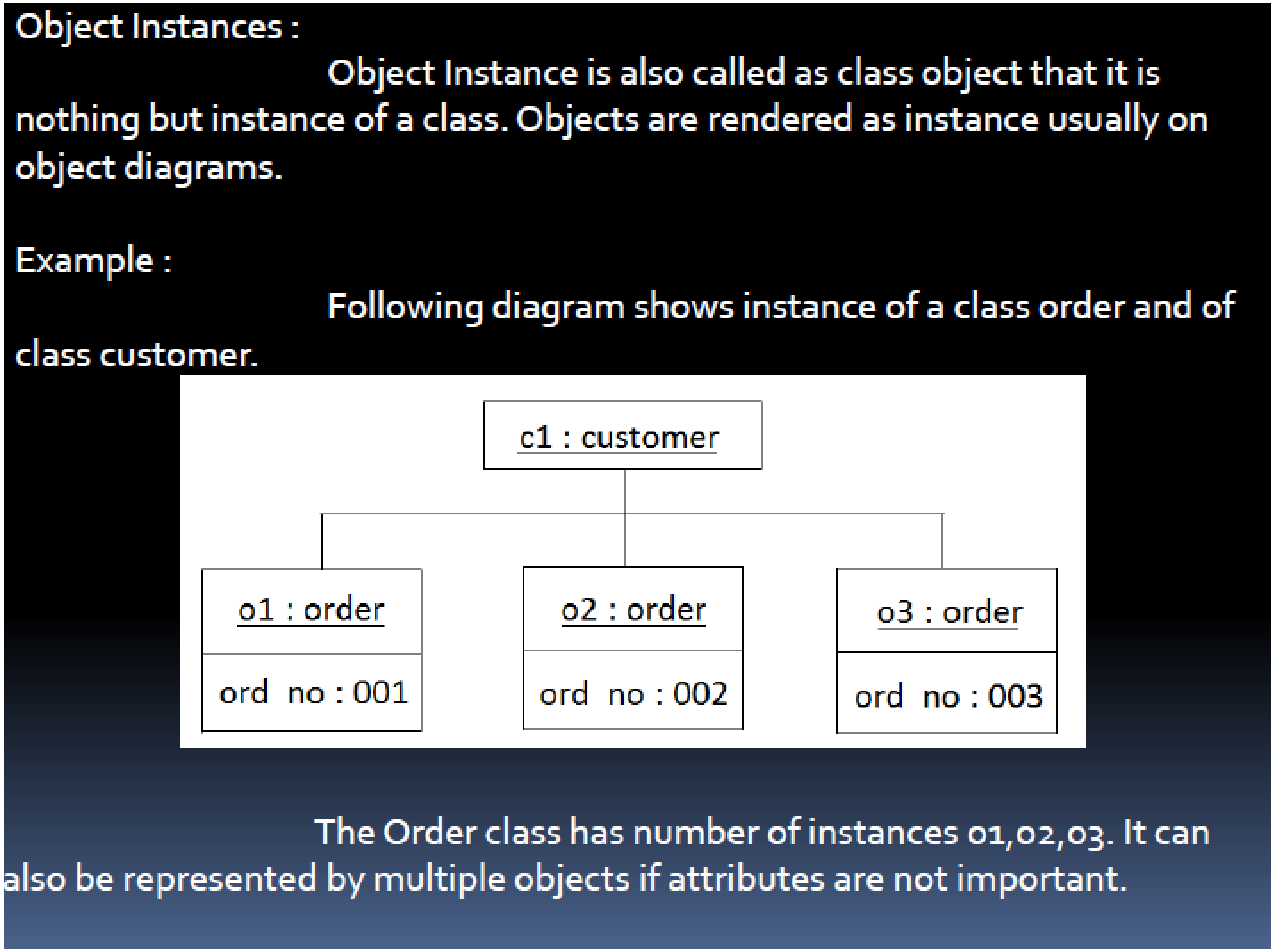
Relationship between objects can also be generalization or aggregation.

Generalization Symbol

Aggregation Symbol

Objects that fulfill more than 1 role can be self linked.

Object Diagram (contd..)



Object Instances :

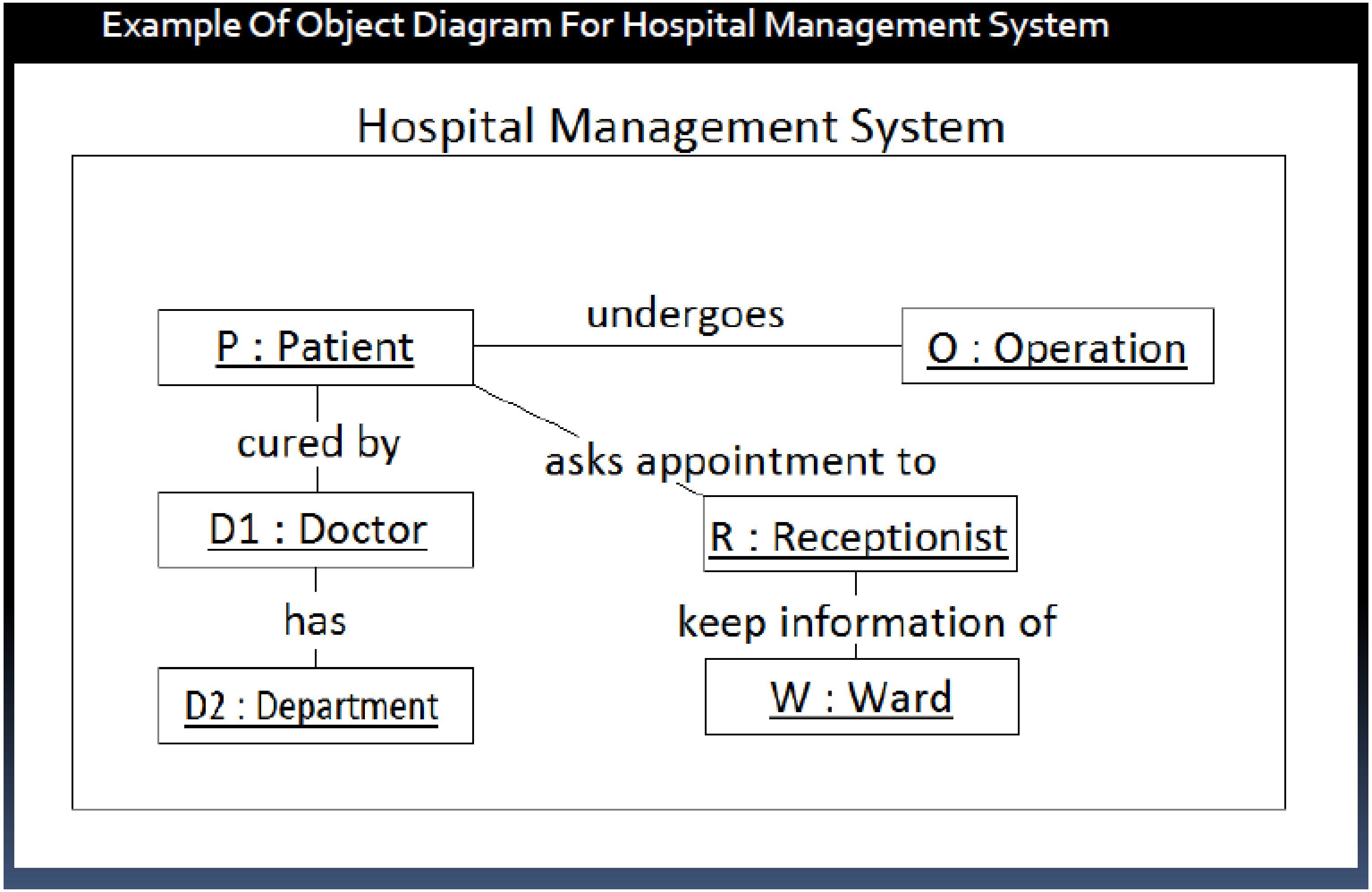
Object Instance is also called as class object that it is nothing but instance of a class. Objects are rendered as instance usually on object diagrams.

Example :

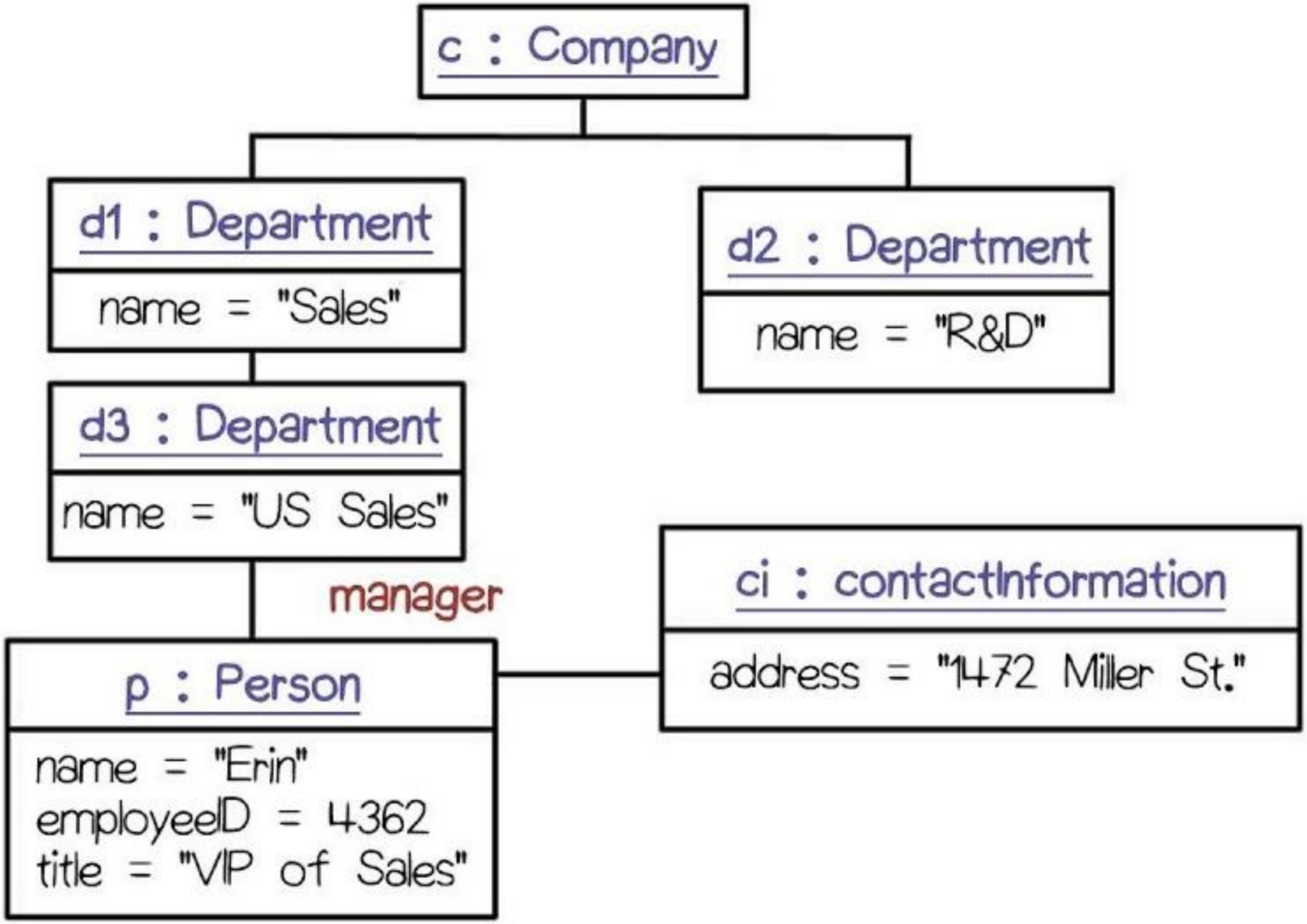
Following diagram shows instance of a class order and of class customer.

The Order class has number of instances o1,o2,o3. It can also be represented by multiple objects if attributes are not important.

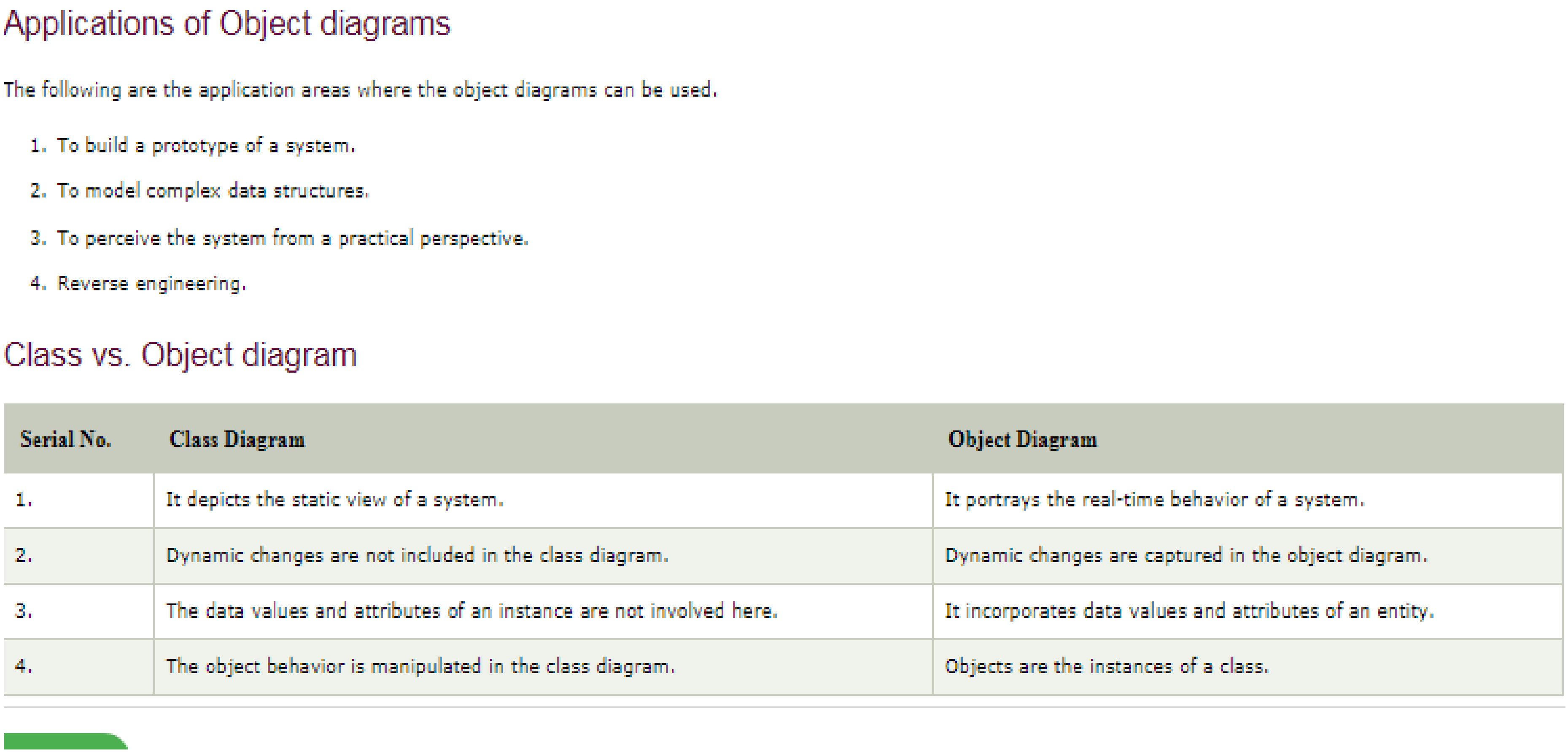
Object Diagram (contd..)



Object Diagram (Example)



Object Diagram (contd..)



Applications of Object diagrams

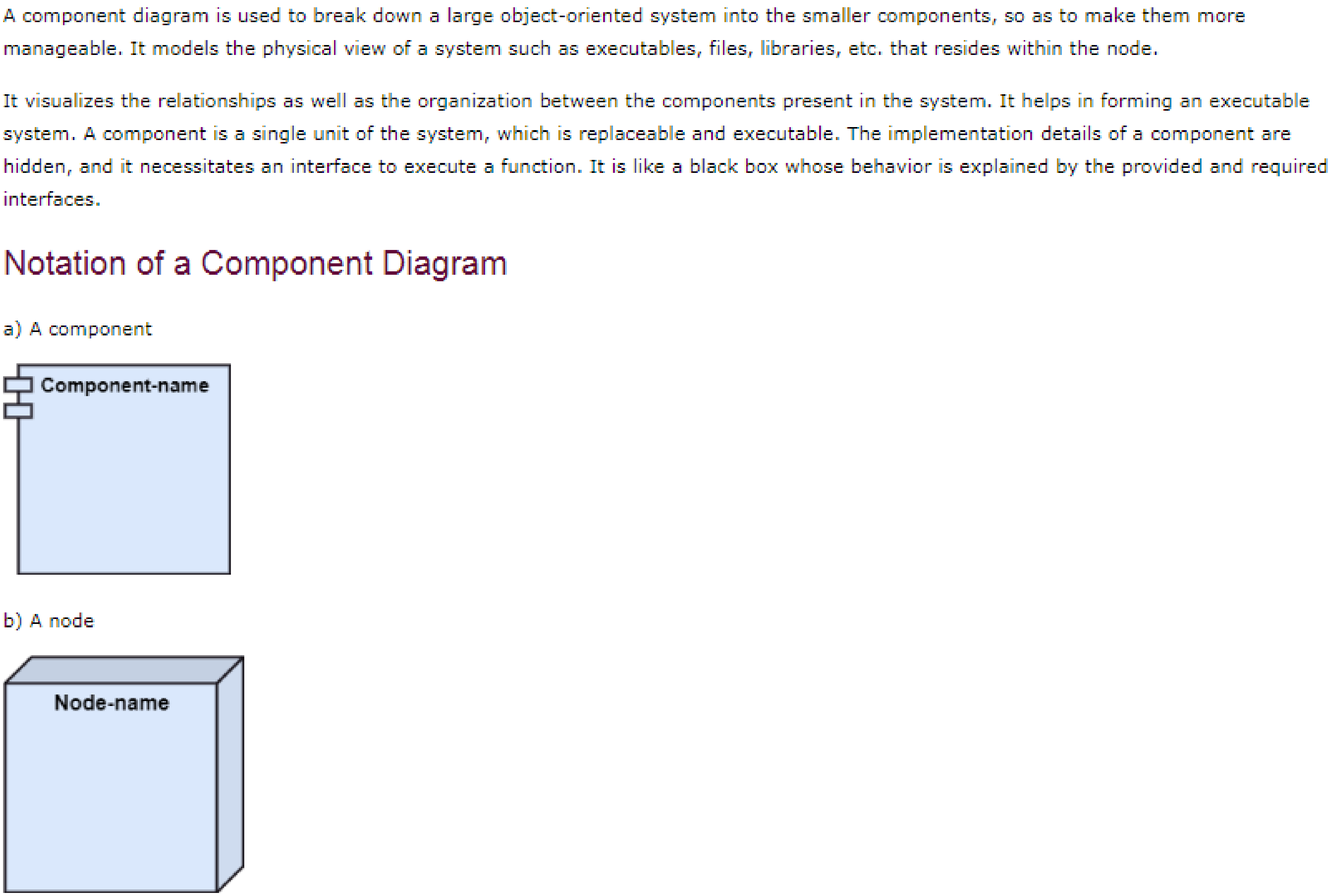
The following are the application areas where the object diagrams can be used.

1. To build a prototype of a system.
2. To model complex data structures.
3. To perceive the system from a practical perspective.
4. Reverse engineering.

Class vs. Object diagram

Serial No.	Class Diagram	Object Diagram
1.	It depicts the static view of a system.	It portrays the real-time behavior of a system.
2.	Dynamic changes are not included in the class diagram.	Dynamic changes are captured in the object diagram.
3.	The data values and attributes of an instance are not involved here.	It incorporates data values and attributes of an entity.
4.	The object behavior is manipulated in the class diagram.	Objects are the instances of a class.

Component diagram



A component diagram is used to break down a large object-oriented system into the smaller components, so as to make them more manageable. It models the physical view of a system such as executables, files, libraries, etc. that resides within the node.

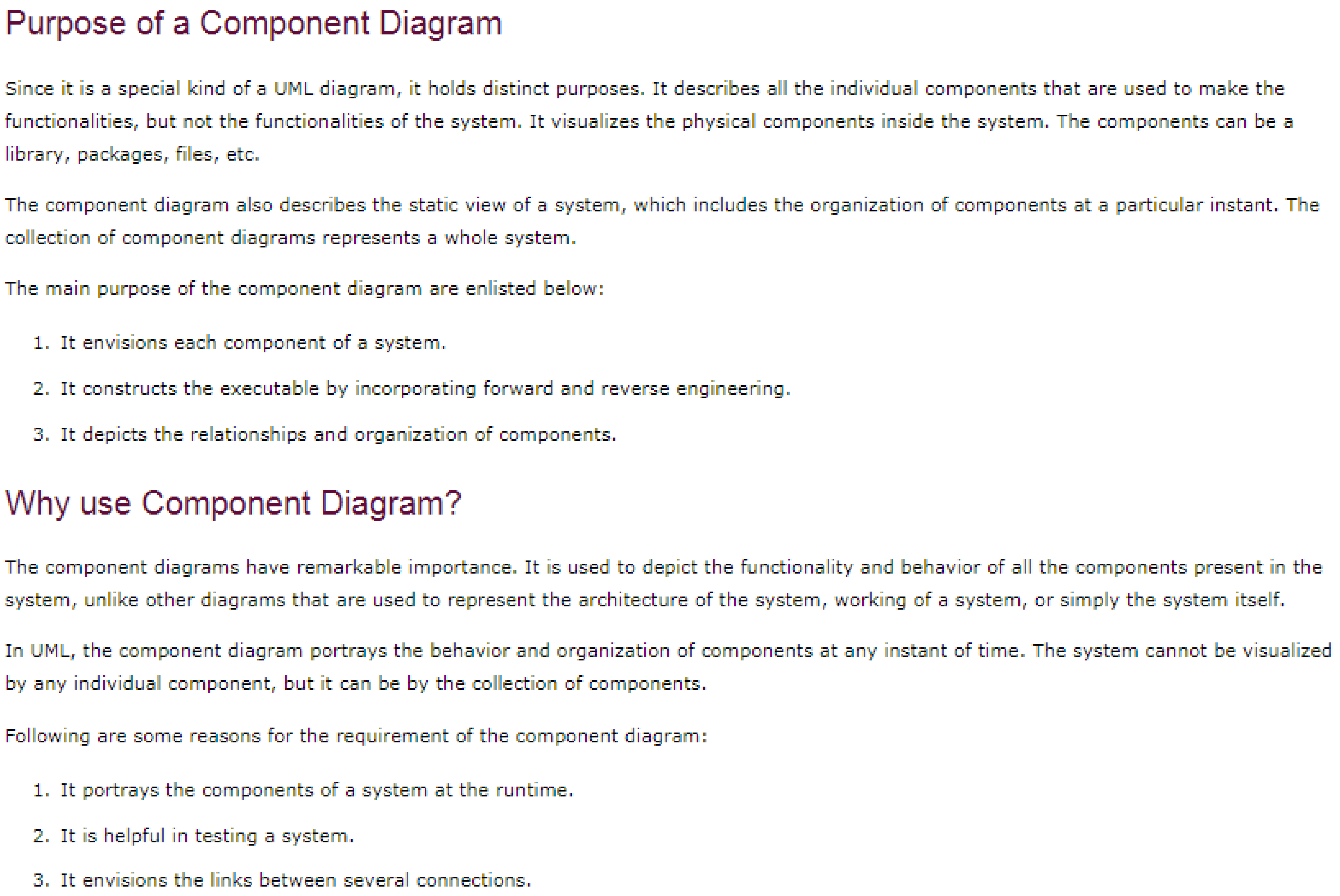
It visualizes the relationships as well as the organization between the components present in the system. It helps in forming an executable system. A component is a single unit of the system, which is replaceable and executable. The implementation details of a component are hidden, and it necessitates an interface to execute a function. It is like a black box whose behavior is explained by the provided and required interfaces.

Notation of a Component Diagram

a) A component

b) A node

Component diagram (contd..)



Purpose of a Component Diagram

Since it is a special kind of a UML diagram, it holds distinct purposes. It describes all the individual components that are used to make the functionalities, but not the functionalities of the system. It visualizes the physical components inside the system. The components can be a library, packages, files, etc.

The component diagram also describes the static view of a system, which includes the organization of components at a particular instant. The collection of component diagrams represents a whole system.

The main purpose of the component diagram are enlisted below:

1. It envisions each component of a system.
2. It constructs the executable by incorporating forward and reverse engineering.
3. It depicts the relationships and organization of components.

Why use Component Diagram?

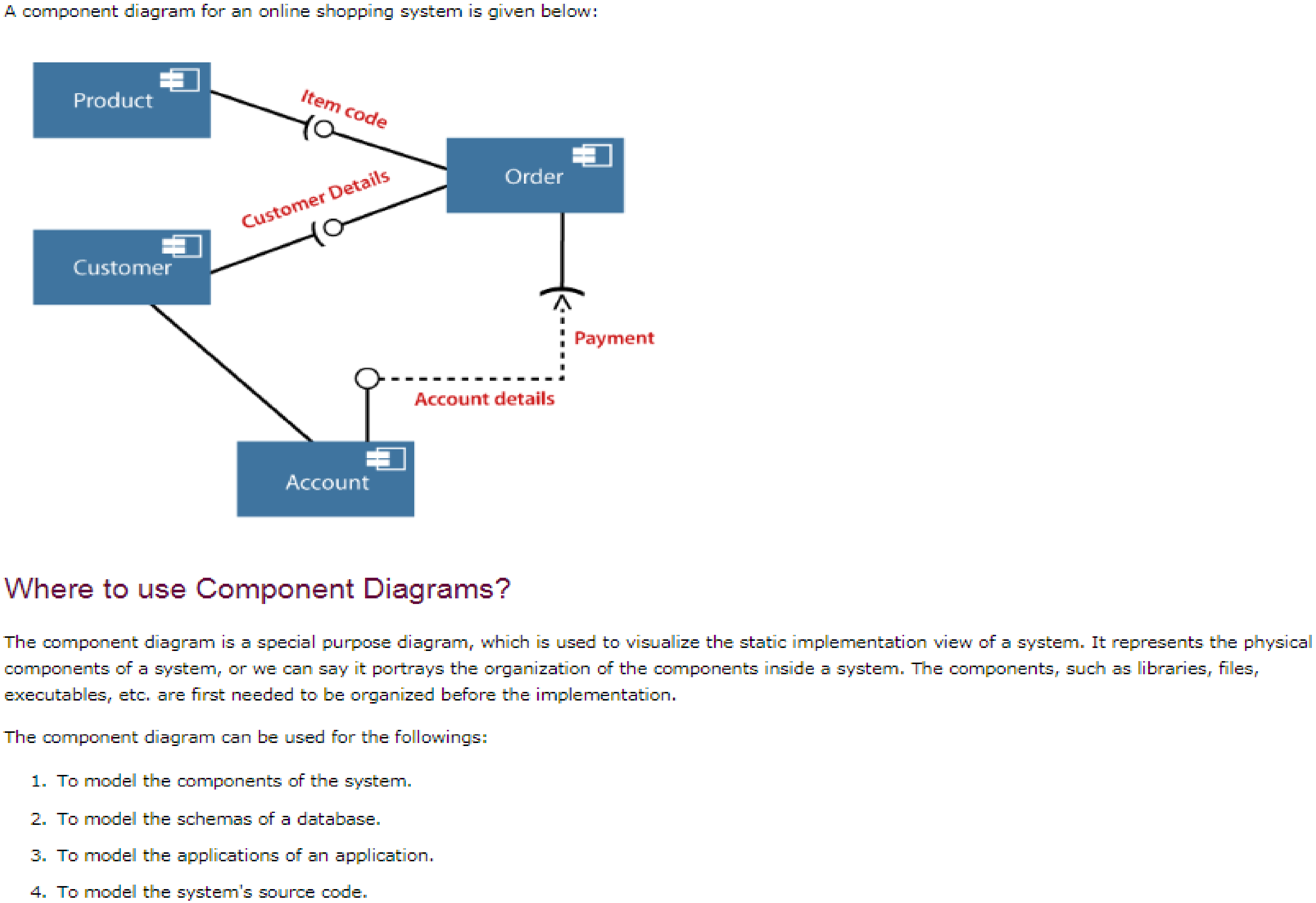
The component diagrams have remarkable importance. It is used to depict the functionality and behavior of all the components present in the system, unlike other diagrams that are used to represent the architecture of the system, working of a system, or simply the system itself.

In UML, the component diagram portrays the behavior and organization of components at any instant of time. The system cannot be visualized by any individual component, but it can be by the collection of components.

Following are some reasons for the requirement of the component diagram:

1. It portrays the components of a system at the runtime.
2. It is helpful in testing a system.
3. It envisions the links between several connections.

Component diagram (Example)



A component diagram for an online shopping system is given below:

Where to use Component Diagrams?

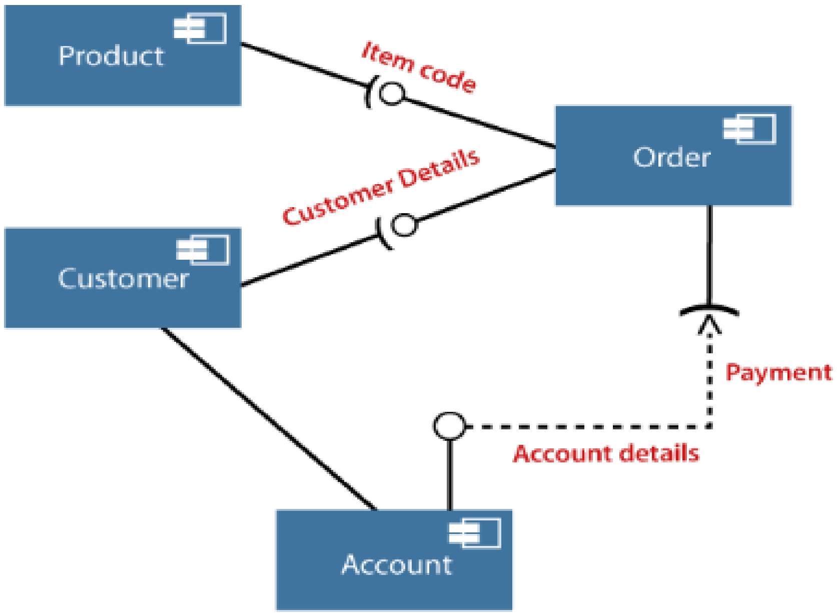
The component diagram is a special purpose diagram, which is used to visualize the static implementation view of a system. It represents the physical components of a system, or we can say it portrays the organization of the components inside a system. The components, such as libraries, files, executables, etc. are first needed to be organized before the implementation.

The component diagram can be used for the followings:

1. To model the components of the system.
2. To model the schemas of a database.
3. To model the applications of an application.
4. To model the system's source code.

Component diagram (Example: Online shopping system)

A component diagram for an online shopping system is given below:



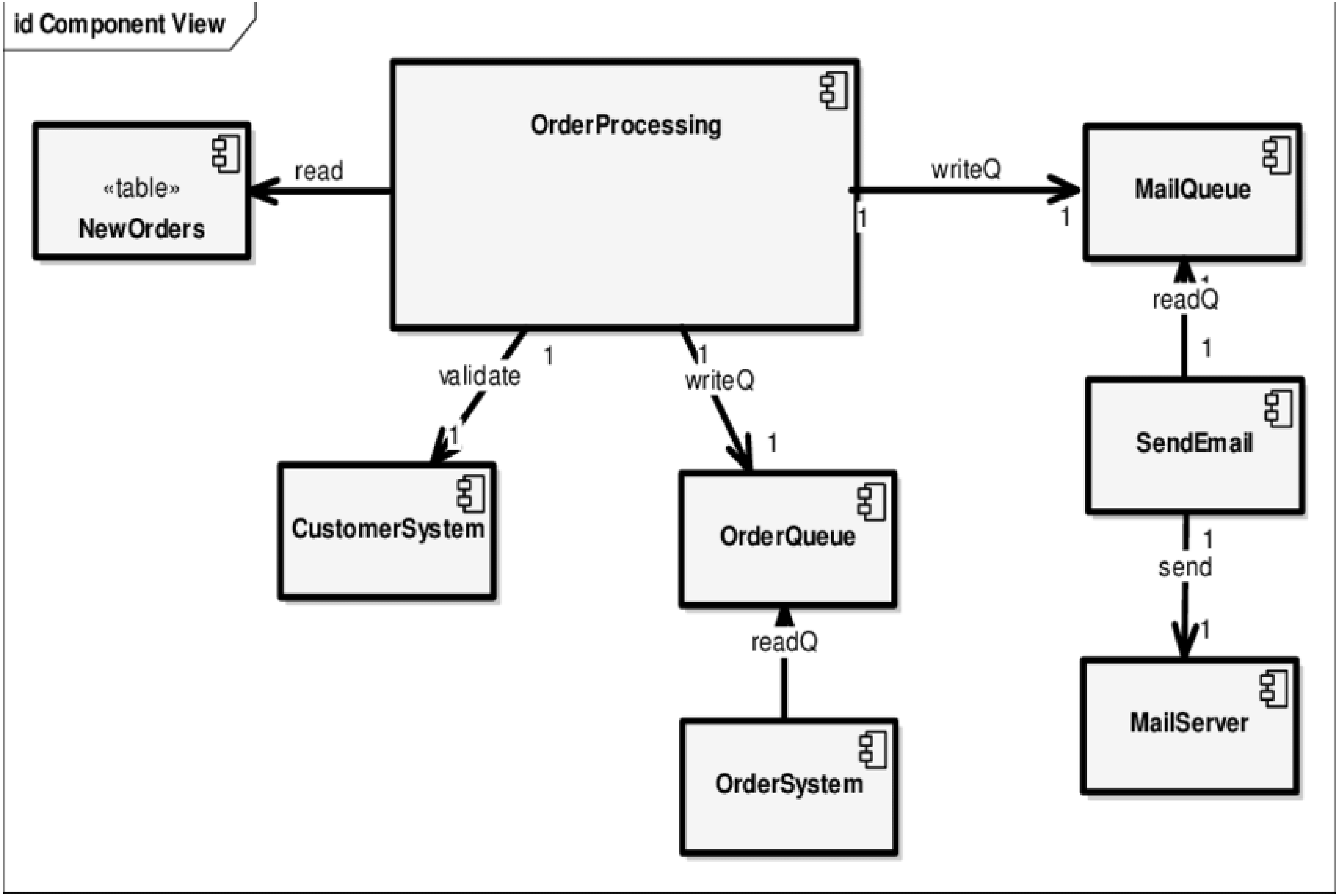
Where to use Component Diagrams?

The component diagram is a special purpose diagram, which is used to visualize the static implementation view of a system. It represents the physical components of a system, or we can say it portrays the organization of the components inside a system. The components, such as libraries, files, executables, etc. are first needed to be organized before the implementation.

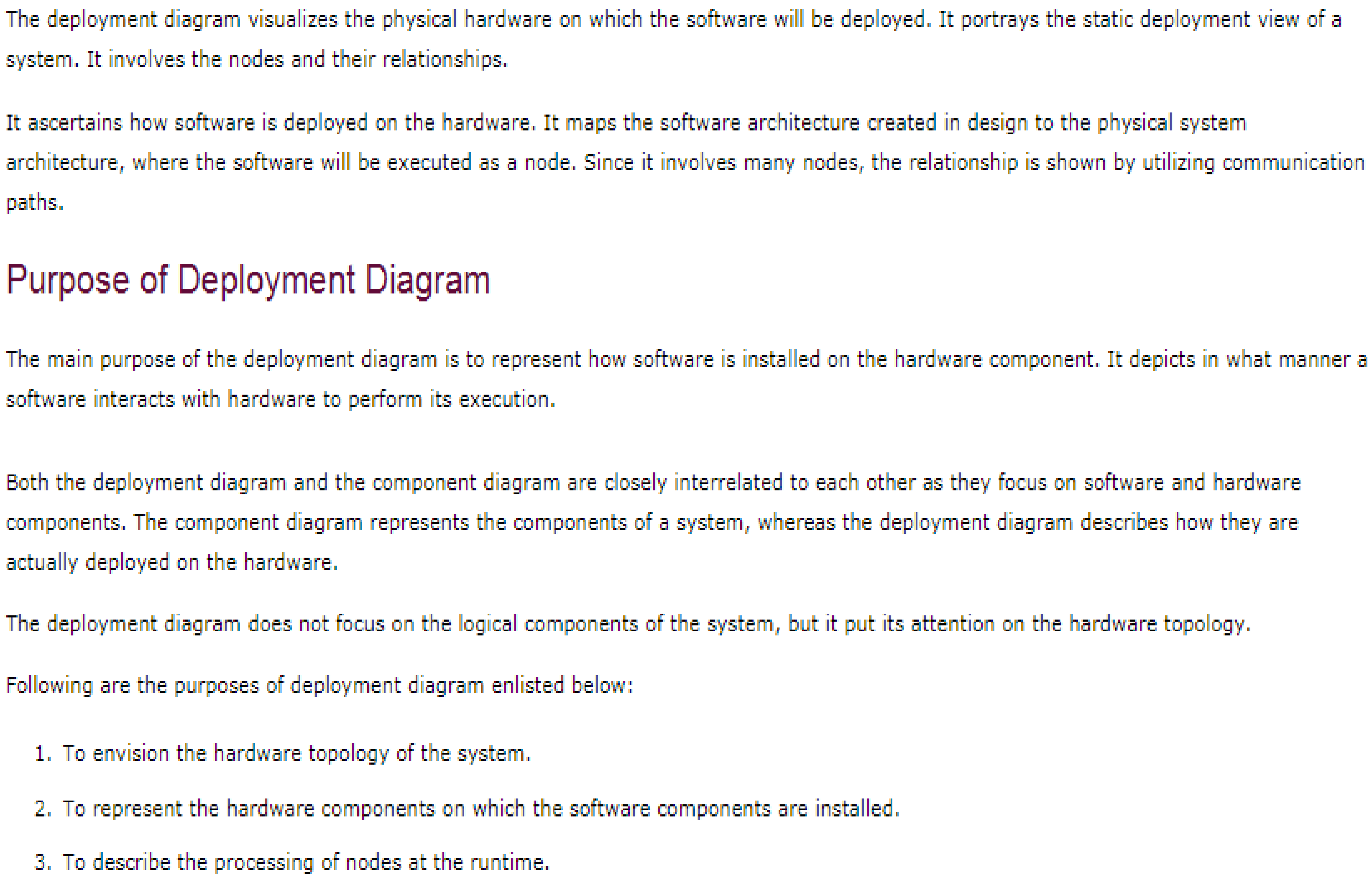
The component diagram can be used for the followings:

1. To model the components of the system.
2. To model the schemas of a database.
3. To model the applications of an application.
4. To model the system's source code.

Component diagram (Example : Order Processing)



Deployment diagram



The deployment diagram visualizes the physical hardware on which the software will be deployed. It portrays the static deployment view of a system. It involves the nodes and their relationships.

It ascertains how software is deployed on the hardware. It maps the software architecture created in design to the physical system architecture, where the software will be executed as a node. Since it involves many nodes, the relationship is shown by utilizing communication paths.

Purpose of Deployment Diagram

The main purpose of the deployment diagram is to represent how software is installed on the hardware component. It depicts in what manner a software interacts with hardware to perform its execution.

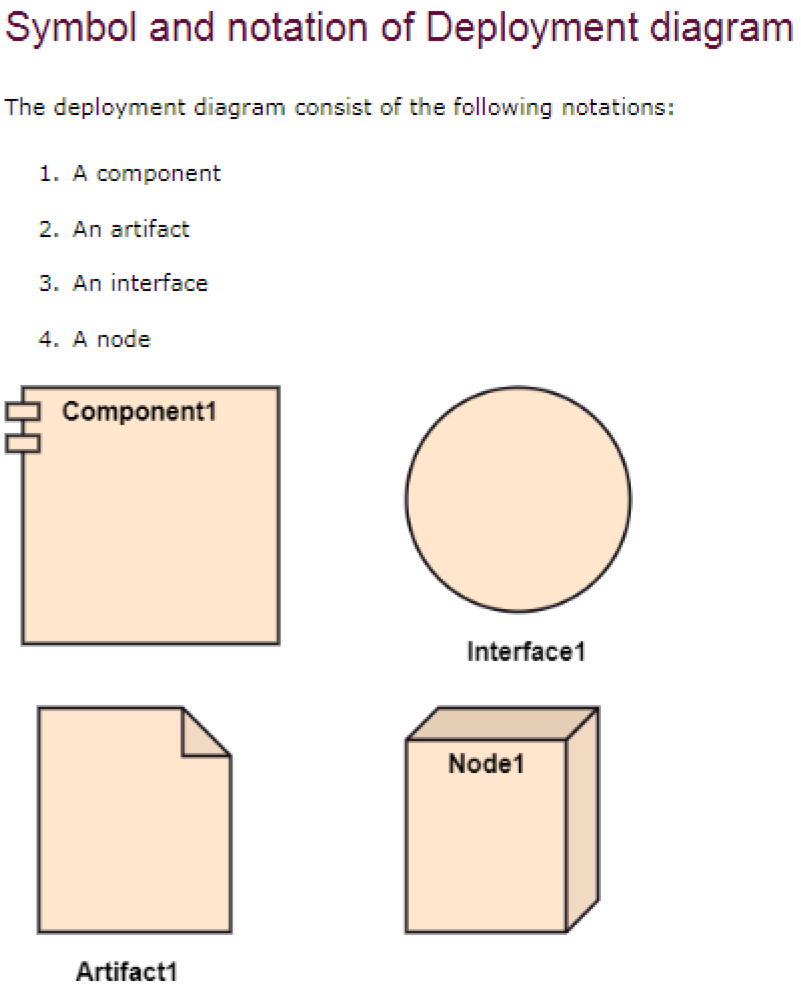
Both the deployment diagram and the component diagram are closely interrelated to each other as they focus on software and hardware components. The component diagram represents the components of a system, whereas the deployment diagram describes how they are actually deployed on the hardware.

The deployment diagram does not focus on the logical components of the system, but it put its attention on the hardware topology.

Following are the purposes of deployment diagram enlisted below:

1. To envision the hardware topology of the system.
2. To represent the hardware components on which the software components are installed.
3. To describe the processing of nodes at the runtime.

Deployment diagram (contd..)

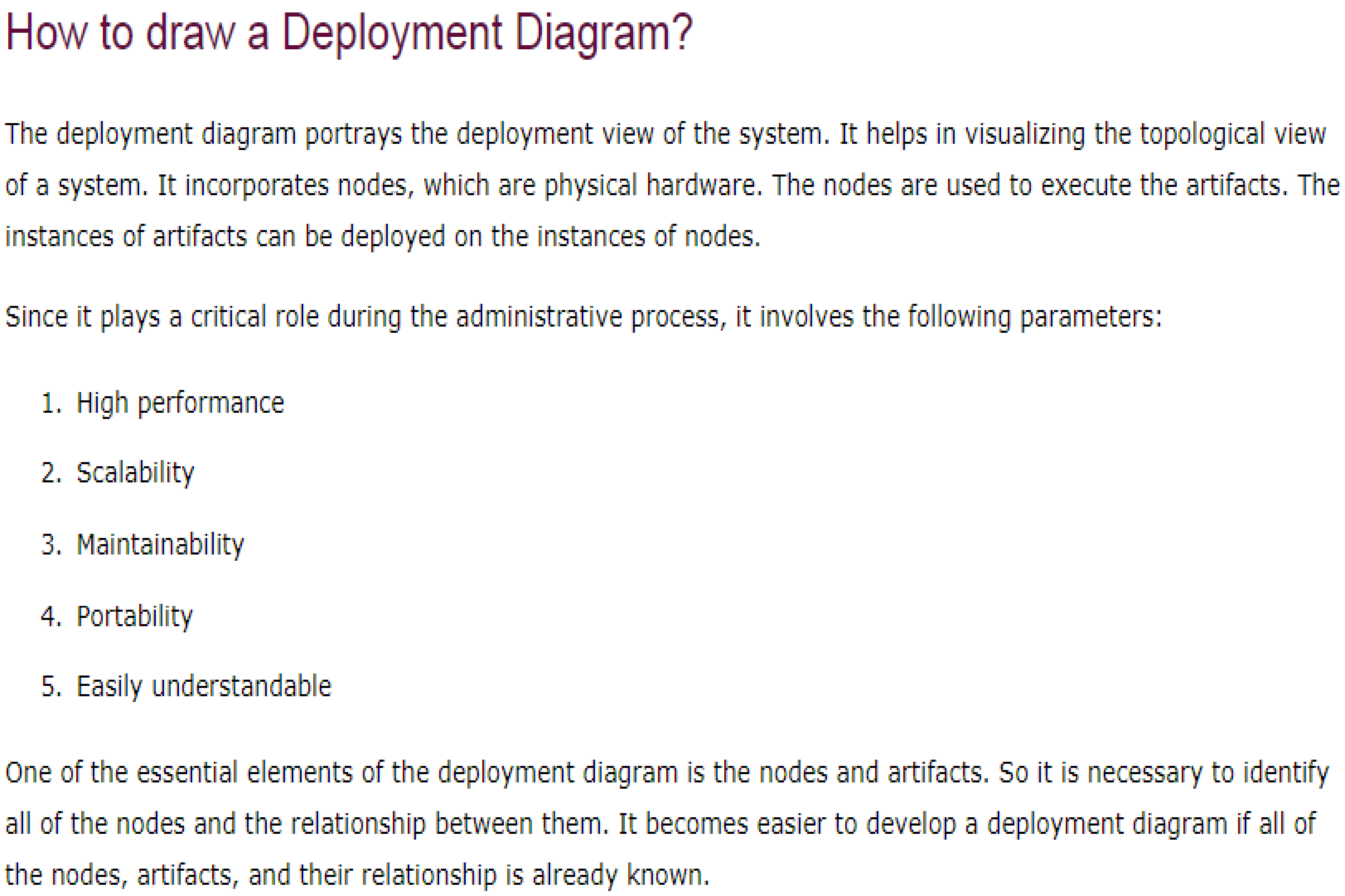


Symbol and notation of Deployment diagram

The deployment diagram consist of the following notations:

1. A component
2. An artifact
3. An interface
4. A node

Deployment diagram (contd..)



How to draw a Deployment Diagram?

The deployment diagram portrays the deployment view of the system. It helps in visualizing the topological view of a system. It incorporates nodes, which are physical hardware. The nodes are used to execute the artifacts. The instances of artifacts can be deployed on the instances of nodes.

Since it plays a critical role during the administrative process, it involves the following parameters:

1. High performance
2. Scalability
3. Maintainability
4. Portability
5. Easily understandable

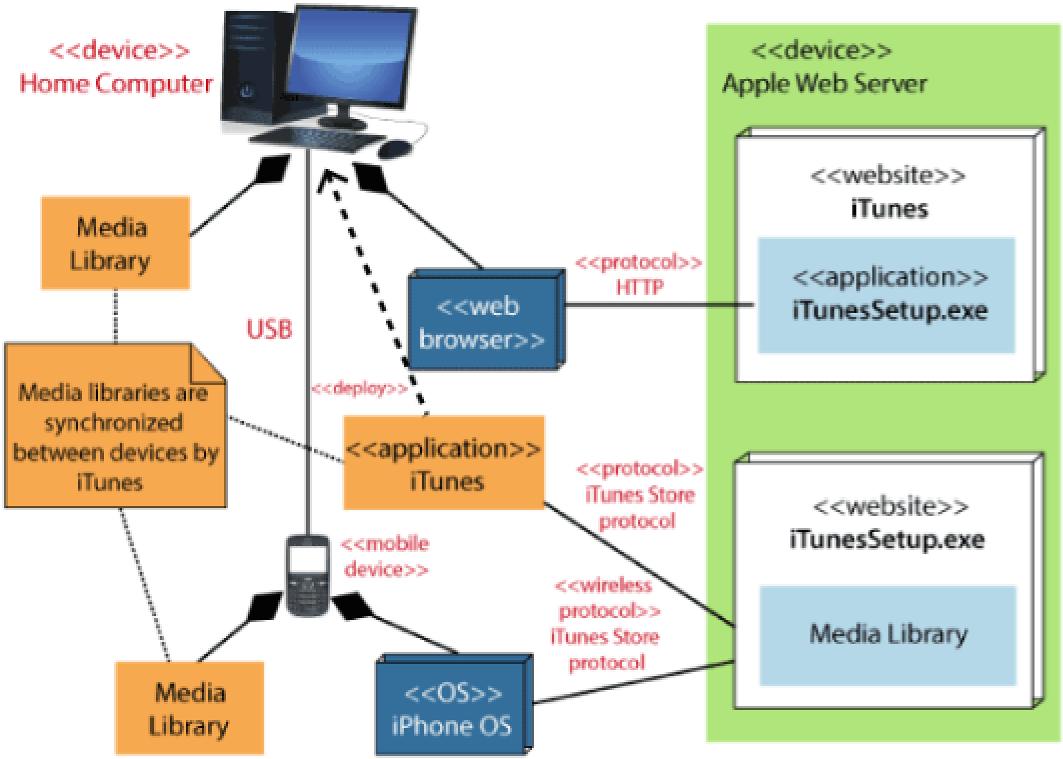
One of the essential elements of the deployment diagram is the nodes and artifacts. So it is necessary to identify all of the nodes and the relationship between them. It becomes easier to develop a deployment diagram if all of the nodes, artifacts, and their relationship is already known.

Deployment diagram (Example)

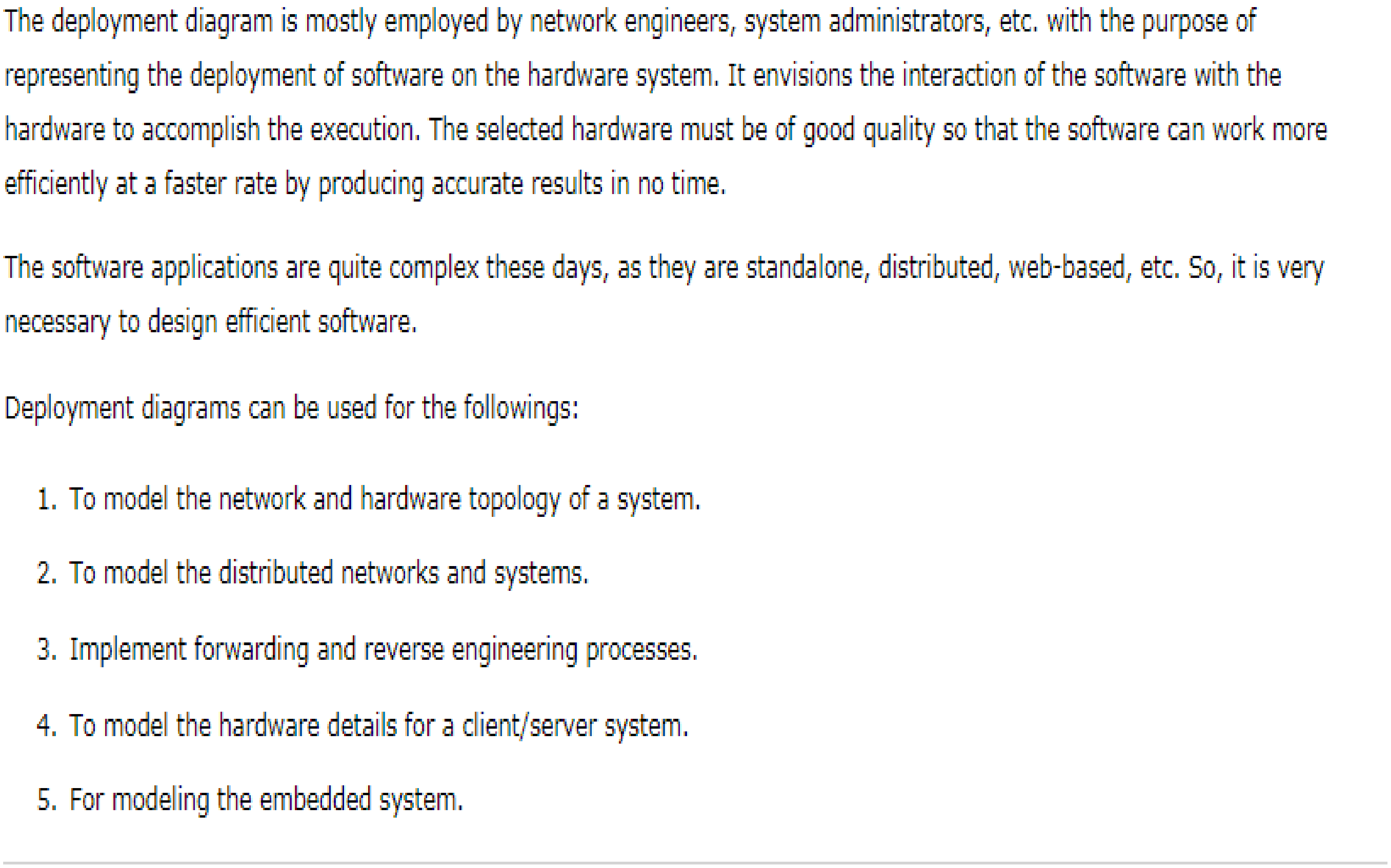
A deployment diagram for the Apple iTunes application is given below.

The iTunes setup can be downloaded from the iTunes website, and also it can be installed on the home computer. Once the installation and the registration are done, iTunes application can easily interconnect with the Apple iTunes store. Users can purchase and download music, video, TV serials, etc. and cache it in the media library.

Devices like Apple iPod Touch and Apple iPhone can update its own media library from the computer with iTunes with the help of USB or simply by downloading media directly from the Apple iTunes store using wireless protocols, for example; Wi-Fi, 3G, or EDGE.



Deployment diagram (Example)



The deployment diagram is mostly employed by network engineers, system administrators, etc. with the purpose of representing the deployment of software on the hardware system. It envisions the interaction of the software with the hardware to accomplish the execution. The selected hardware must be of good quality so that the software can work more efficiently at a faster rate by producing accurate results in no time.

The software applications are quite complex these days, as they are standalone, distributed, web-based, etc. So, it is very necessary to design efficient software.

Deployment diagrams can be used for the followings:

1. To model the network and hardware topology of a system.
2. To model the distributed networks and systems.
3. Implement forwarding and reverse engineering processes.
4. To model the hardware details for a client/server system.
5. For modeling the embedded system.

Interaction diagram

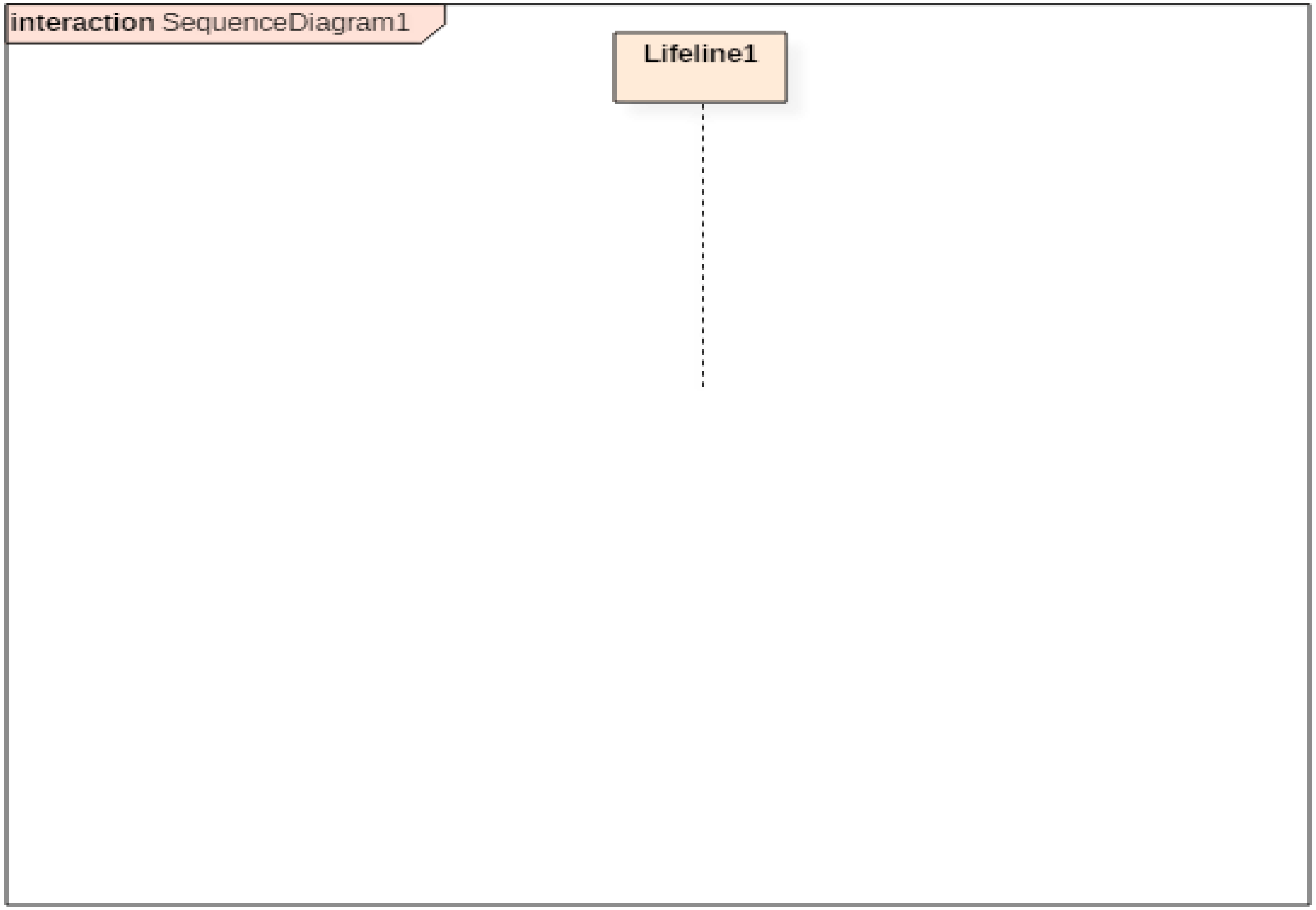
INTERACTION DIAGRAM are used in UML to establish communication between objects. It does not manipulate the data associated with the particular communication path.

•Interaction diagrams mostly focus on message passing and how these messages make up one functionality of a system.

•Interaction diagrams are designed to display how the objects will realize the particular requirements of a system.

•The critical component in an interaction diagram is **lifeline and messages.**

Notation of Interaction diagram



Notation of an interaction diagram

Types of Interaction diagrams

Following are the different types of interaction diagrams defined in UML:

•Sequence diagram
•Collaboration diagram
•Timing diagram

•**Sequence diagram:** 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.

•**Collaboration diagram:** The collaboration diagram is also called as a communication diagram. The purpose of a collaboration diagram is to emphasize structural aspects of a system, i.e., how various lifelines in the system connects.

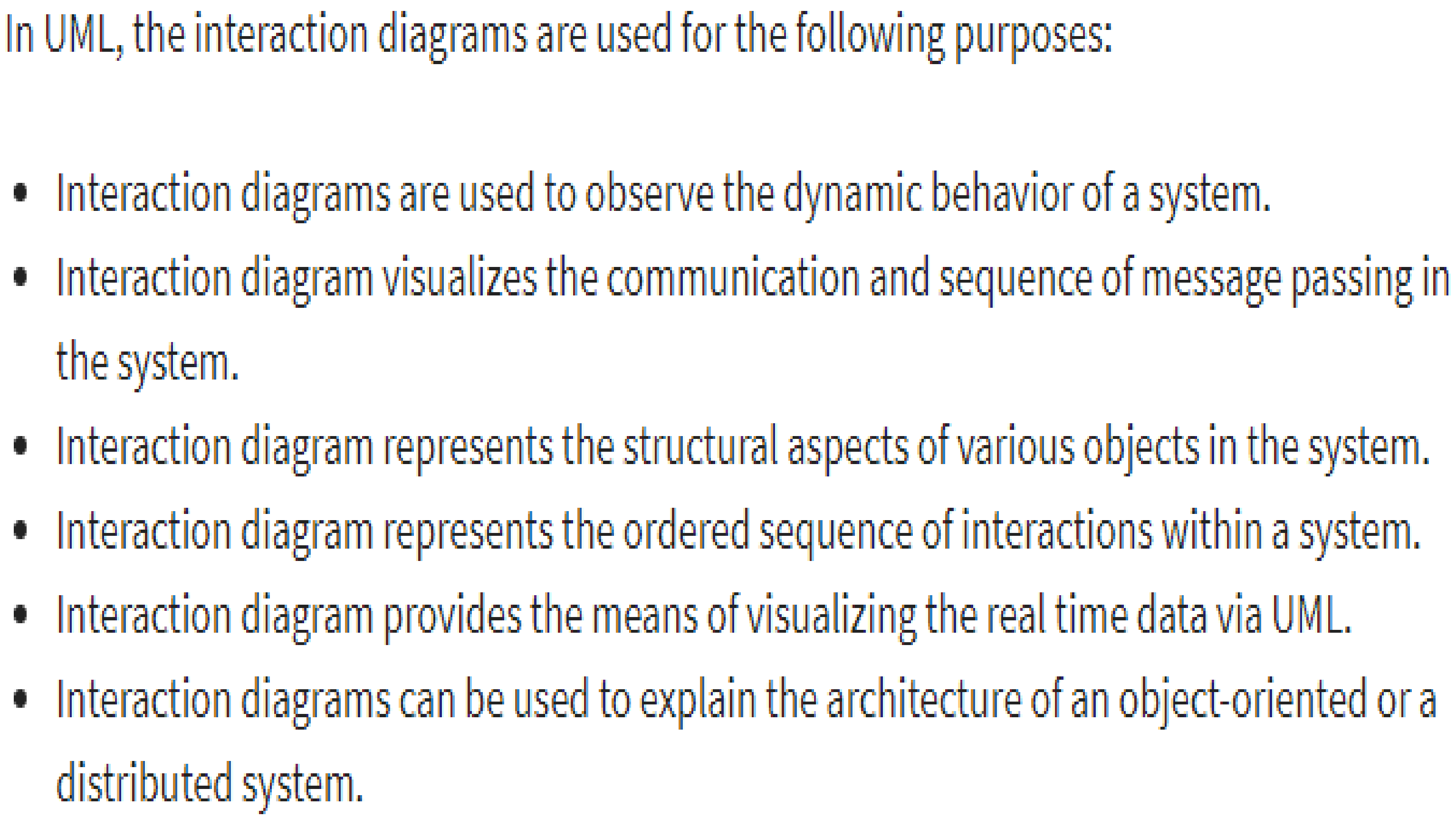
•**Timing diagram:** Timing diagrams focus on the instance at which a message is sent from one object to another object.

Terminology of Interaction diagram

An interaction diagram contains lifelines, messages, operators, state invariants and constraints.

1. **Lifelines:** A lifeline represents a single participant in an interaction. It describes how an instance of a specific classifier participates in the interaction.
2. **Messages**: A message is a specific type of communication between two lifelines in an interaction. A message involves following activities,
   - A call message which is used to call an operation.
   - A message to create an instance.
   - A message to destroy an instance.
   - For sending a signal.
3. **Operator:** An operator specifies an operation on how the operands are going to be executed. The operators in UML supports operations on data in the form of branching as well as an iteration.
4. **State invariants**: When an instance or a lifeline receives a message, it can cause it to change the state. A state is a condition or a situation during a lifetime of an object at which it satisfies some constraint, performs some operations, and waits for some event.

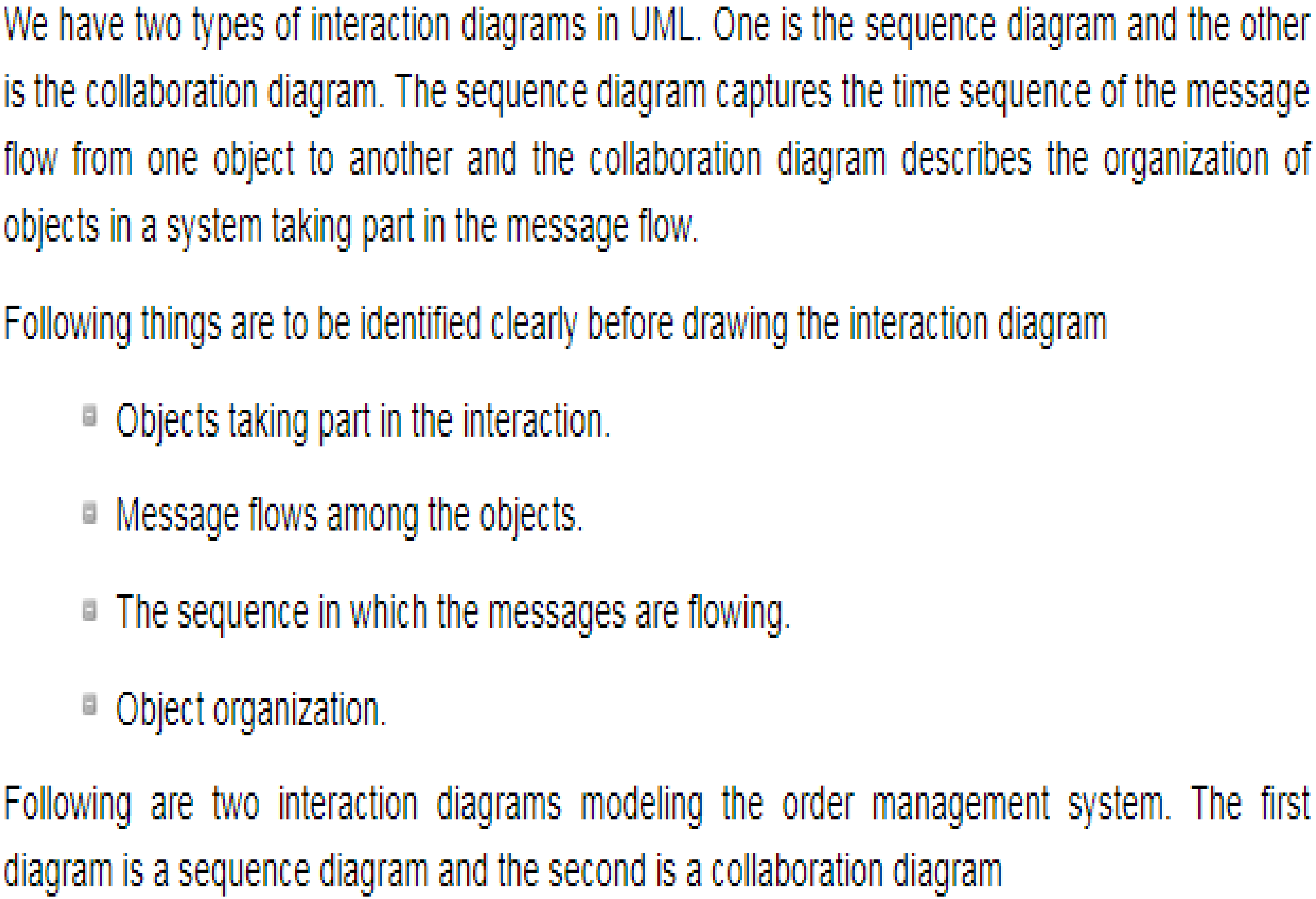
Purpose of Interaction diagram



In UML, the interaction diagrams are used for the following purposes:

- Interaction diagrams are used to observe the dynamic behavior of a system.
- Interaction diagram visualizes the communication and sequence of message passing in the system.
- Interaction diagram represents the structural aspects of various objects in the system.
- Interaction diagram represents the ordered sequence of interactions within a system.
- Interaction diagram provides the means of visualizing the real time data via UML.
- Interaction diagrams can be used to explain the architecture of an object-oriented or a distributed system.

How to draw an Interaction diagram?



We have two types of interaction diagrams in UML. One is the sequence diagram and the other is the collaboration diagram. The sequence diagram captures the time sequence of the message flow from one object to another and the collaboration diagram describes the organization of objects in a system taking part in the message flow.

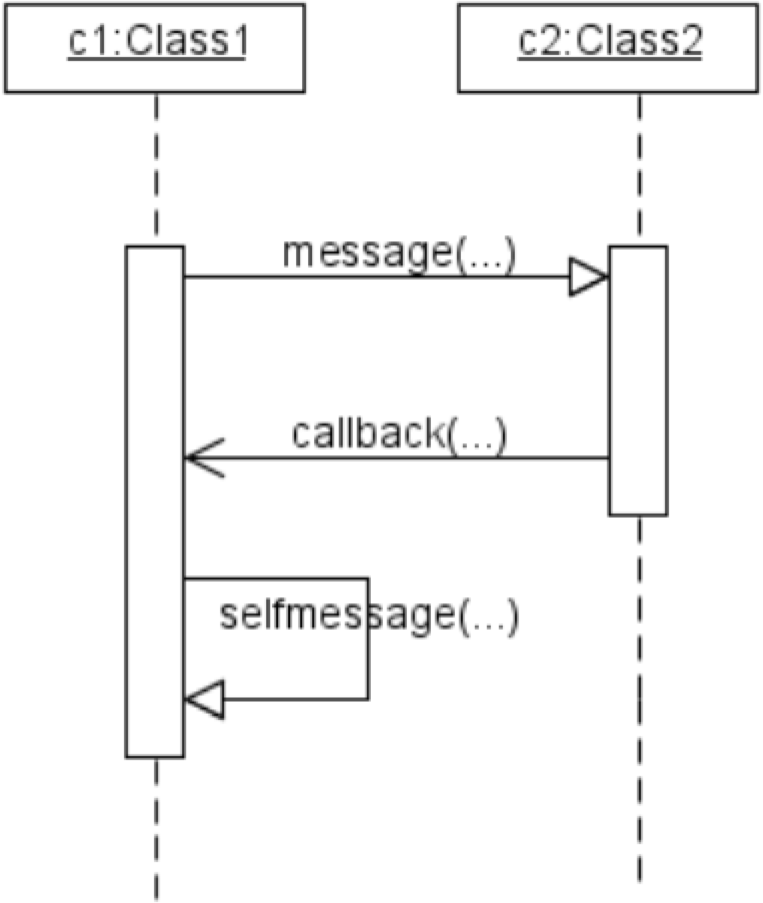
Following things are to be identified clearly before drawing the interaction diagram

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

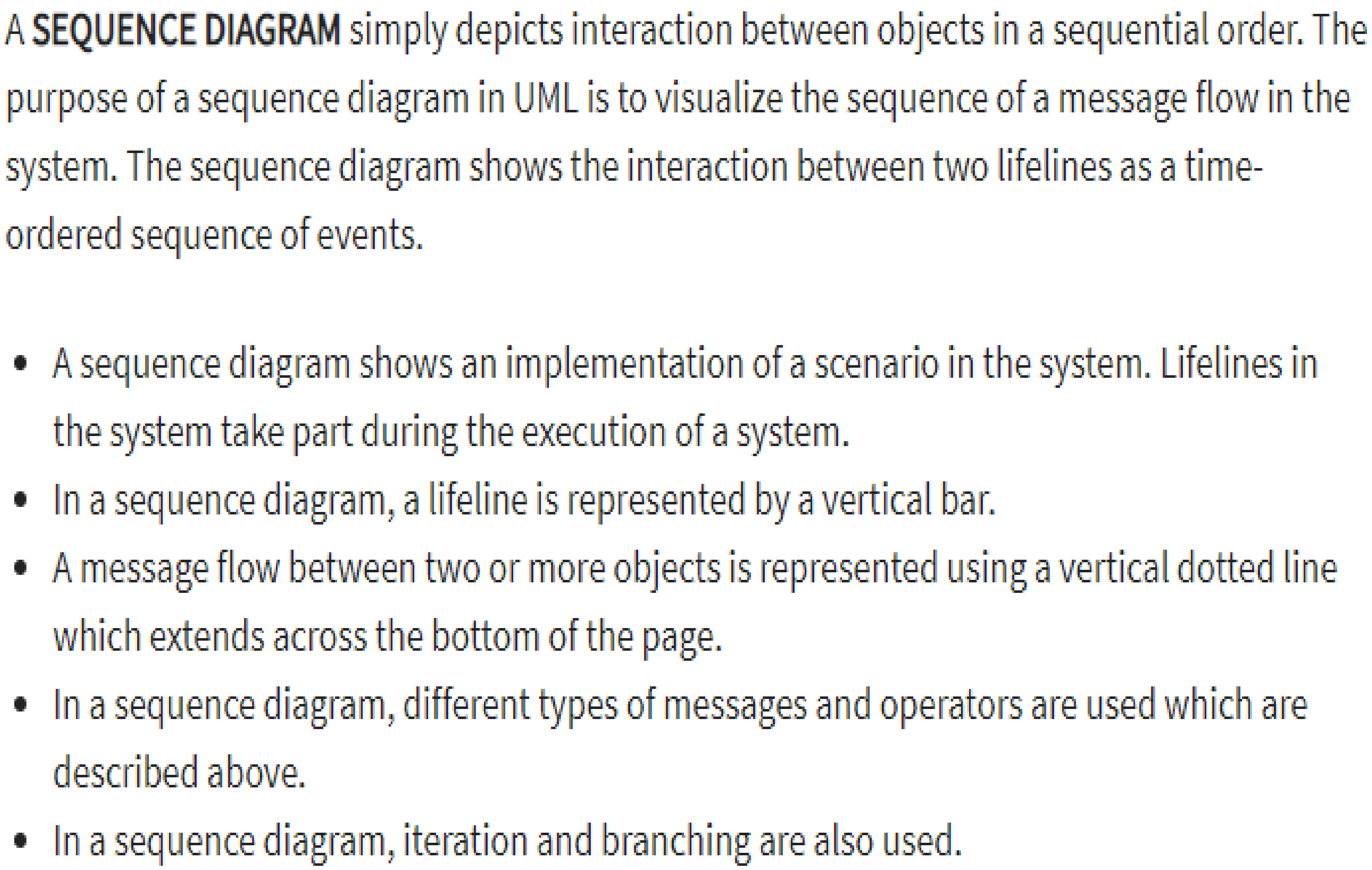
Following are two interaction diagrams modeling the order management system. The first diagram is a sequence diagram and the second is a collaboration diagram

Sequence diagram

- Sequence diagrams illustrate the sequence of actions that occur in a system.
- They are composed of 2 elements
  - Object
  - Messages



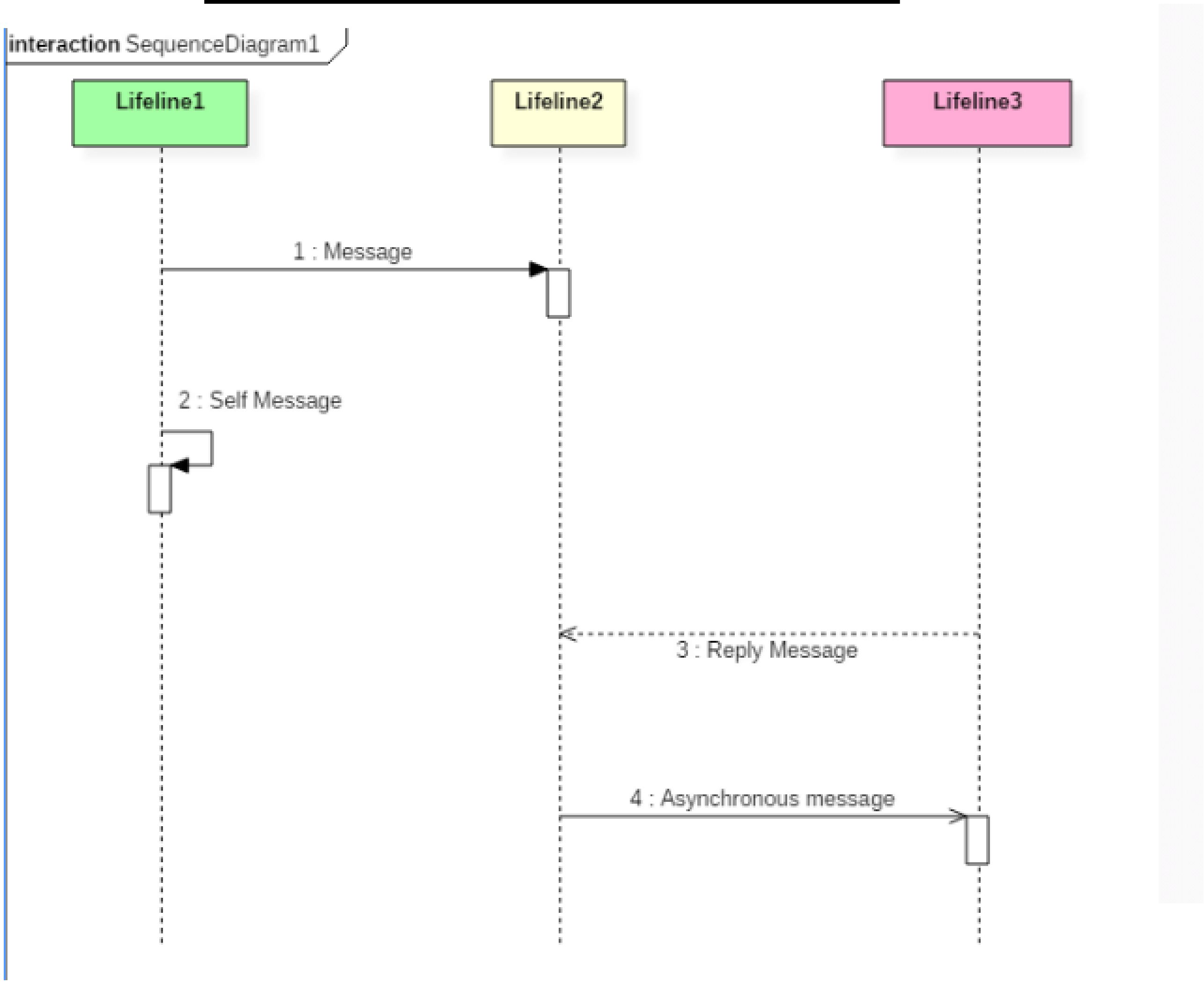
Sequence diagram



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.

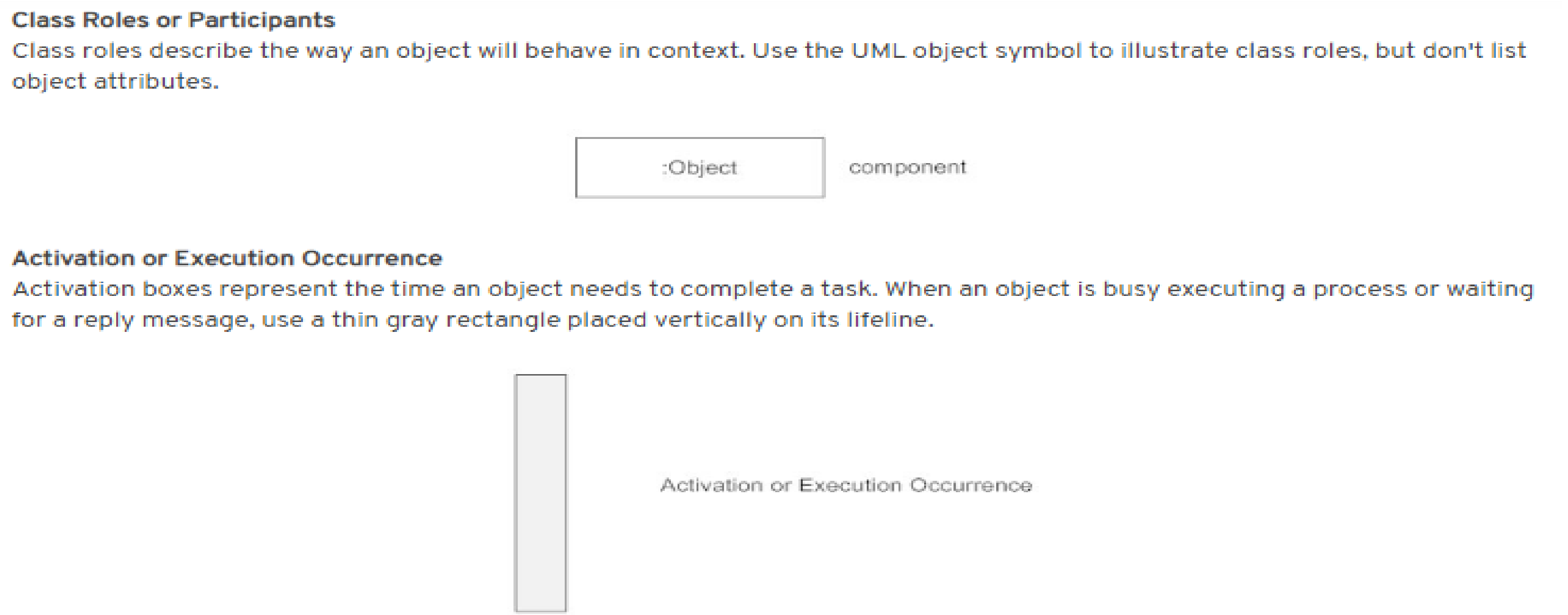
- A sequence diagram shows an implementation of a scenario in the system. Lifelines in the system take part during the execution of a system.
- In a sequence diagram, a lifeline is represented by a vertical bar.
- A message flow between two or more objects is represented using a vertical dotted line 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.

Sequence diagram (contd..)



The above sequence diagram contains lifeline notations and notation of various messages used in a sequence diagram such as a create, reply, asynchronous message, etc.

Basic Notations for Sequence diagram

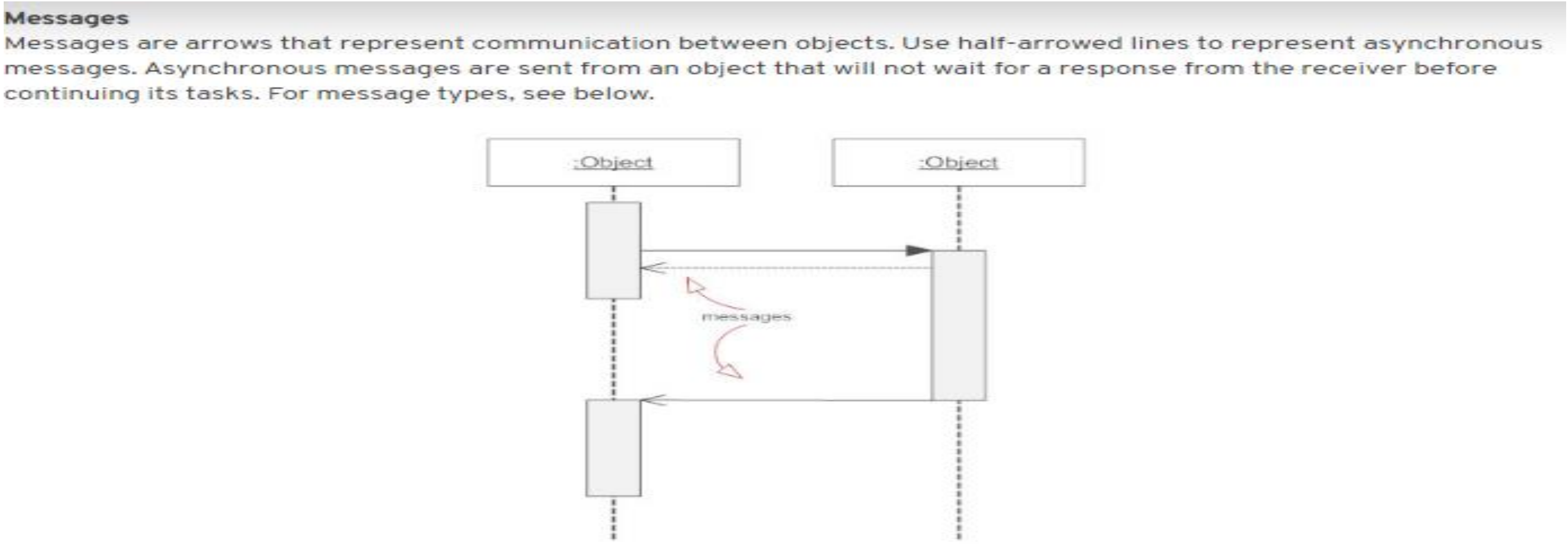


Class Roles or Participants

Class roles describe the way an object will behave in context. Use the UML object symbol to illustrate class roles, but don't list object attributes.

Activation or Execution Occurrence

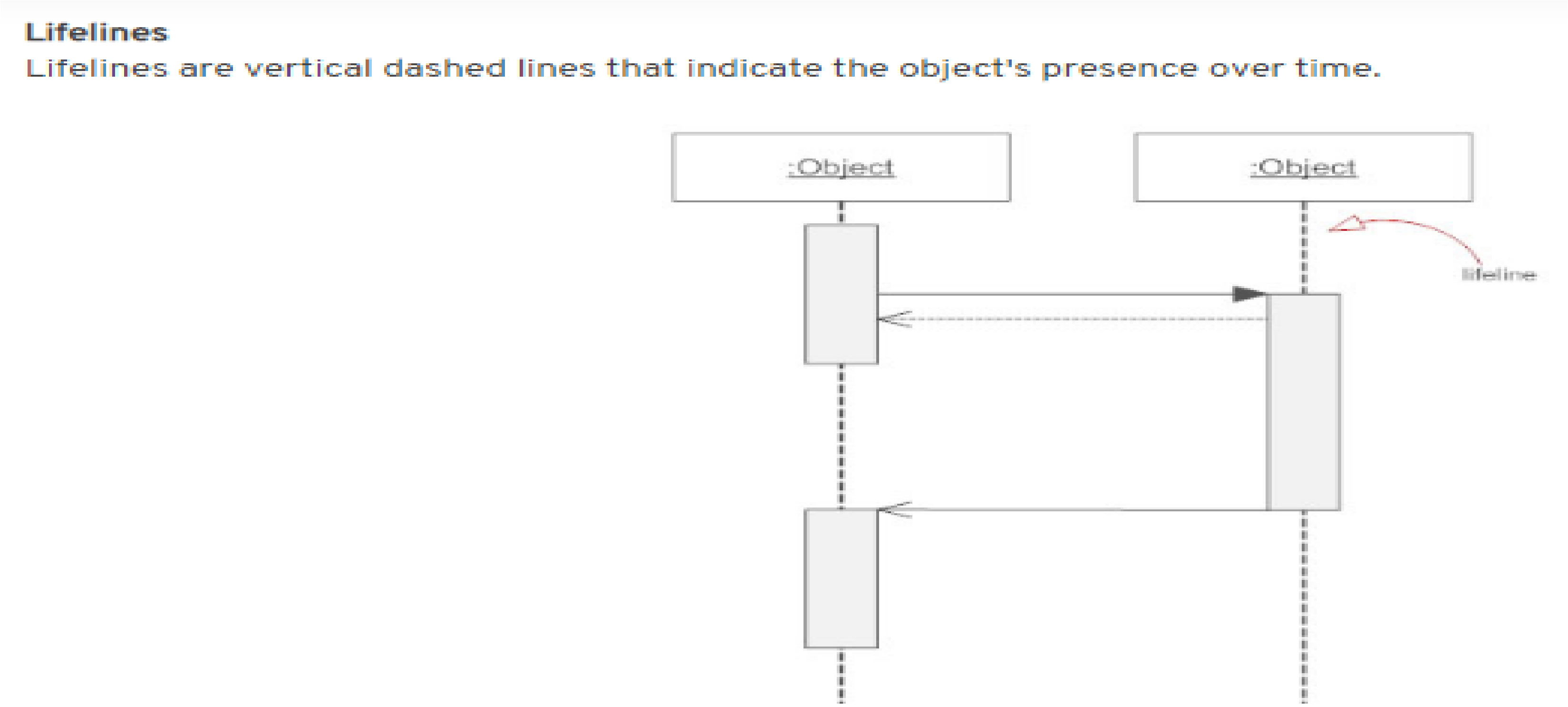
Activation boxes represent the time an object needs to complete a task. When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.



Messages

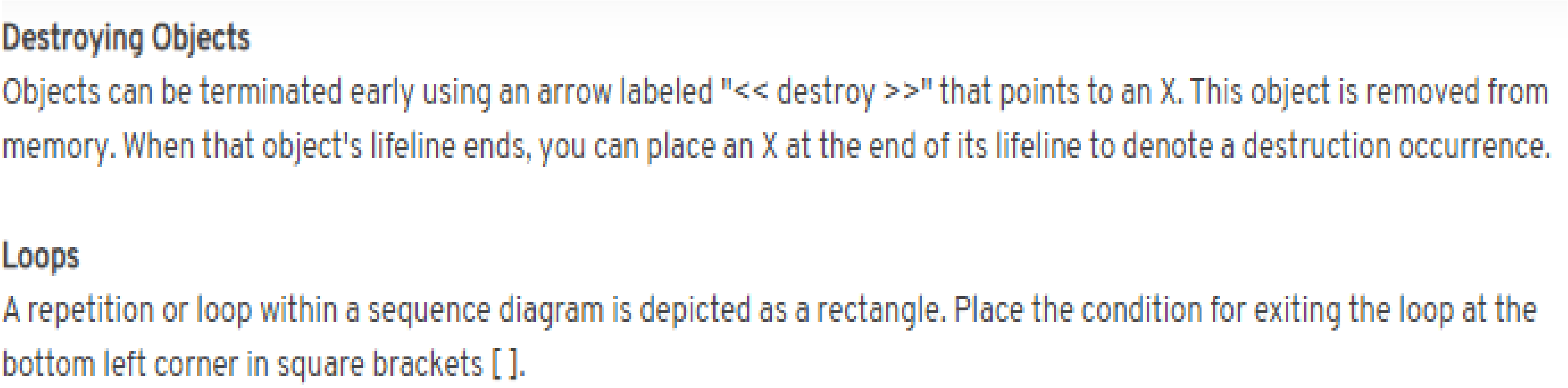
Messages are arrows that represent communication between objects. Use half-arrowed lines to represent asynchronous messages. Asynchronous messages are sent from an object that will not wait for a response from the receiver before continuing its tasks. For message types, see below.

Basic Notations for Sequence diagram (contd..)



Lifelines

Lifelines are vertical dashed lines that indicate the object's presence over time.



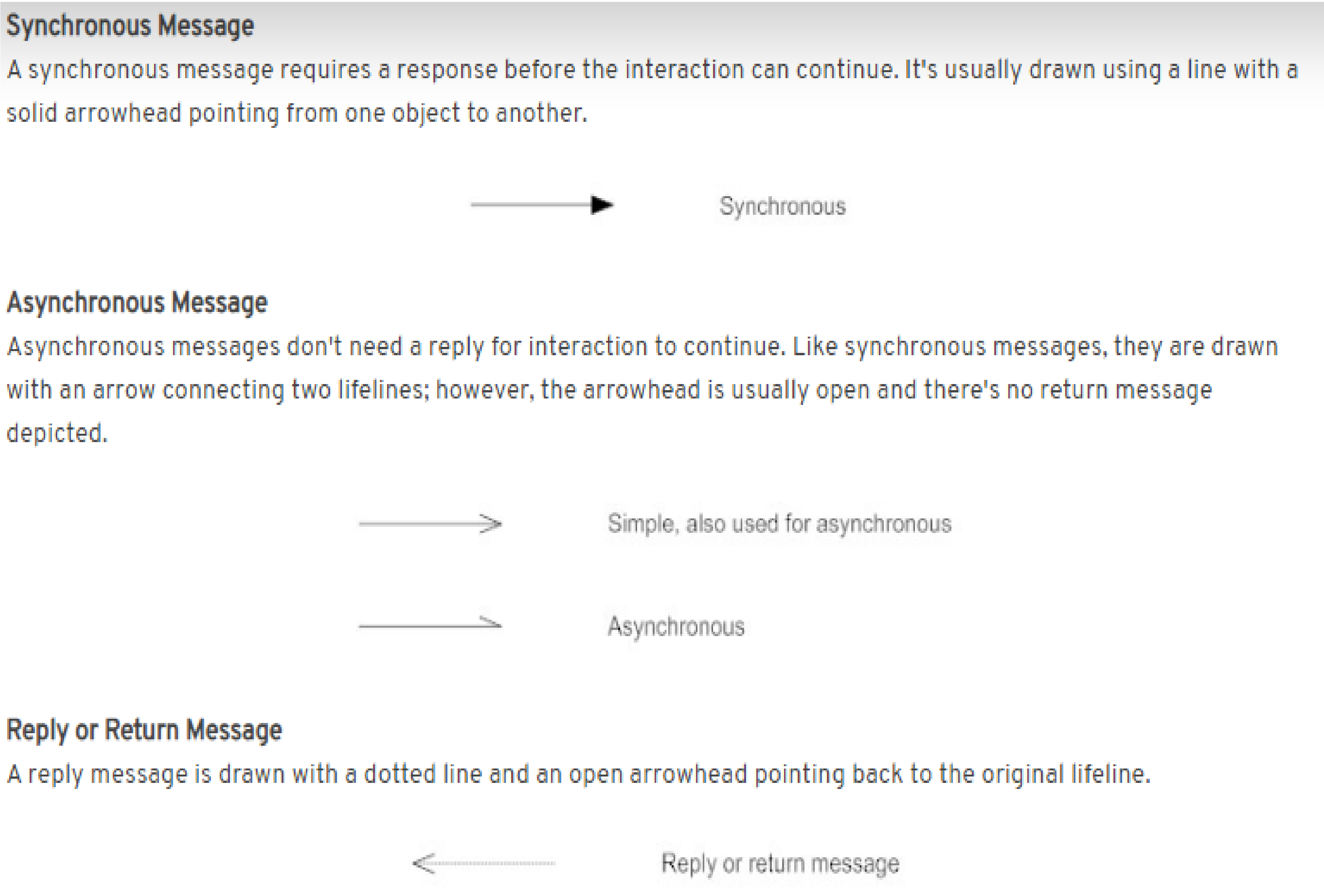
Destroying Objects

Objects can be terminated early using an arrow labeled "<< destroy >>" that points to an X. This object is removed from memory. When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

Loops

A repetition or loop within a sequence diagram is depicted as a rectangle. Place the condition for exiting the loop at the bottom left corner in square brackets [ ].

Types of messages in Sequence diagrams



Synchronous Message

A synchronous message requires a response before the interaction can continue. It's usually drawn using a line with a solid arrowhead pointing from one object to another.

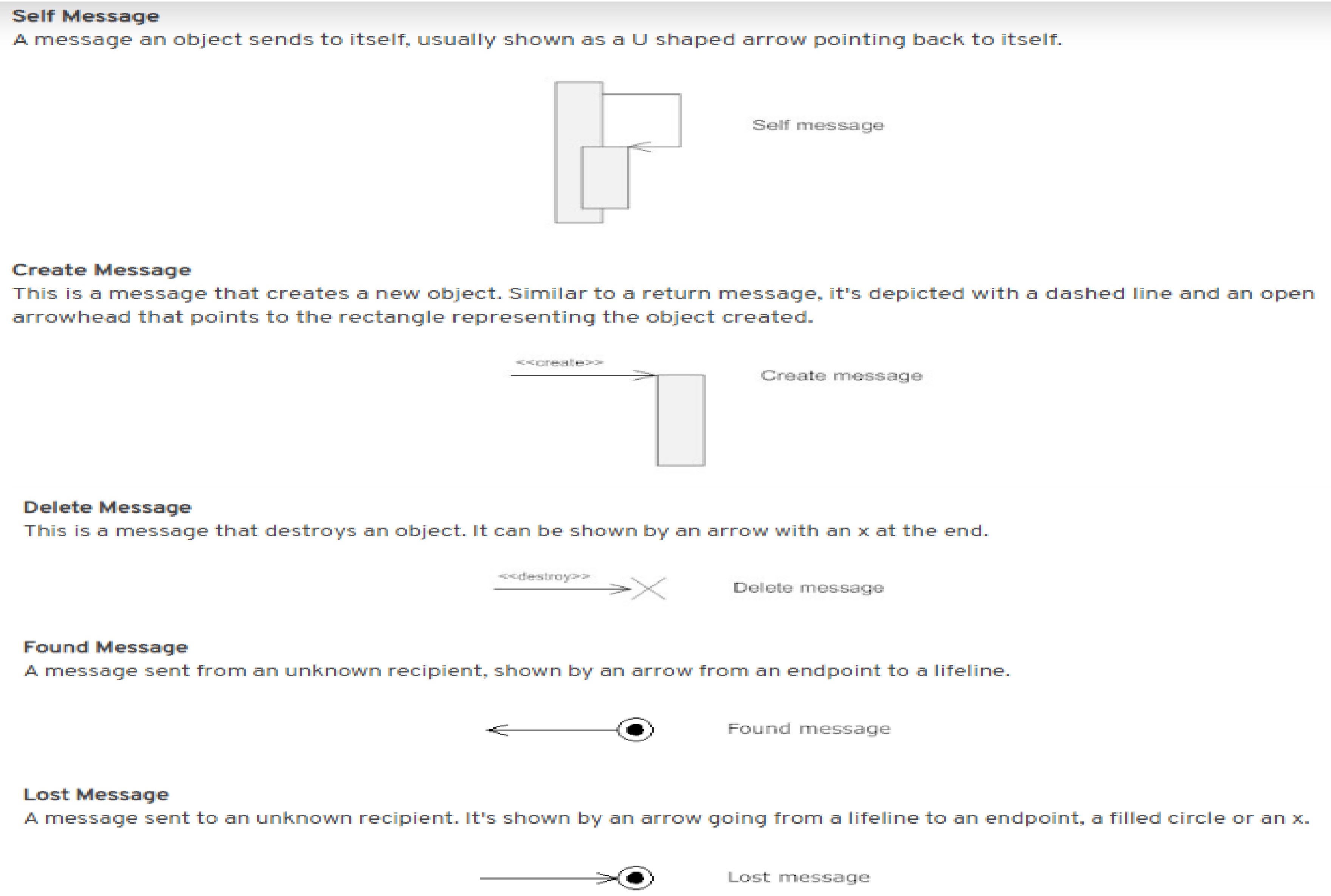
Asynchronous Message

Asynchronous messages don't need a reply for interaction to continue. Like synchronous messages, they are drawn with an arrow connecting two lifelines; however, the arrowhead is usually open and there's no return message depicted.

Reply or Return Message

A reply message is drawn with a dotted line and an open arrowhead pointing back to the original lifeline.

Types of messages in Sequence diagrams (contd..)



Self Message

A message an object sends to itself, usually shown as a U shaped arrow pointing back to itself.

Create Message

This is a message that creates a new object. Similar to a return message, it's depicted with a dashed line and an open arrowhead that points to the rectangle representing the object created.

Delete Message

This is a message that destroys an object. It can be shown by an arrow with an x at the end.

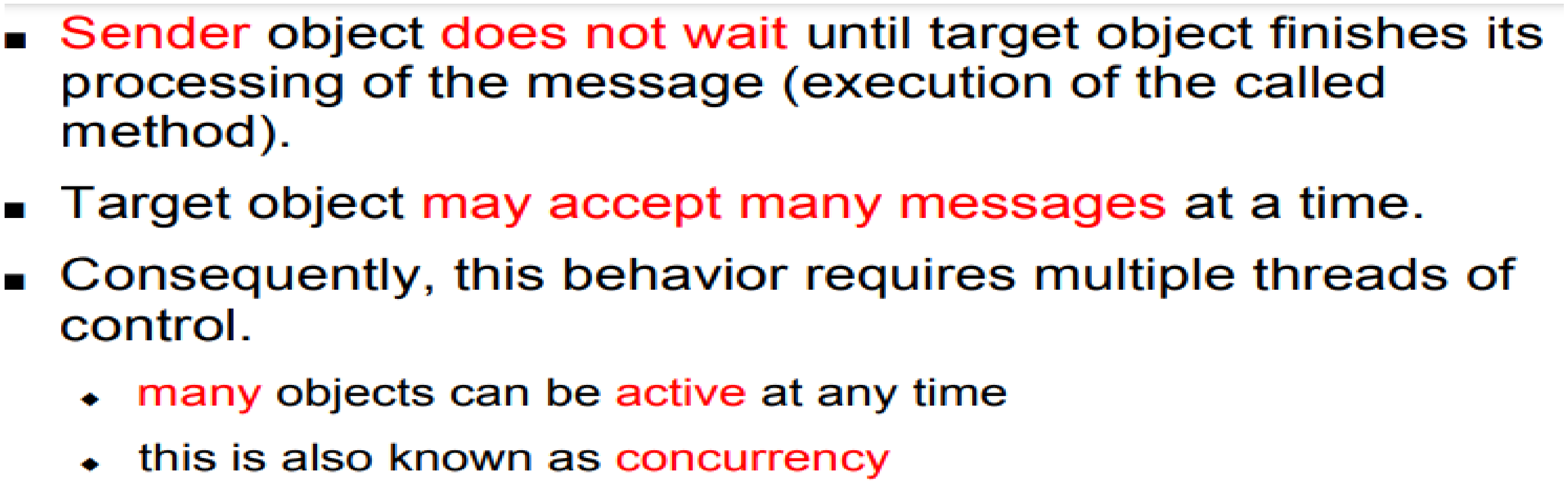
Found Message

A message sent from an unknown recipient, shown by an arrow from an endpoint to a lifeline.

Lost Message

A message sent to an unknown recipient. It's shown by an arrow going from a lifeline to an endpoint, a filled circle or an x.

Sequence diagrams(Asychronous messages)



- Sender object does not wait until target object finishes its processing of the message (execution of the called method).
- Target object may accept many messages at a time.
- Consequently, this behavior requires multiple threads of control.
  - many objects can be active at any time
  - this is also known as concurrency

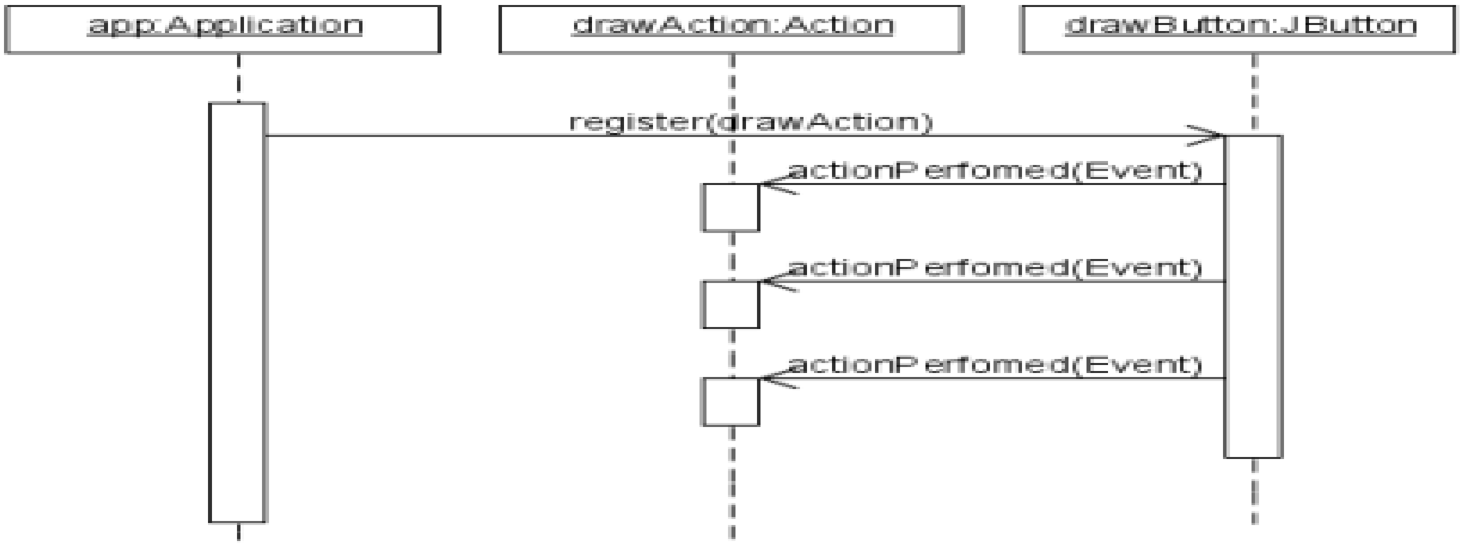
How to deal with concurrency?
One way to deal with asychronous messages is to queue them so only one of them is processed at a times.

But if one message is more important than others then we can use message priorities accordingly.

Sequence diagrams(Callback mechanism)

- Uses asynchronous messages.
- A subscriber object o1 is interested in an event e that occurs in o2.
- o1 registers interest in e by sending a message (that contains a reference to itself) to o2 and continues its execution.
- When e occurs, o2 will callback asynchronously to o1 (and any other subscribers).

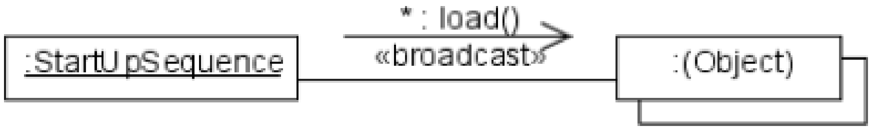
Callback (but not to self) illustrated



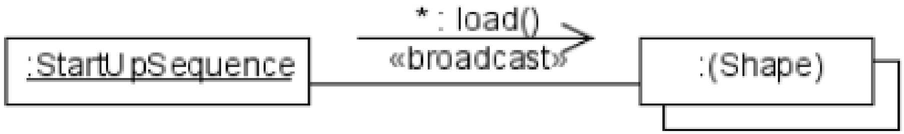
Sequence diagram (contd..)

Broadcast

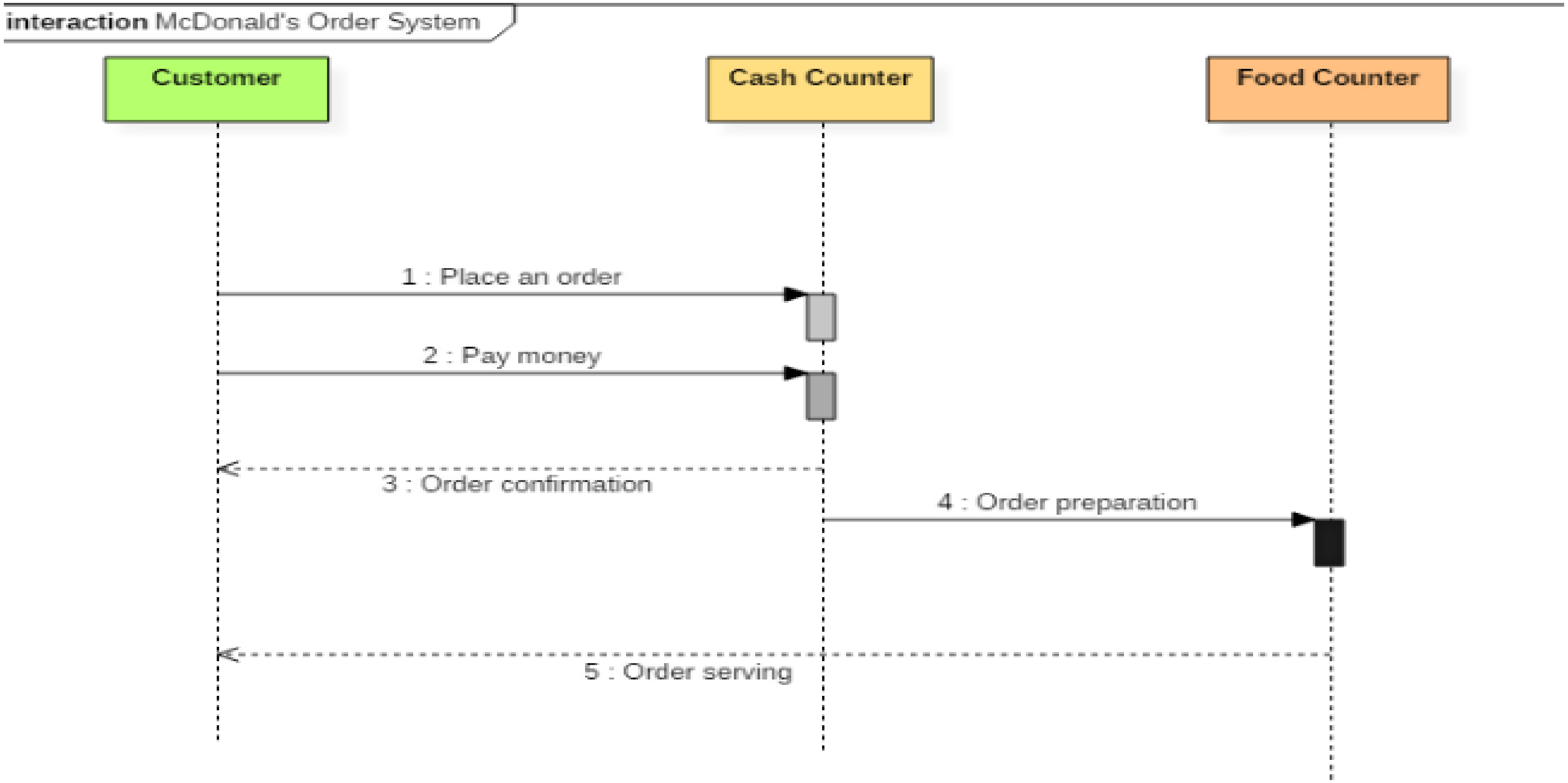
- Similar to iterative messaging, broadcast allows you to send a message to multiple objects.
- However, contrary to iterative messaging, no references are required.
- A broadcast is sent to all the objects in the system.



- If only a specific category of objects is targeted, we call this narrowcast.



Sequence diagram (Example: McDonald's ordering system:)

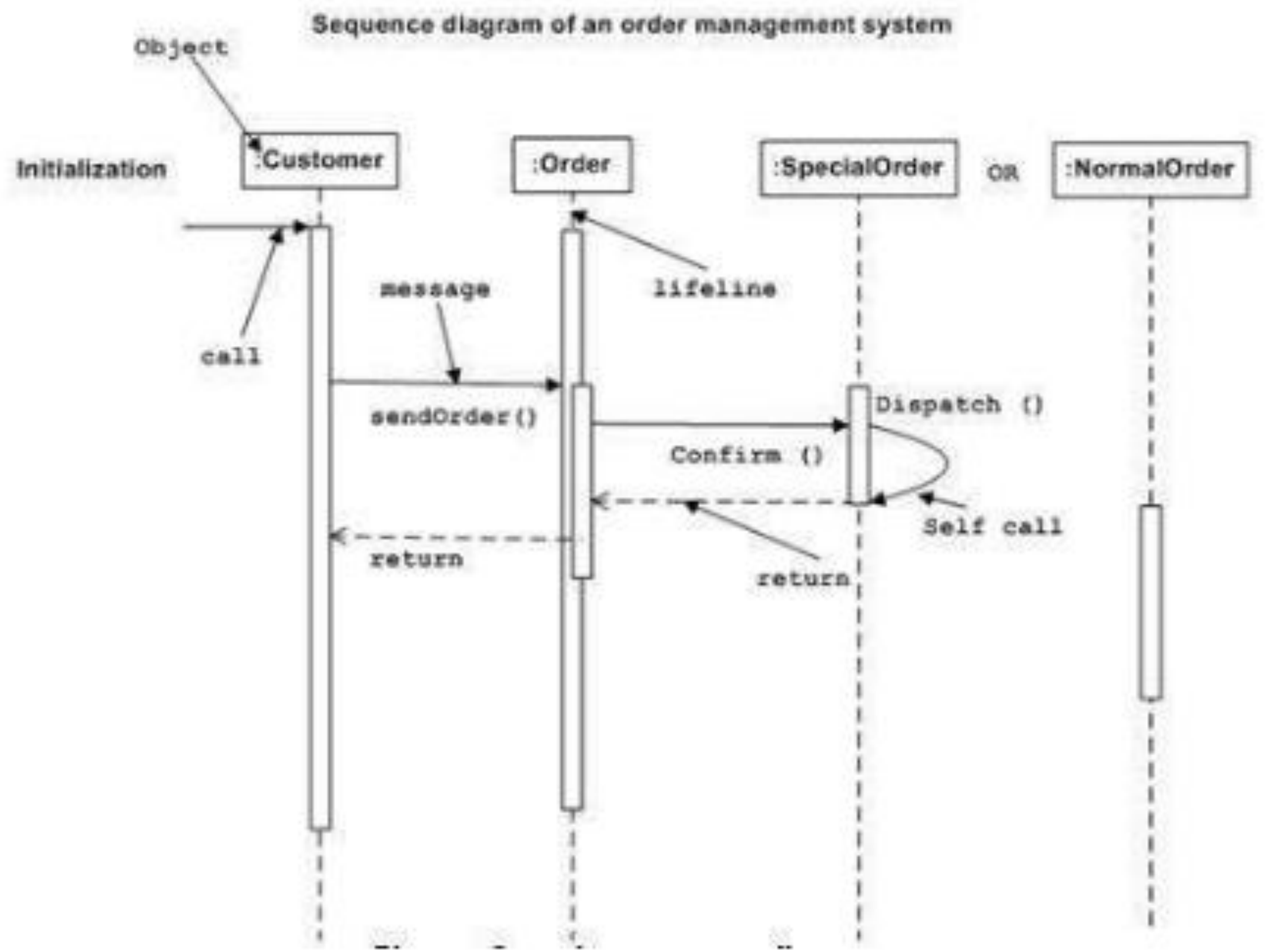


Sequence diagram of Mcdonald's ordering system

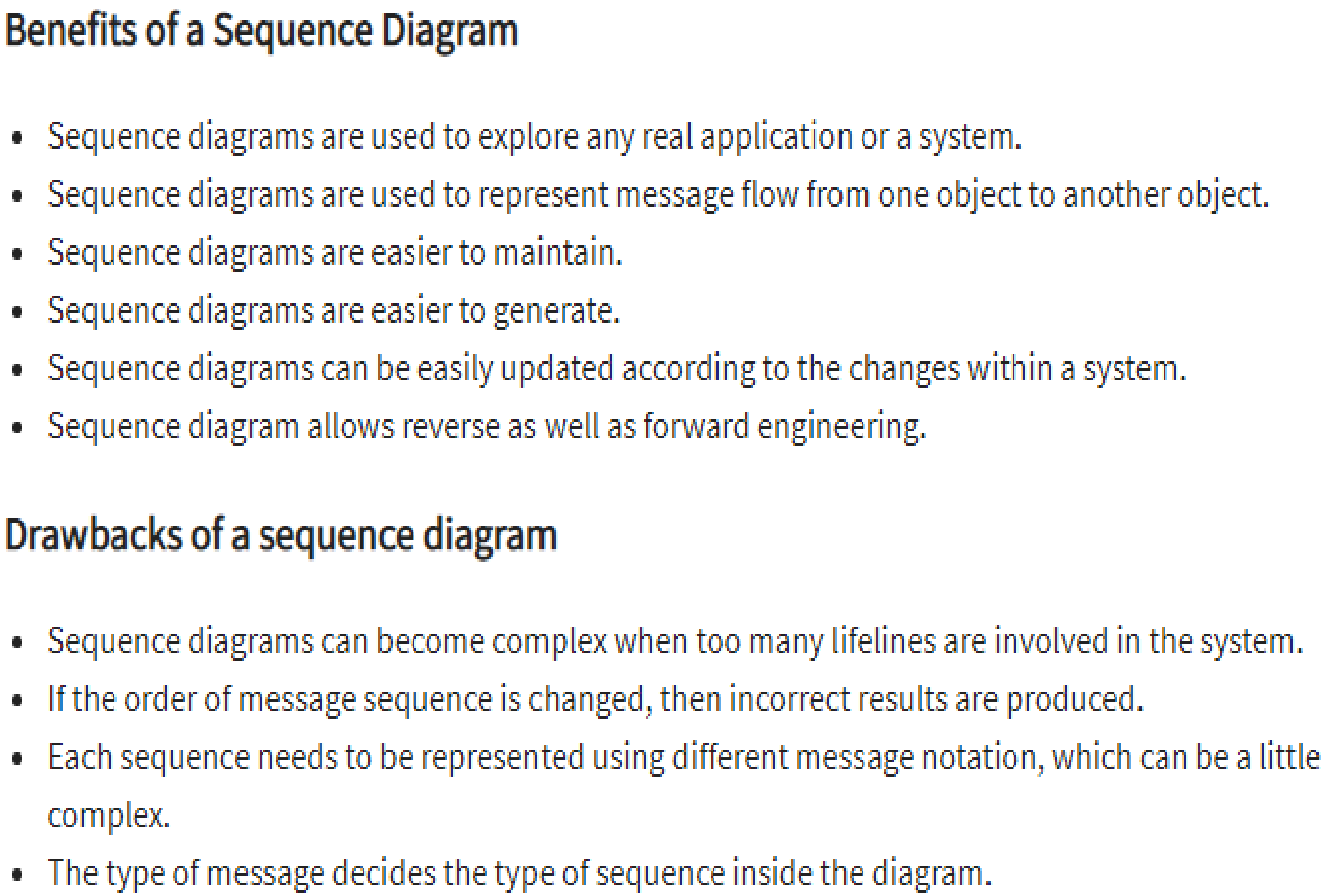
The ordered sequence of events in a given sequence diagram is as follows:

1. Place an order.
2. Pay money to the cash counter.
3. Order Confirmation.
4. Order preparation.
5. Order serving.

Sequence diagram (Example: McDonald's ordering system:)



Sequence diagram (contd..)



Benefits of a Sequence Diagram

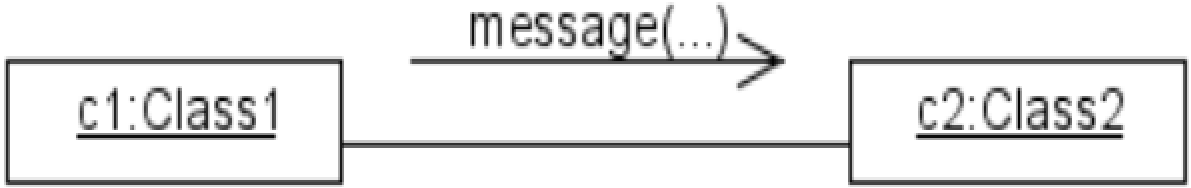
- Sequence diagrams are used to explore any real application or a system.
- Sequence diagrams are used to represent message flow from one object to another object.
- Sequence diagrams are easier to maintain.
- Sequence diagrams are easier to generate.
- Sequence diagrams can be easily updated according to the changes within a system.
- Sequence diagram allows reverse as well as forward engineering.

Drawbacks of a sequence diagram

- Sequence diagrams can become complex when too many lifelines are involved in the system.
- If the order of message sequence is changed, then incorrect results are produced.
- Each sequence needs to be represented using different message notation, which can be a little complex.
- The type of message decides the type of sequence inside the diagram.

Collaboration diagram

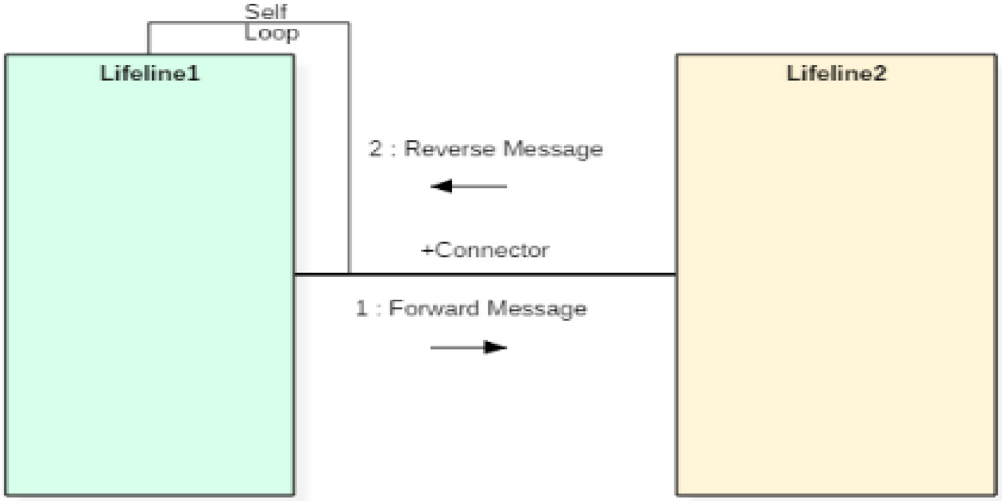
- Collaboration diagrams represent objects in a system an their associations.
- They are composed of three elements:
  - Objects
  - Associations
  - Messages



Collaboration diagram

COLLABORATION DIAGRAM depicts the relationships and interactions among software objects. They are used to understand the object architecture within a system rather than the flow of a message as in a sequence diagram. They are also known as "Communication Diagrams."

Collaboration Diagrams are used to explore the architecture of objects inside the system. The message flow between the objects can be represented using a collaboration diagram.

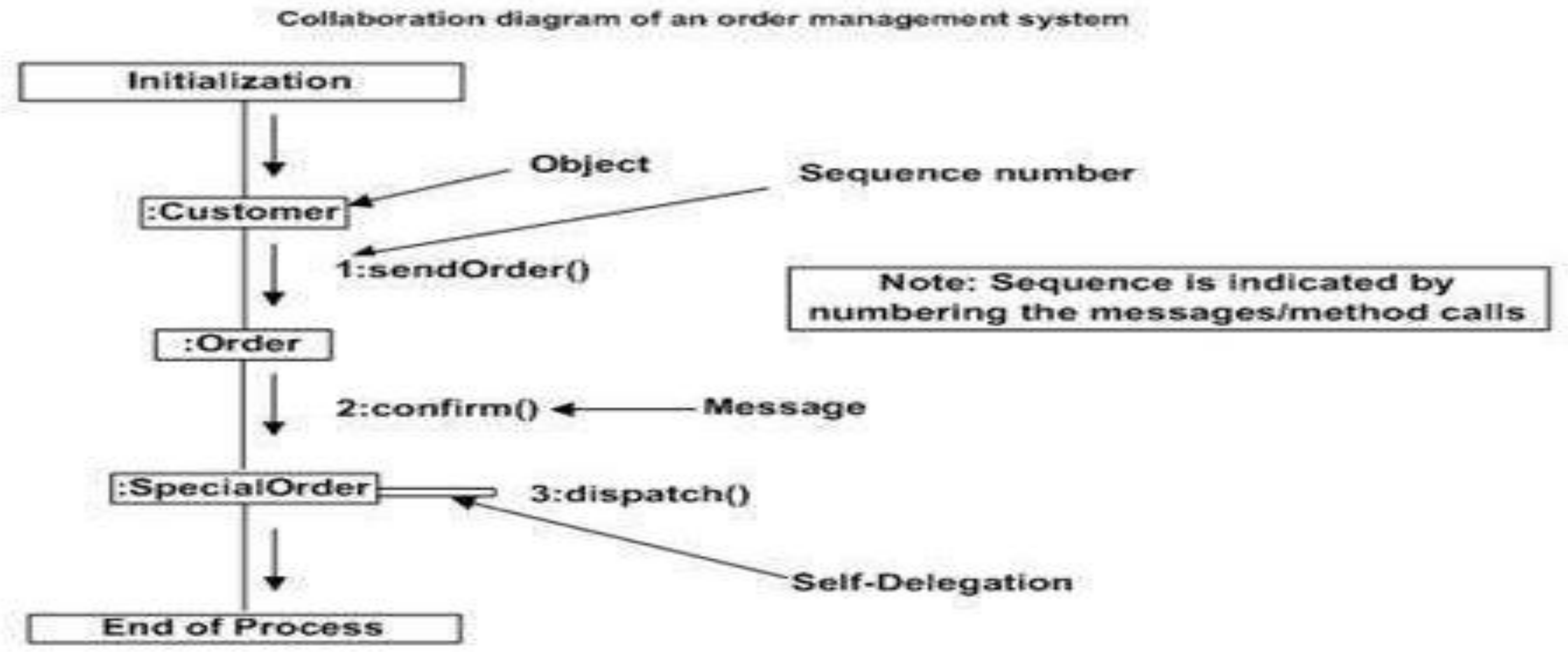


The above collaboration diagram notation contains lifelines along with connectors, self-loops, forward, and reverse messages used in a collaboration diagram.

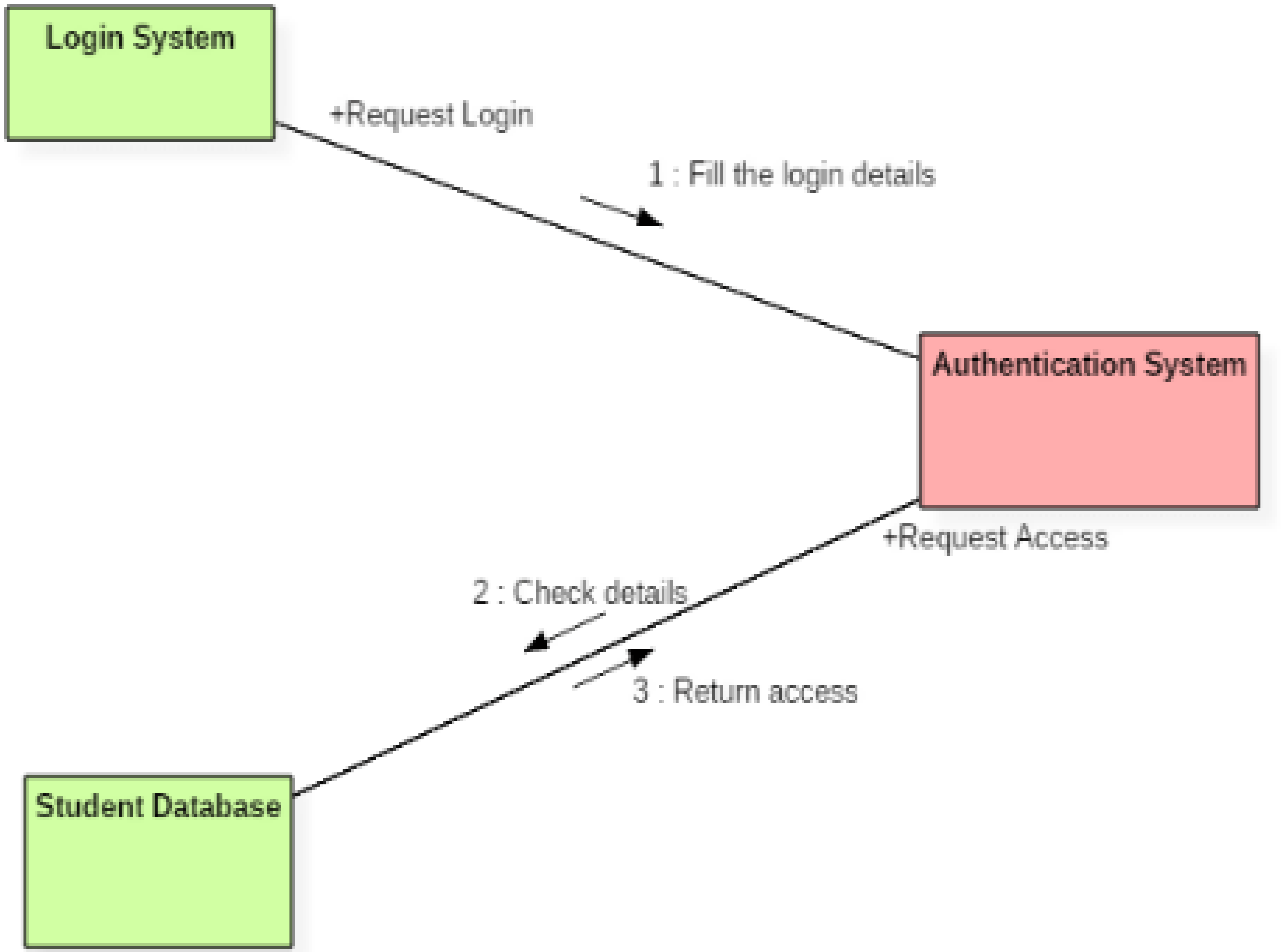
Collaboration diagram (contd..)

In the collaboration diagram, the method call sequence is indicated by some numbering technique. The number indicates how the methods are called one after another. Sequence diagram does not describe the object organization, whereas the collaboration diagram shows the object organization.

To **choose between these two diagrams**, emphasis is placed on the type of requirement. If the time sequence is important, then the sequence diagram is used. If organization is required, then collaboration diagram is used.



Collaboration diagram (contd..)



Collaboration diagram for student management system

Collaboration diagram (contd..)

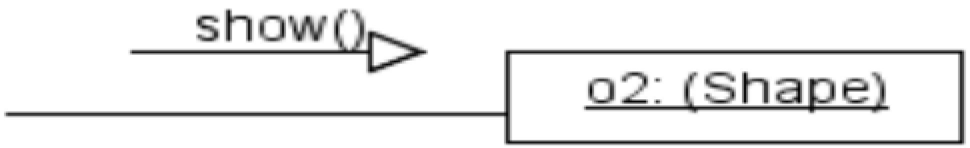
Messages :

Messages in collaboration diagrams are shown as arrows pointing from the Client object to the Supplier object. As a client invoking an operation on a supplier object.
Message icons have one or more messages associated with them. Messages are composed of message text prefixed by a sequence number. This sequence number indicates the time-ordering of the message. For example:

1. Enter Borrower ID
1.1 CalcAmtCanBorrow
1.1.1 <<create>>
1.1.2 CalcBorrowerFines
1.1.3 GetBorrowersCheckedOutMedia
1.1.4 IsMediaOverdue
1.1.5 Amt Can Borrow
1.2 Display Invalid User Msg

Collaboration diagram (Polymorphism)

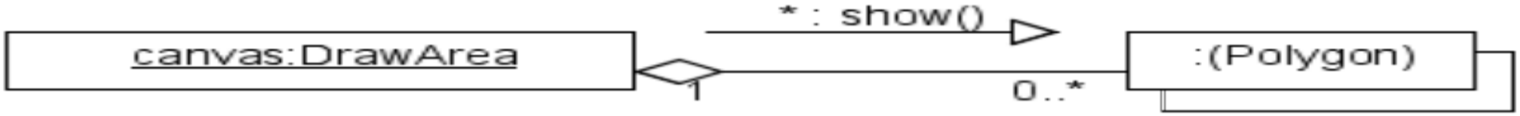
- Polymorphism is a problem in object interaction.
- Suppose we want to send the message show() to a Shape object.
  - That could be an instance of Triangle, Rectangle or Square at run-time.
  - How do we depict this in a collaboration diagram?
- Usually, we are certain that o1 sends a message to o2
- Also, suppose that Triangle, Rectangle and Square are subclasses of Shape.
- Make the target object's class the lowest class in the inheritance hierarchy that is a superclass of all the classes to which the target object may belong to.
- Put the superclass name in parenthesis to show that it will be evaluated at run-time.
- This is a form of substitutability.



Collaboration diagram (Iterated messages)

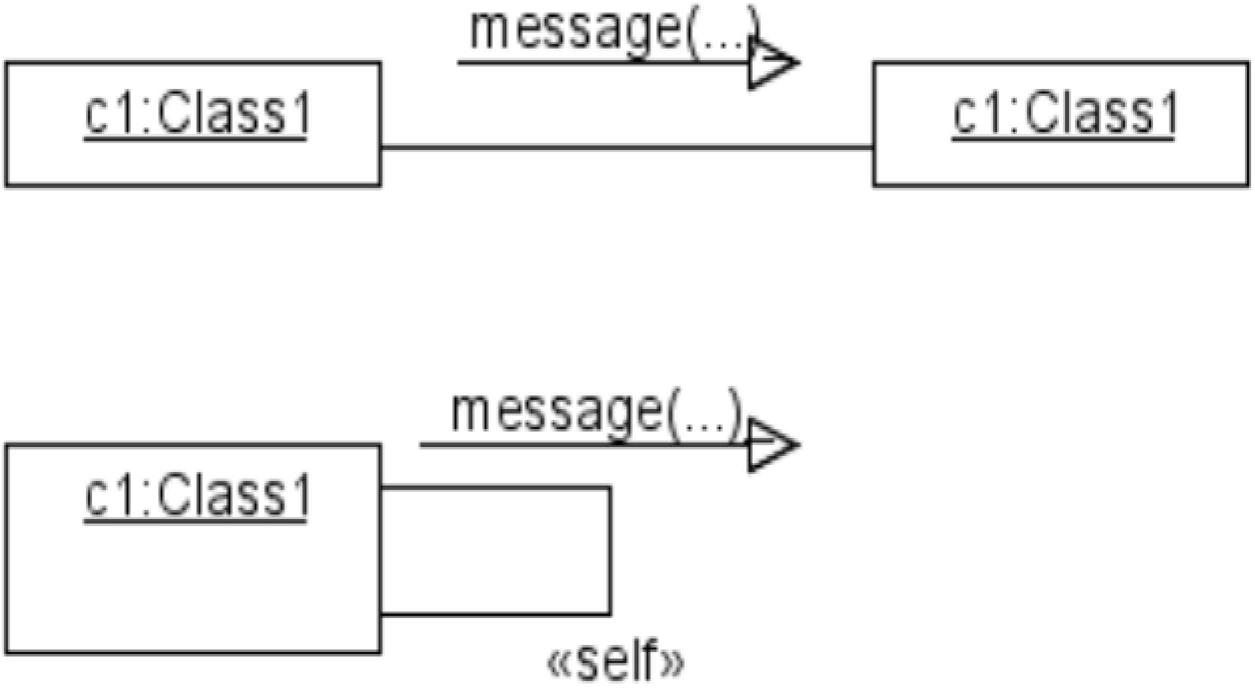
A particular message **iterates** by prefixing a message sequence number with an iteration expression. You can simply use an asterisk (*) to indicate that a message runs more than once, or you can get more specific and show the number of times a message is repeated (for example, 1..5).

- Suppose we have an object DrawArea which has a shapes array of Polygons (Triangles, Rectangles and Squares) that belong to its area.
- We want to repeatedly send the message show() to all the constituent objects (Polygons) of the aggregate object (DrawArea).
- Iterator Pattern (a design pattern) can be used as a traversal method.
- Notice the aggregate connector.
- show() message is called many times (the *)
- DrawArea may have 0 or more Polygons in its array named shapes.
- Target object is unnamed and double boxed to show multiplicity.



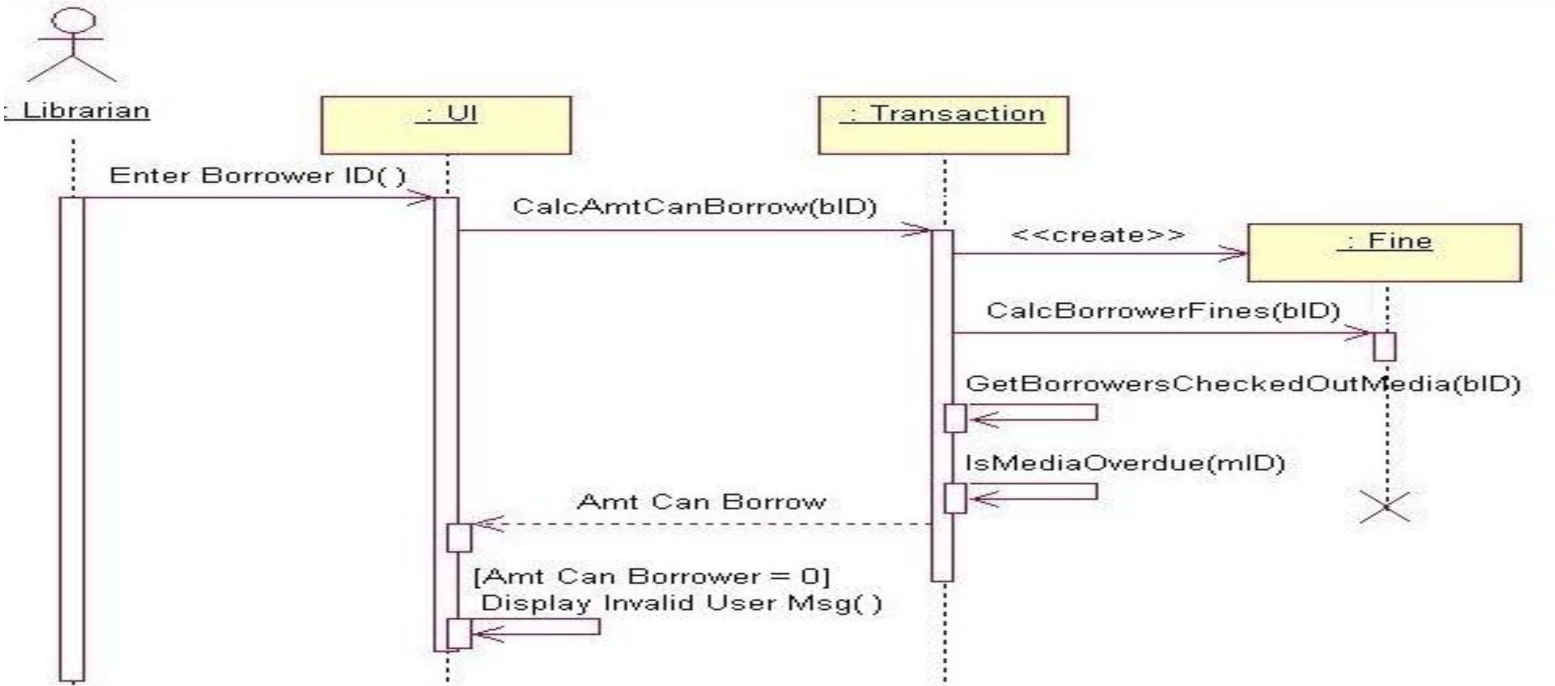
Collaboration diagram (Use of self in messages)

- There are two ways to depict this.

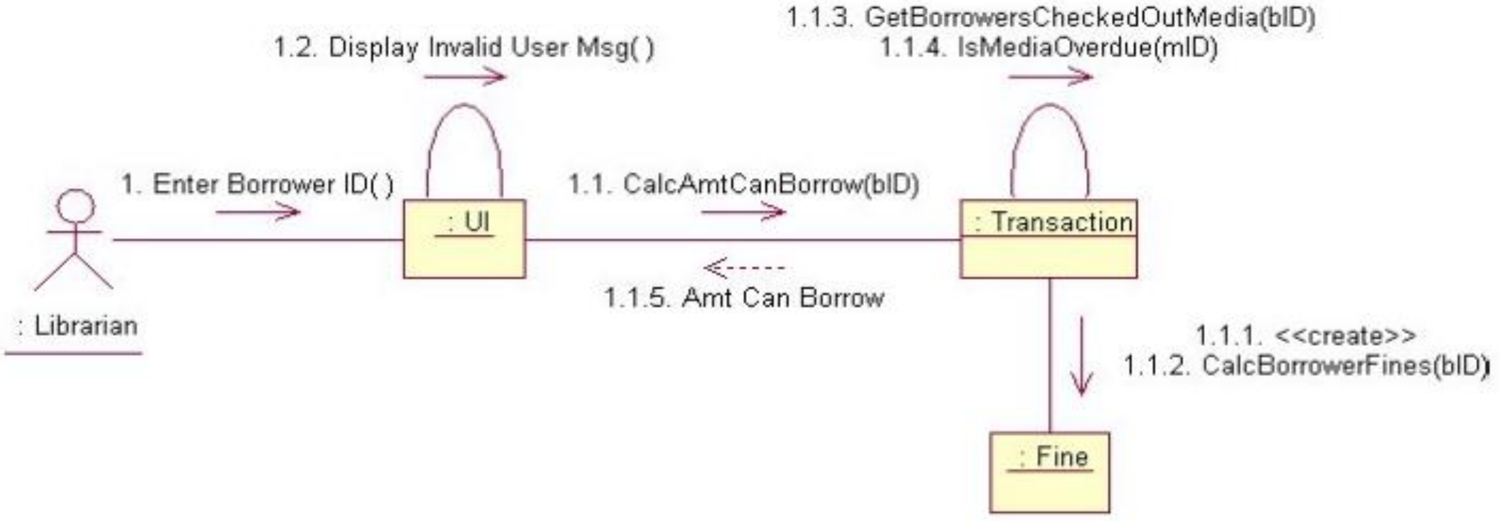


Sequence diagram to Collaboration diagram

Sequence diagram (Figure.1):

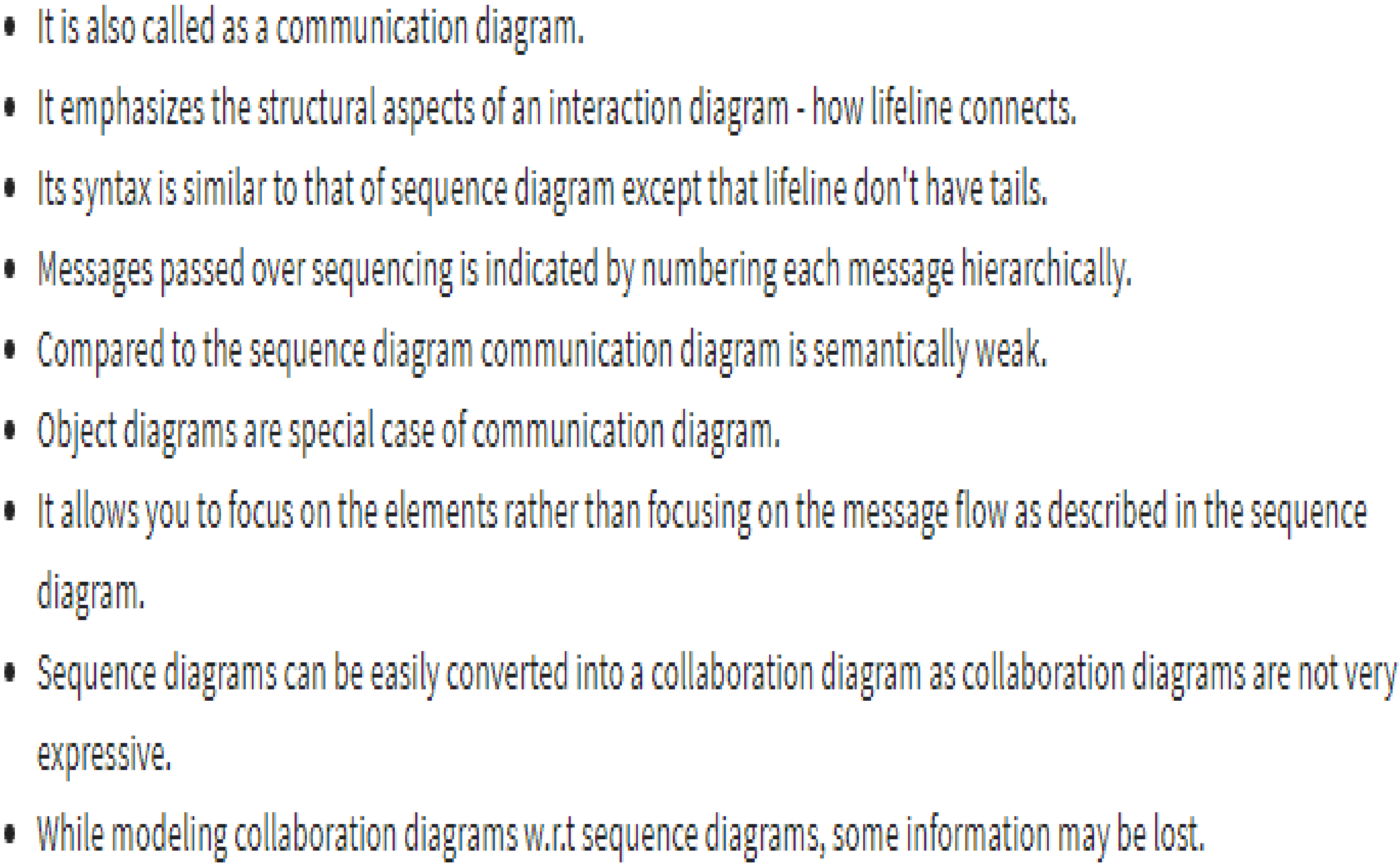


Collaboration diagram (FIGURE.2) :



1.1.3. GetBorrowersCheckedOutMedia(bID)
1.1.4. IsMediaOverdue(mID)
1.2. Display Invalid User Msg( )
1. Enter Borrower ID( )
1.1. CalcAmtCanBorrow(bID)
: UI
: Transaction
1.1.5. Amt Can Borrow
: Librarian
1.1.1. <<create>>
1.1.2. CalcBorrowerFines(bID)
: Fine

Benefits of Collaboration diagram



- It is also called as a communication diagram.
- It emphasizes the structural aspects of an interaction diagram - how lifeline connects.
- Its syntax is similar to that of sequence diagram except that lifeline don't have tails.
- Messages passed over sequencing is indicated by numbering each message hierarchically.
- Compared to the sequence diagram communication diagram is semantically weak.
- Object diagrams are special case of communication diagram.
- It allows you to focus on the elements rather than focusing on the message flow as described in the sequence diagram.
- Sequence diagrams can be easily converted into a collaboration diagram as collaboration diagrams are not very expressive.
- While modeling collaboration diagrams w.r.t sequence diagrams, some information may be lost.

Use Case

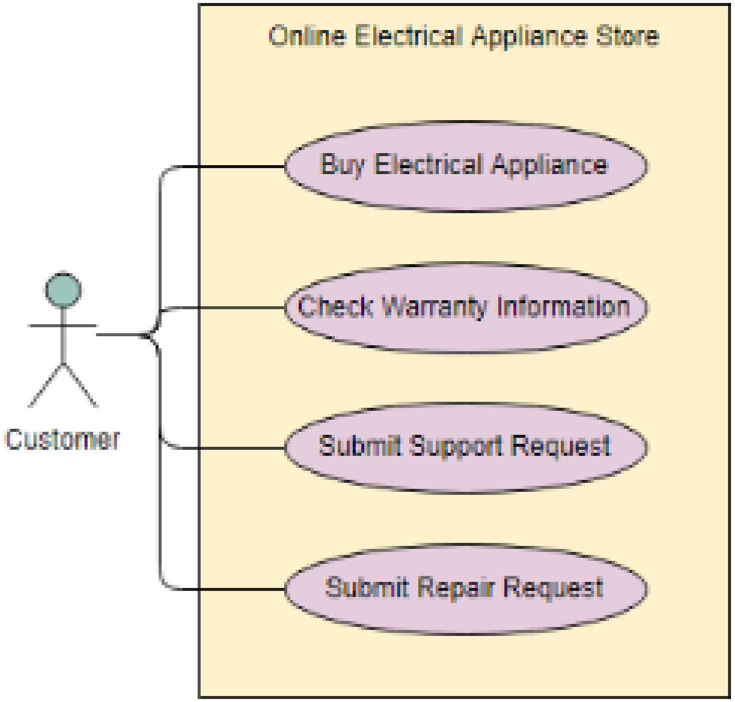
A **use case** describes how a user uses a system to accomplish a particular goal. A use case diagram consists of the system, the related use cases and actors and relates these to each other to visualize:

•what is being described? (**system**)

• who is using the system? (**actors**)

•what do the actors want to achieve? (**use cases**)

Thus, use cases help ensure that the correct system is developed by capturing the requirements from the user's point of view.



Use Case in UML

A use case is a list of actions or event steps typically defining the interactions between a role of an actor and a system to achieve a goal. A use case is a useful technique for identifying, clarifying, and organizing system requirements. A use case is made up of a set of possible sequences of interactions between systems and users that defines the features to be implemented and the resolution of any errors that may be encountered.

A use-case diagram can help provide a higher-level view of the system, providing the simplified and graphical representation of what the system must actually do.

Characteristics:

- Organizes functional requirements
- Models the goals of system/actor (user) interactions
- Describes one main flow of events (main scenarios) and possibly other exceptional flows (alternatives), also called paths or user scenarios

Use Case Diagram (**Notations)**

Use case diagrams consist of 4 objects.

1. Actor
2. Use case
3. System
4. Package

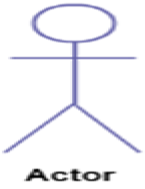
In details , these are:

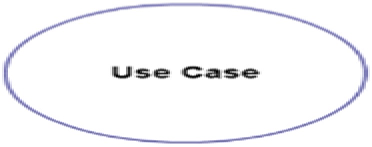
1. Actor: Actor in a use case diagram is **any entity that performs a role** in one given system.

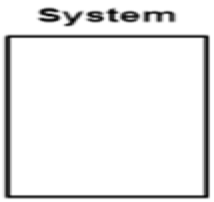
2. Use case : A use case **represents a function or an action within the system**. It’s drawn as an oval and named with the function.

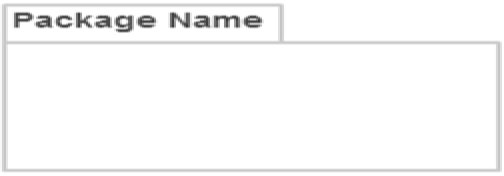
3. **System:** The system is used to **define the scope of the use case** and drawn as rectangle. This an optional element but useful when you’re visualizing large systems.

4. Package: The package is another optional element that is extremely useful in complex diagrams. Similar to class diagram, packages are **used to group together use cases**.





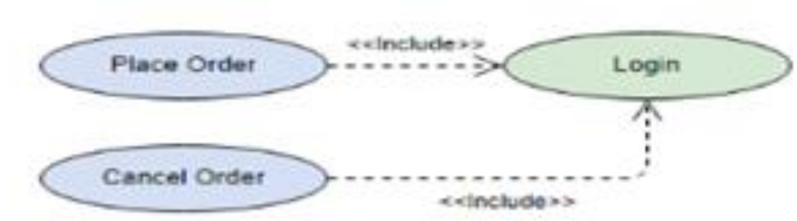


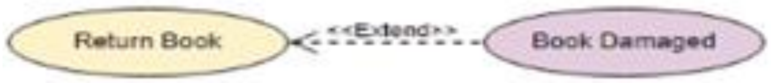


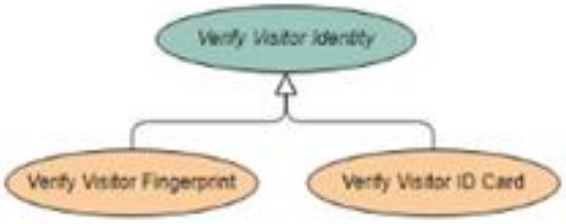
Use Case Diagram (**Notations) contd..**

5. Relationships: There are five types of relationships in a use case diagram. They are

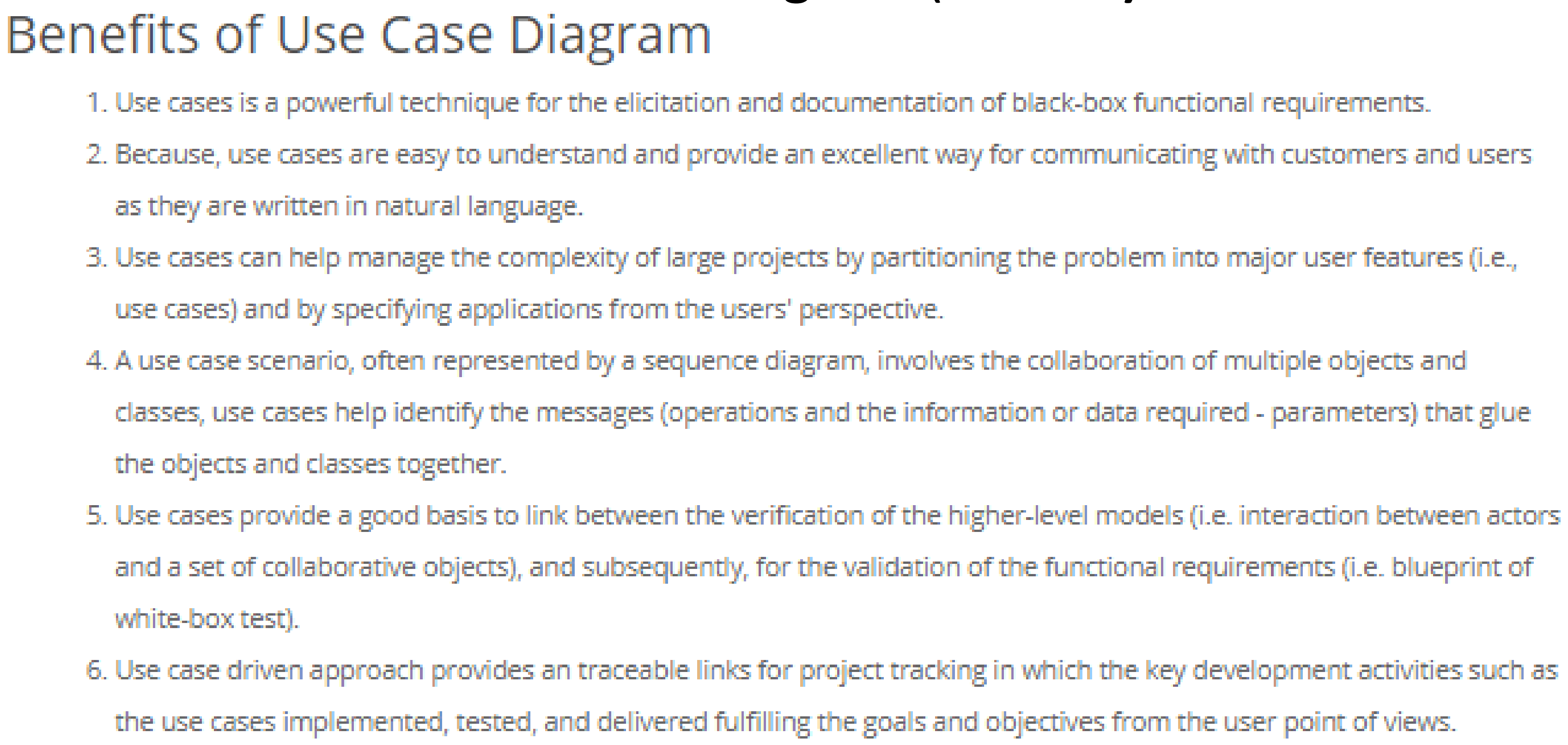
1) Association between an actor and a use case
2) Generalization of an actor
3) Extend relationship between two use cases (<<extend>>)
4) Include relationship between two use cases (<<include>>
5) Generalization of a use case





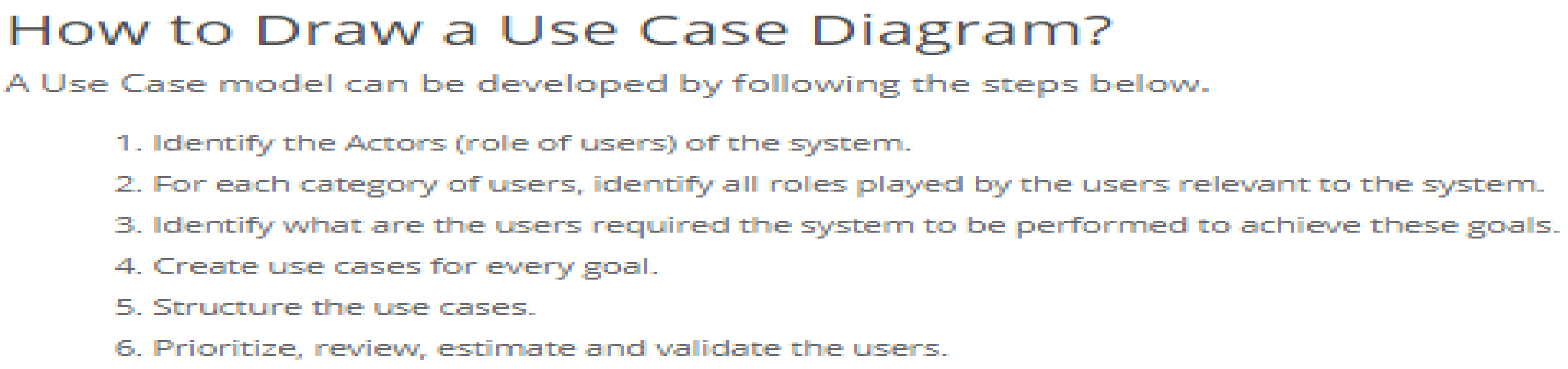


Use Case Diagram (**contd..)**



Benefits of Use Case Diagram

1. Use cases is a powerful technique for the elicitation and documentation of black-box functional requirements.
2. Because, use cases are easy to understand and provide an excellent way for communicating with customers and users as they are written in natural language.
3. Use cases can help manage the complexity of large projects by partitioning the problem into major user features (i.e., use cases) and by specifying applications from the users' perspective.
4. A use case scenario, often represented by a sequence diagram, involves the collaboration of multiple objects and classes, use cases help identify the messages (operations and the information or data required - parameters) that glue the objects and classes together.
5. Use cases provide a good basis to link between the verification of the higher-level models (i.e. interaction between actors and a set of collaborative objects), and subsequently, for the validation of the functional requirements (i.e. blueprint of white-box test).
6. Use case driven approach provides an traceable links for project tracking in which the key development activities such as the use cases implemented, tested, and delivered fulfilling the goals and objectives from the user point of views.



How to Draw a Use Case Diagram?

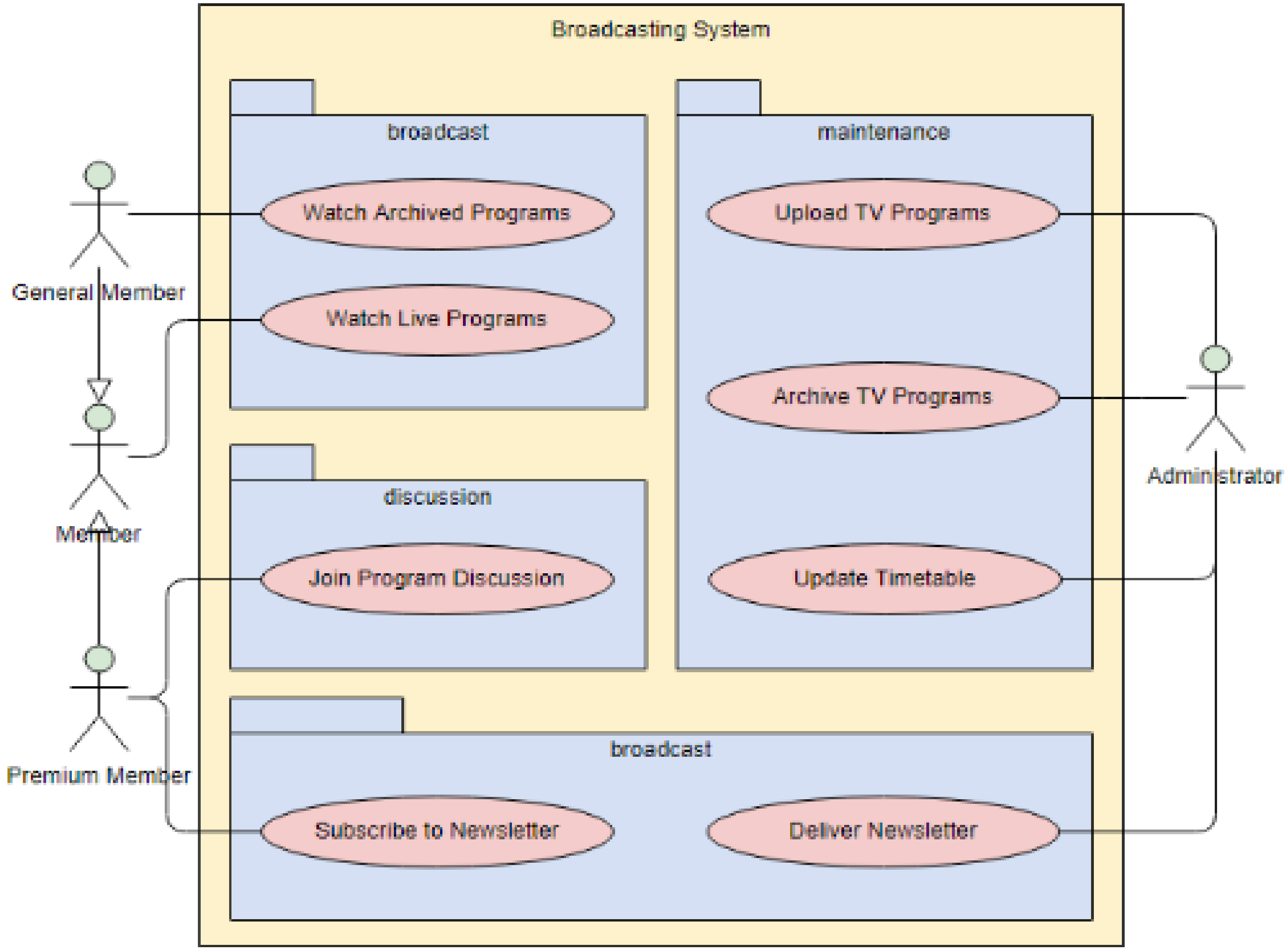
A Use Case model can be developed by following the steps below.

1. Identify the Actors (role of users) of the system.
2. For each category of users, identify all roles played by the users relevant to the system.
3. Identify what are the users required the system to be performed to achieve these goals.
4. Create use cases for every goal.
5. Structure the use cases.
6. Prioritize, review, estimate and validate the users.

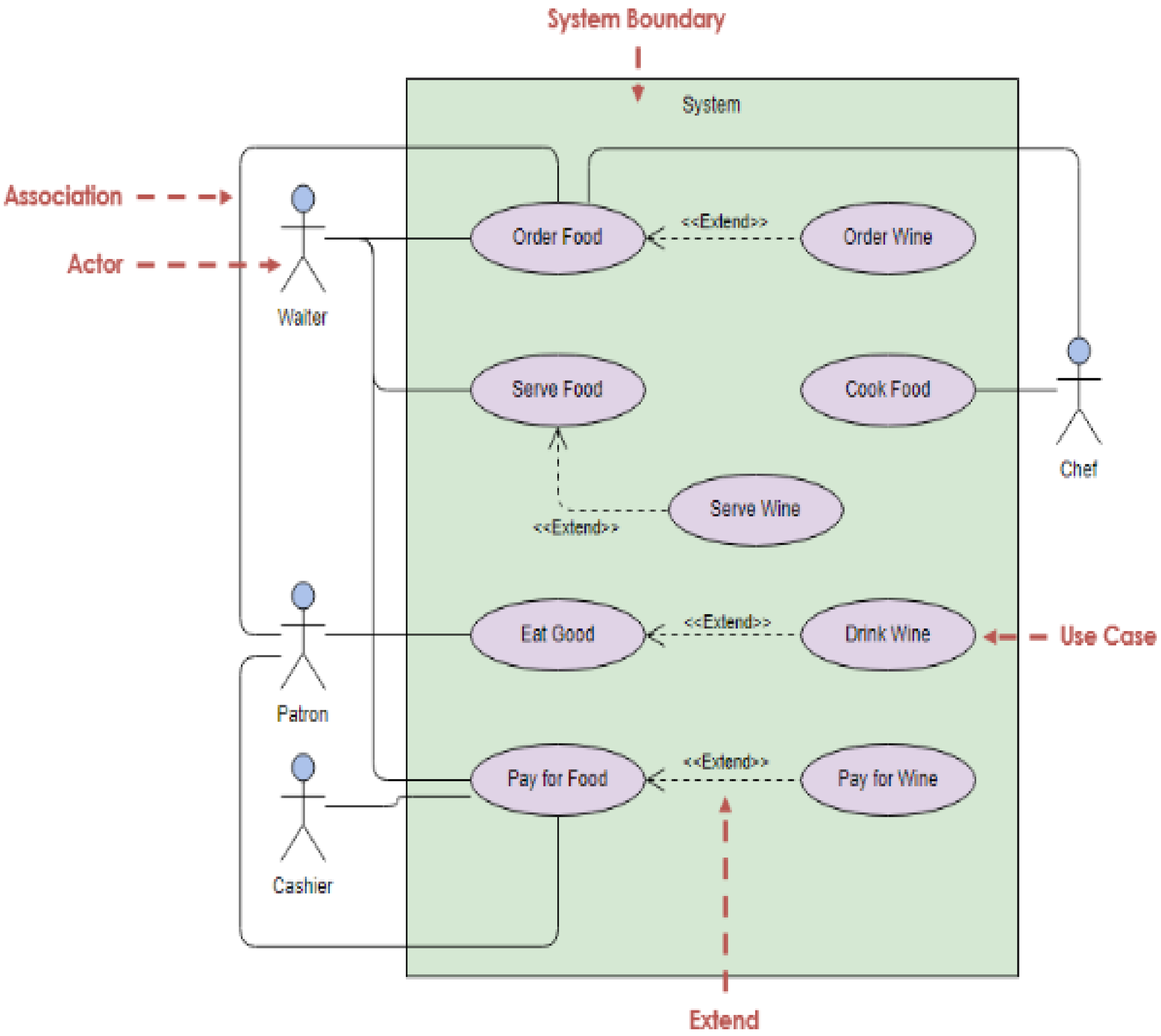
Use Case Diagram (**contd..)**

You can also:

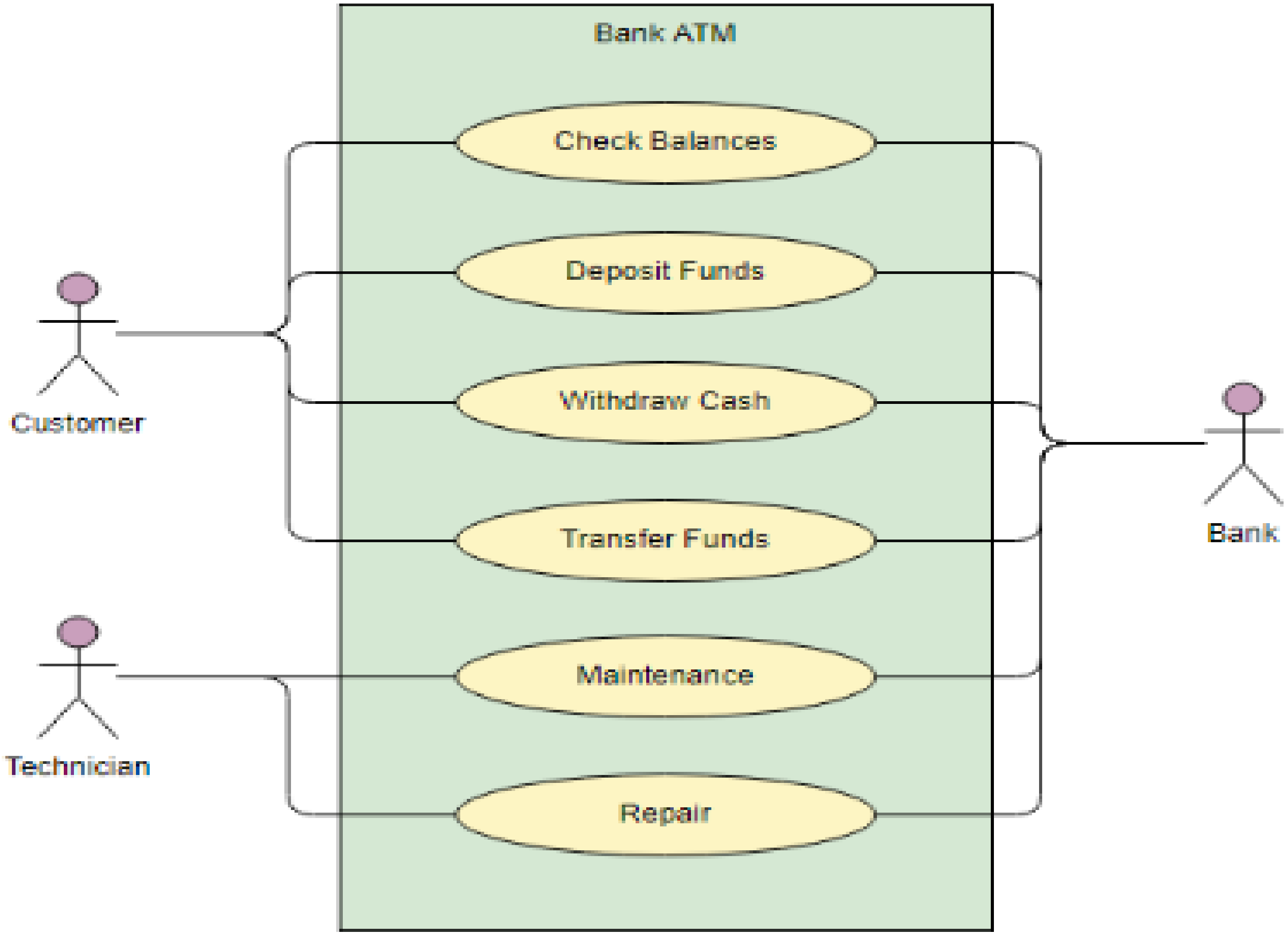
1. Draw packages for logical categorization of use cases into related subsystems.



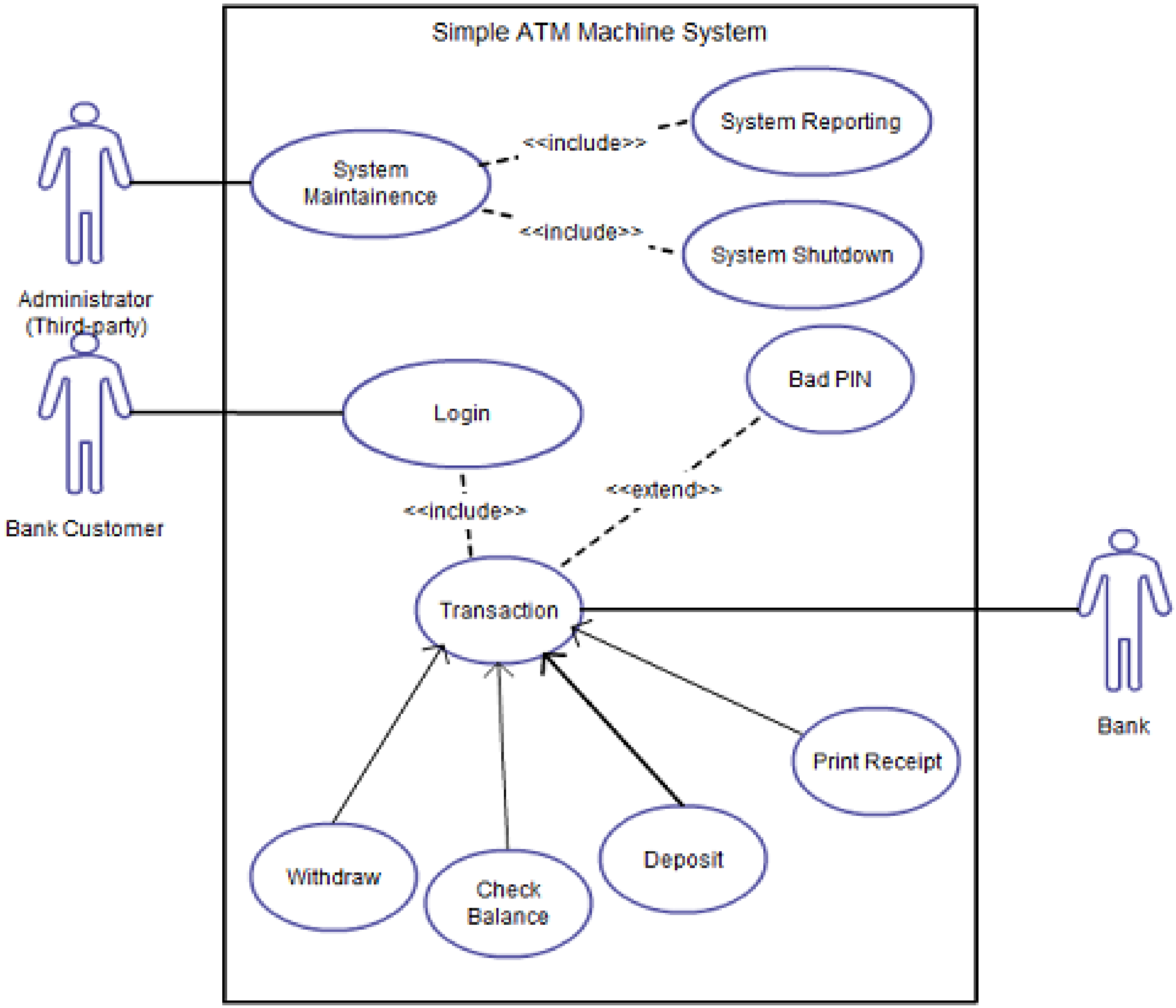
Use Case Diagram : Example (Restaurant business system)



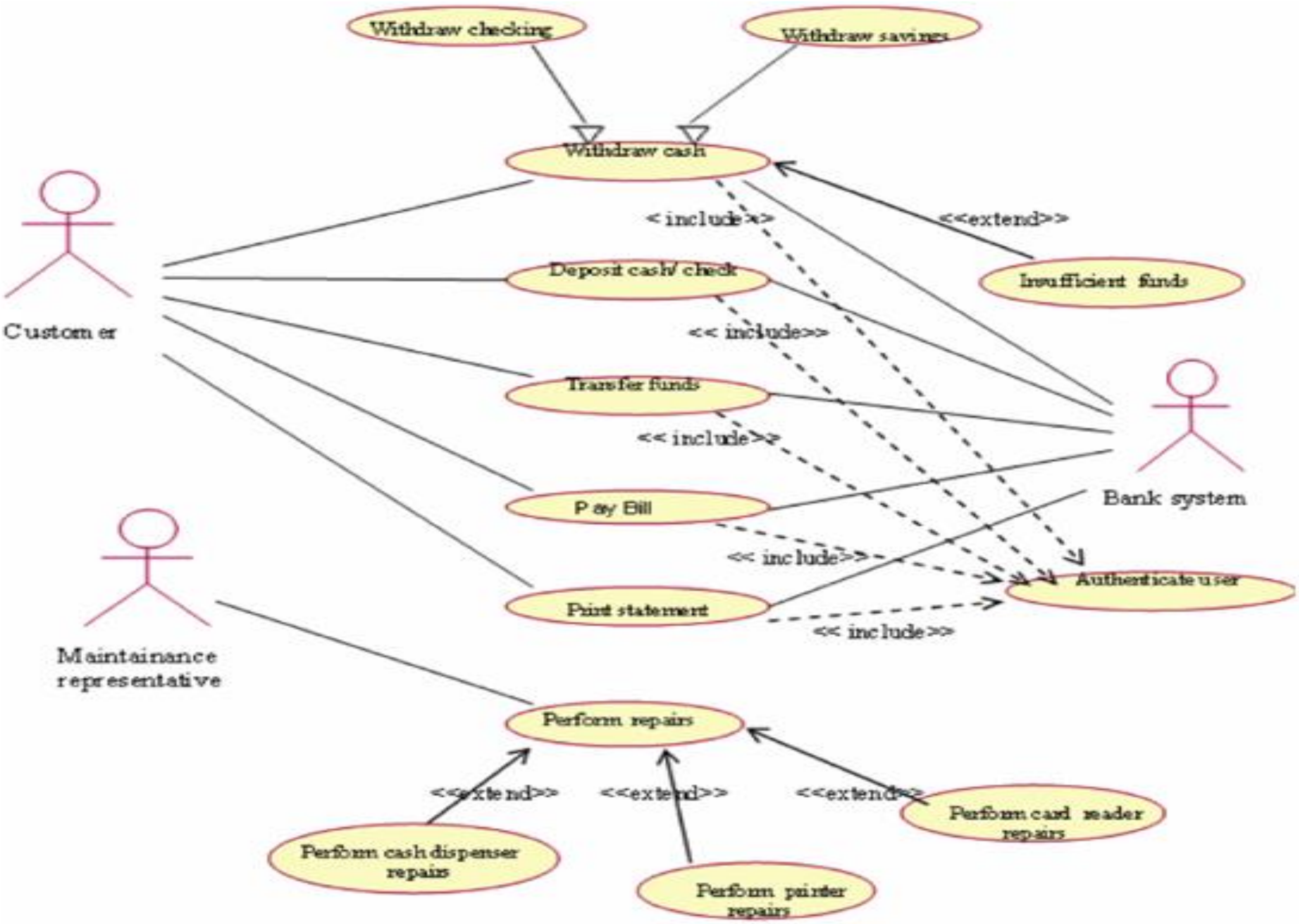
ATM Use Case diagram (diagram:1)



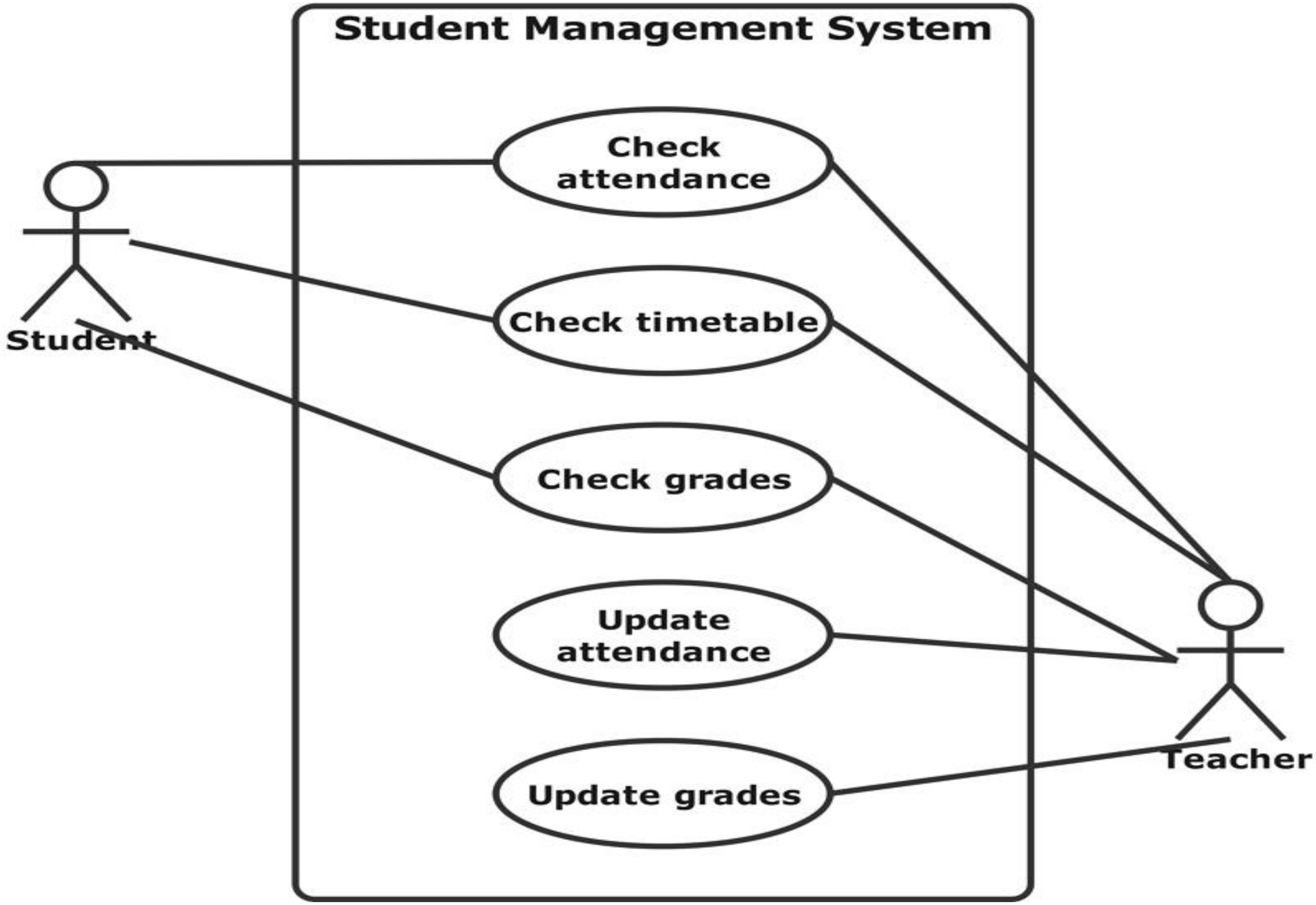
ATM Use Case diagram (diagram:2)



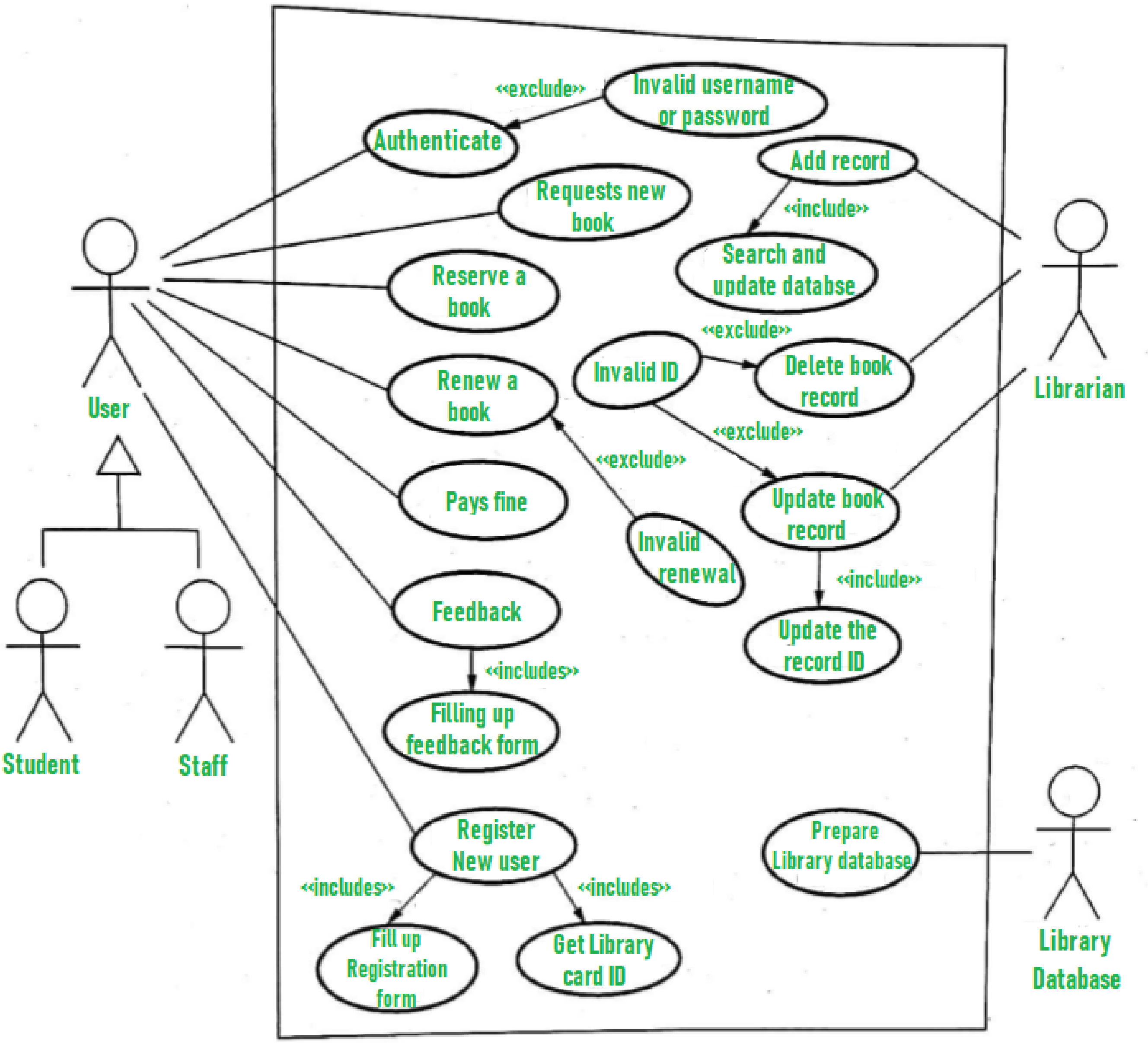
ATM Use Case diagram (diagram:3)



Student Management System Use Case diagram



Library Management System Use Case diagram

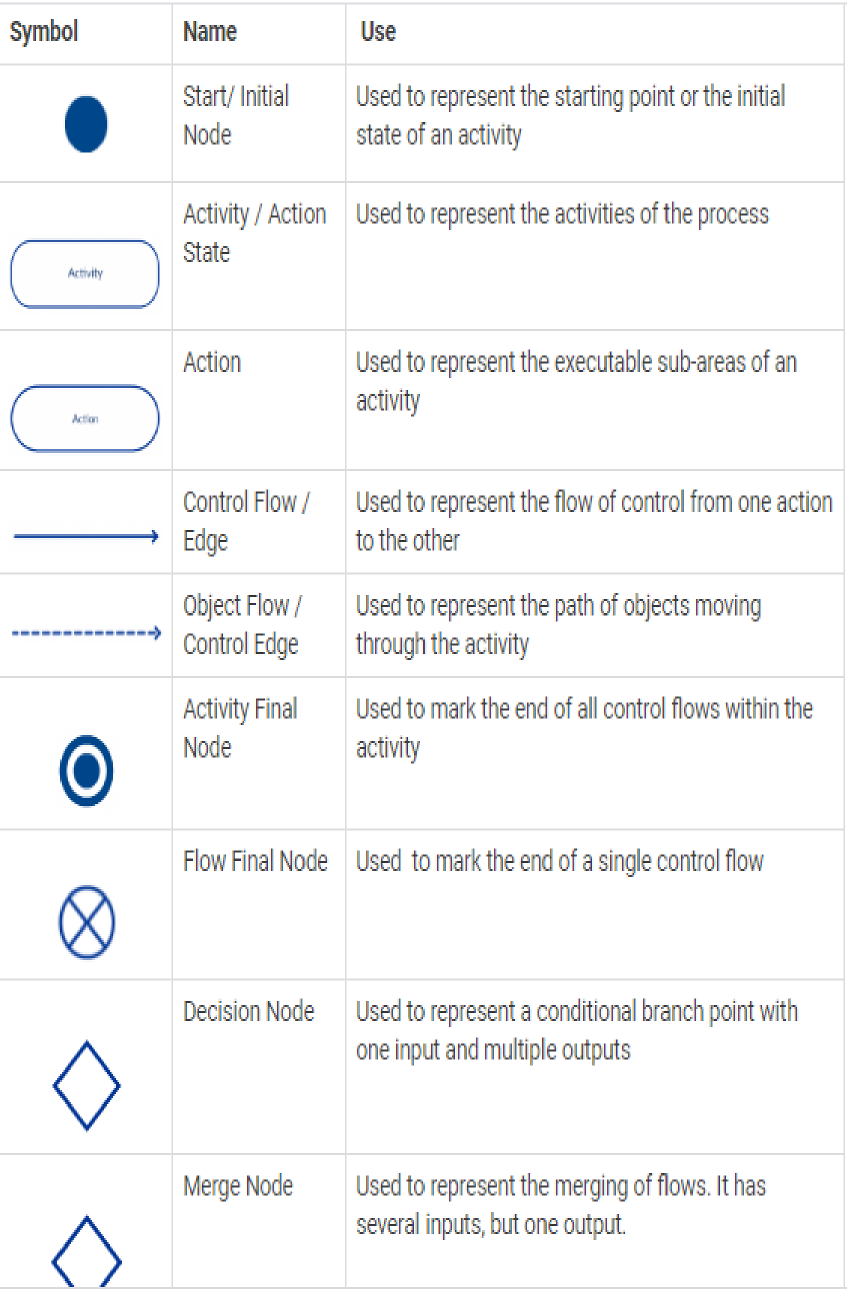


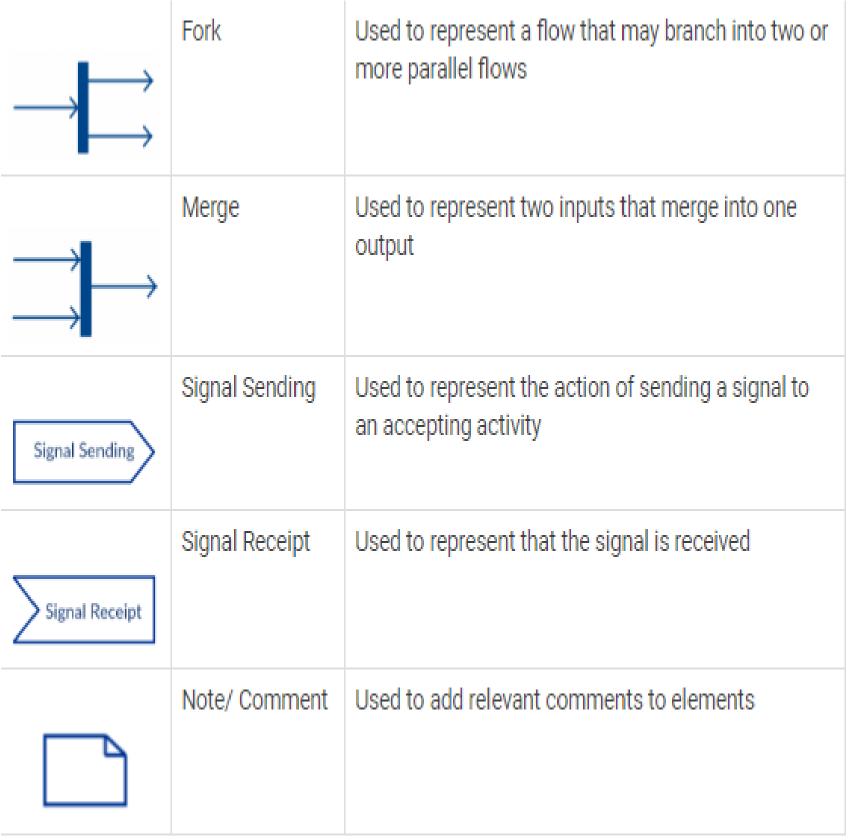
Activity diagram

A UML activity diagram helps to visualize a certain use case at a more detailed level. It is a behavioral diagram that illustrates the flow of activities through a system.

UML activity diagrams can also be used to depict a flow of events in a business process. They can be used to examine business processes in order to identify its flow and requirements.

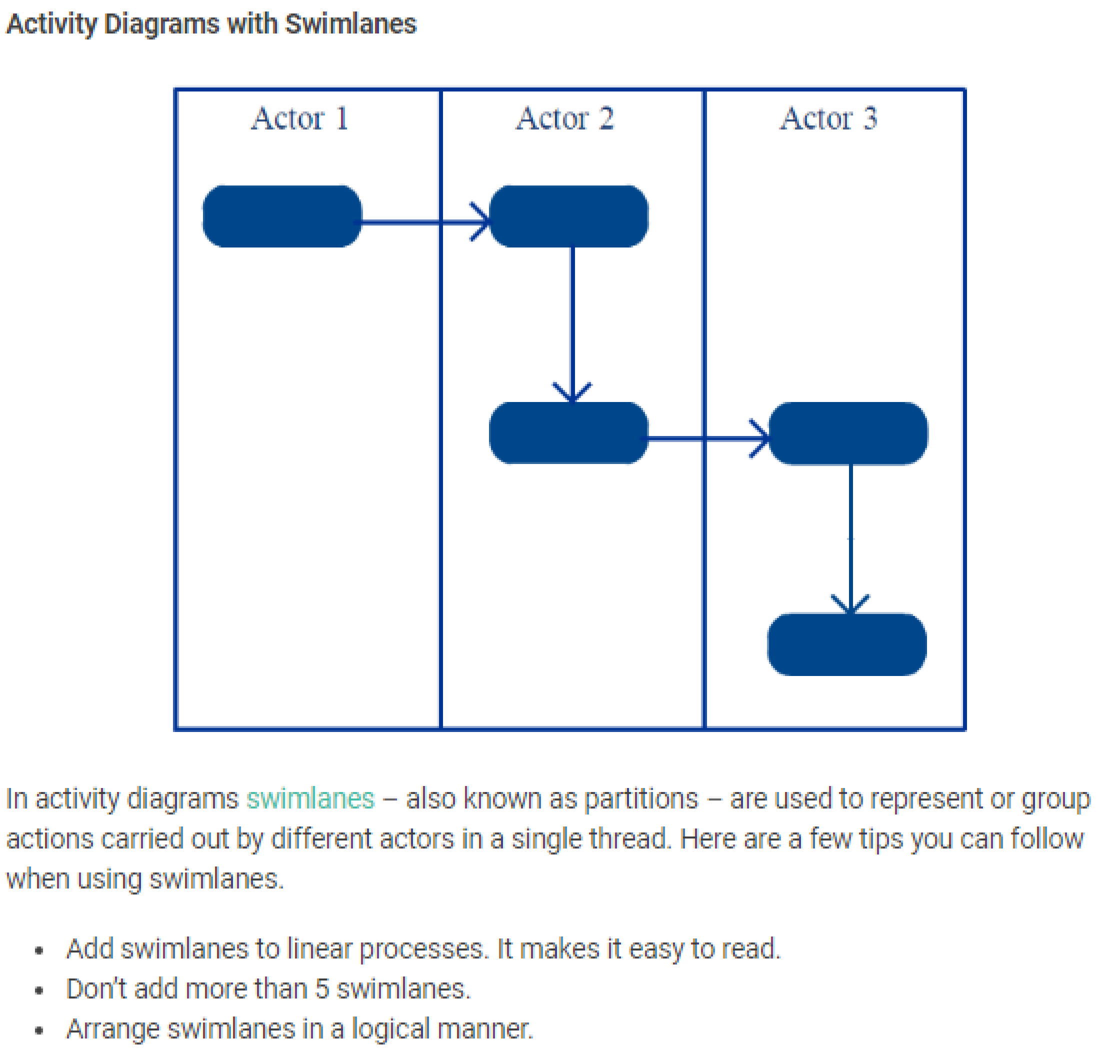
Activity diagram (Symbols)





Symbol	Name	Use
●	Start/ Initial Node	Used to represent the starting point or the initial state of an activity
Activity	Activity / Action State	Used to represent the activities of the process
Action	Action	Used to represent the executable sub-areas of an activity
⟶	Control Flow / Edge	Used to represent the flow of control from one action to the other
- - - - →	Object Flow / Control Edge	Used to represent the path of objects moving through the activity
◉	Activity Final Node	Used to mark the end of all control flows within the activity
⊗	Flow Final Node	Used to mark the end of a single control flow
◇	Decision Node	Used to represent a conditional branch point with one input and multiple outputs
◇	Merge Node	Used to represent the merging of flows. It has several inputs, but one output.
	Fork	Used to represent a flow that may branch into two or more parallel flows
	Merge	Used to represent two inputs that merge into one output
Signal Sending	Signal Sending	Used to represent the action of sending a signal to an accepting activity
Signal Receipt	Signal Receipt	Used to represent that the signal is received
	Note/ Comment	Used to add relevant comments to elements

Activity diagram (contd..)



Activity Diagrams with Swimlanes

In activity diagrams swimlanes – also known as partitions – are used to represent or group actions carried out by different actors in a single thread. Here are a few tips you can follow when using swimlanes.

- Add swimlanes to linear processes. It makes it easy to read.
- Don't add more than 5 swimlanes.
- Arrange swimlanes in a logical manner.

Activity diagram (contd..)

How to draw an Activity diagram?

Step 1: Figure out the action steps for the use case.

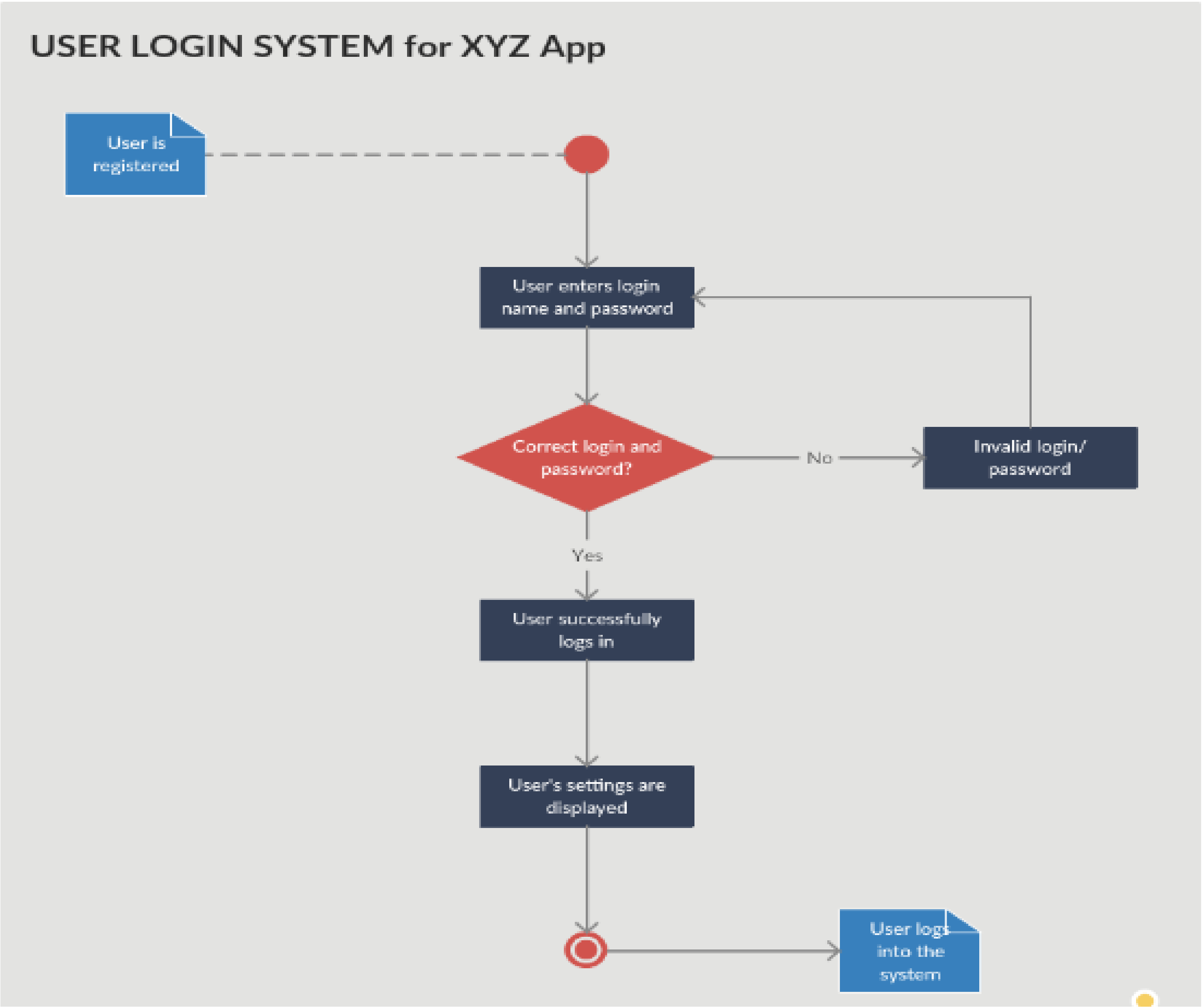
Step 2: Identify the actors who are involved.

Step 3: Find a flow among the activities.

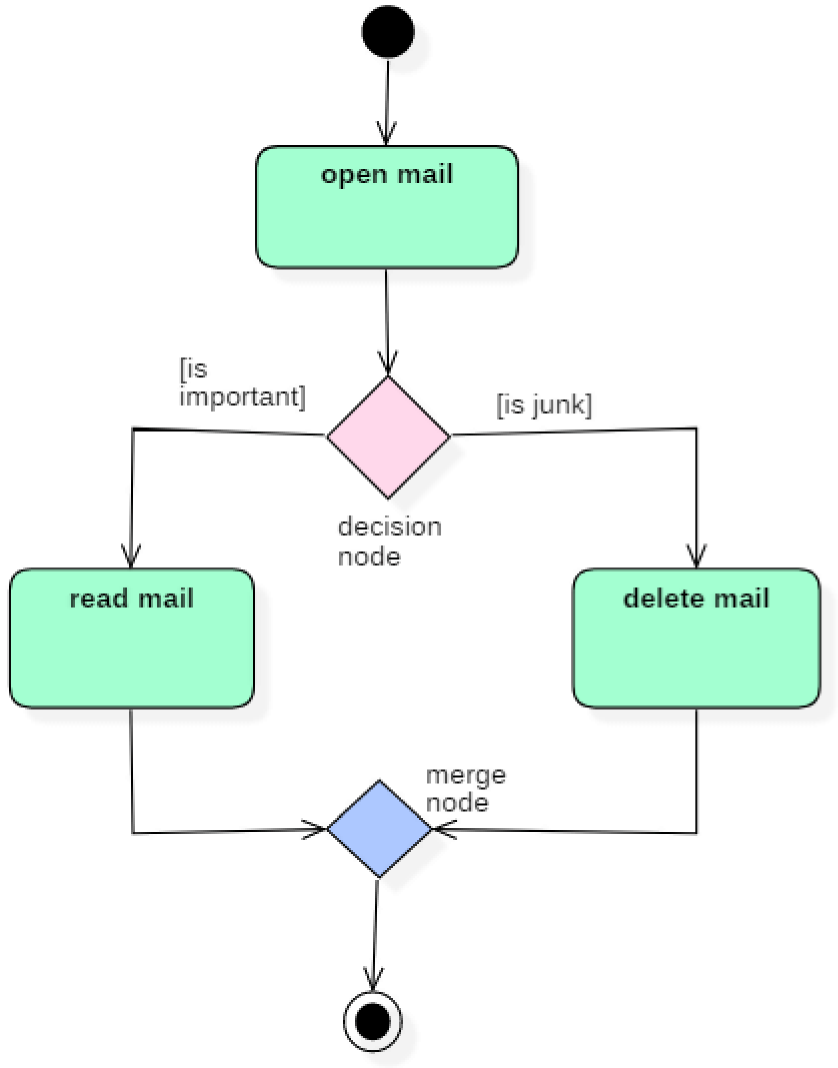
Step 4: Add swimlanes.

Activity diagram (contd..)

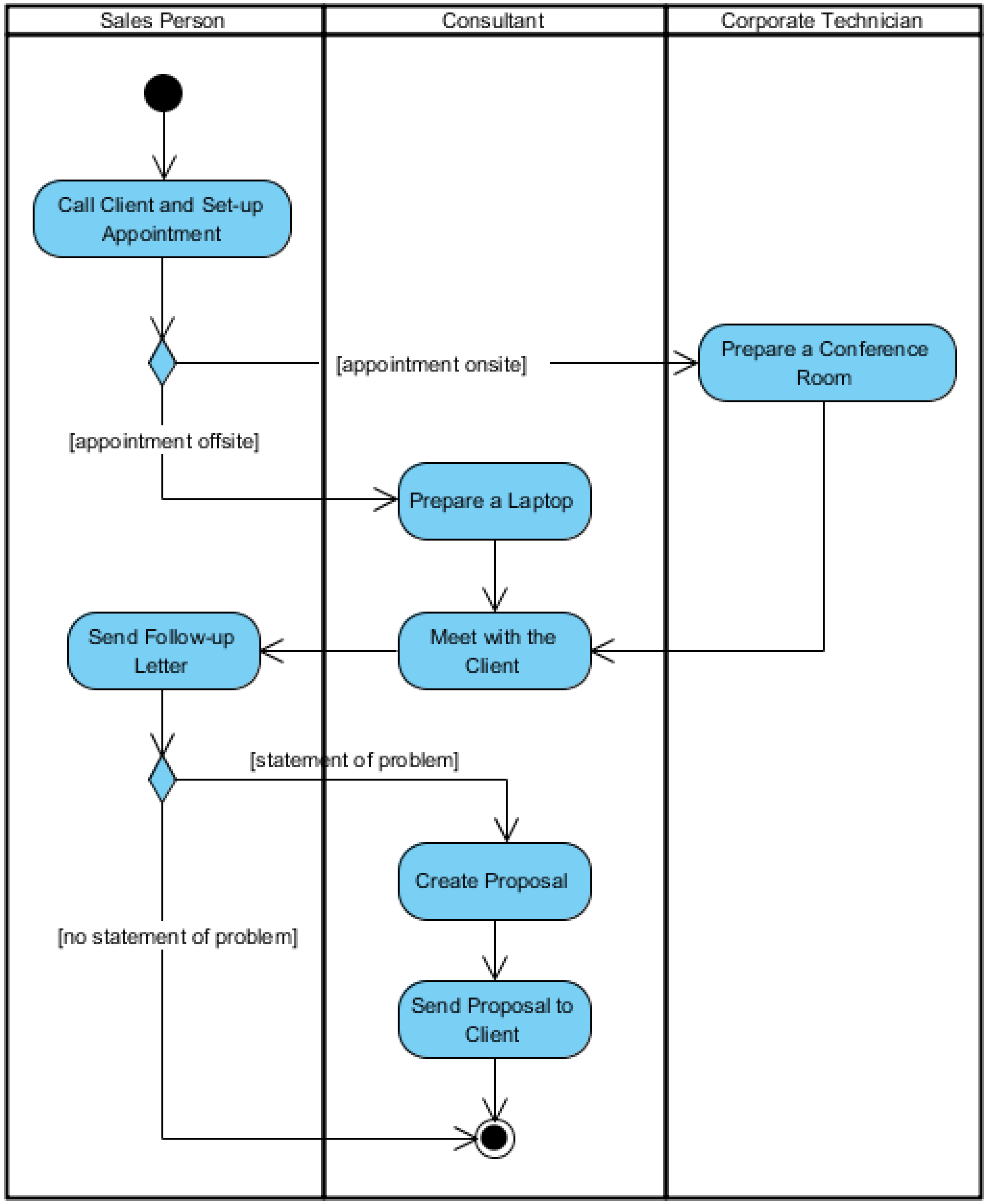
Activity Diagram for Login


USER LOGIN SYSTEM for XYZ App
User is registered
User enters login name and password
Correct login and password?
No
Invalid login/ password
Yes
User successfully logs in
User's settings are displayed
User logs into the system

Activity diagram for Mail Processing

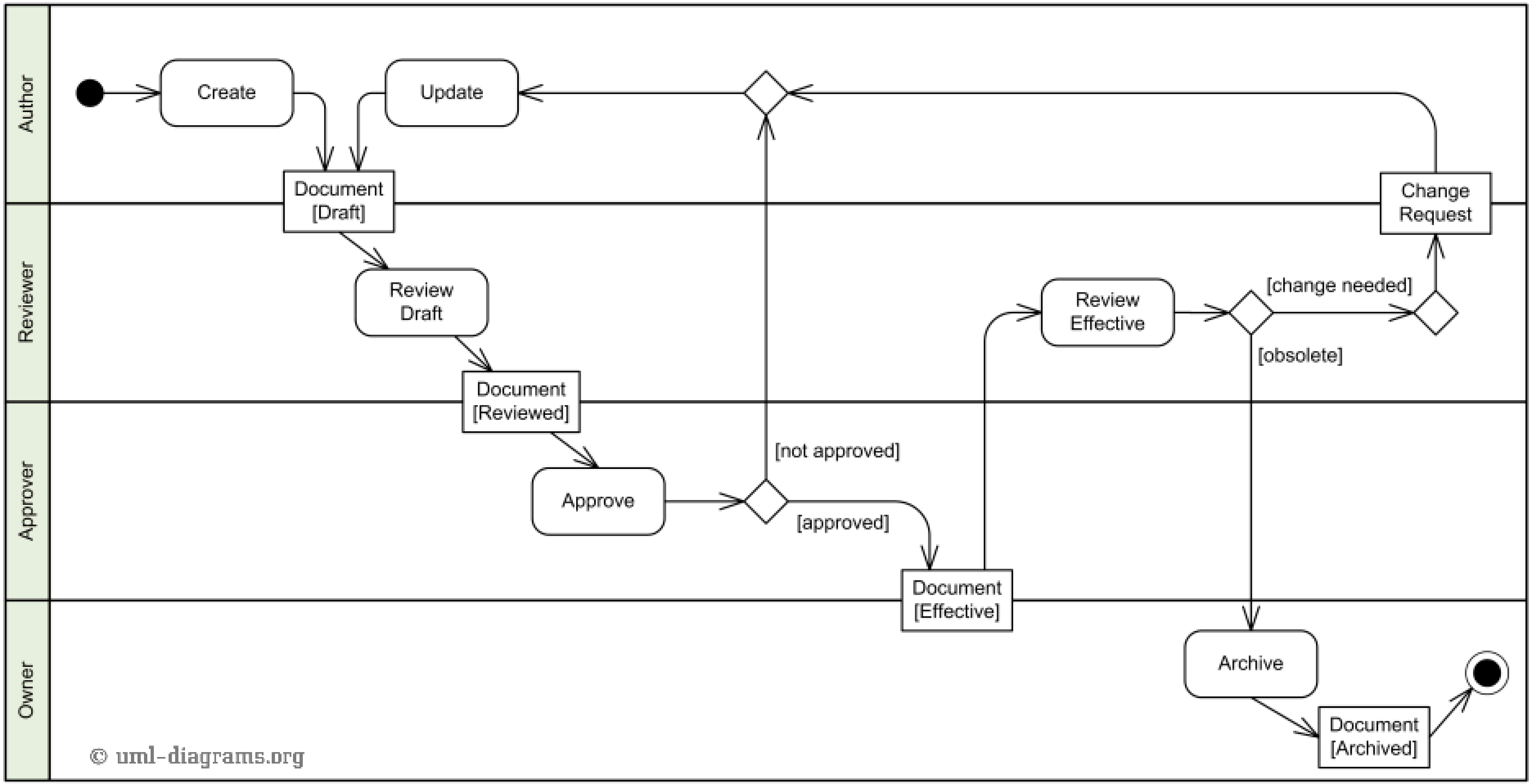


Activity diagram (contd..)



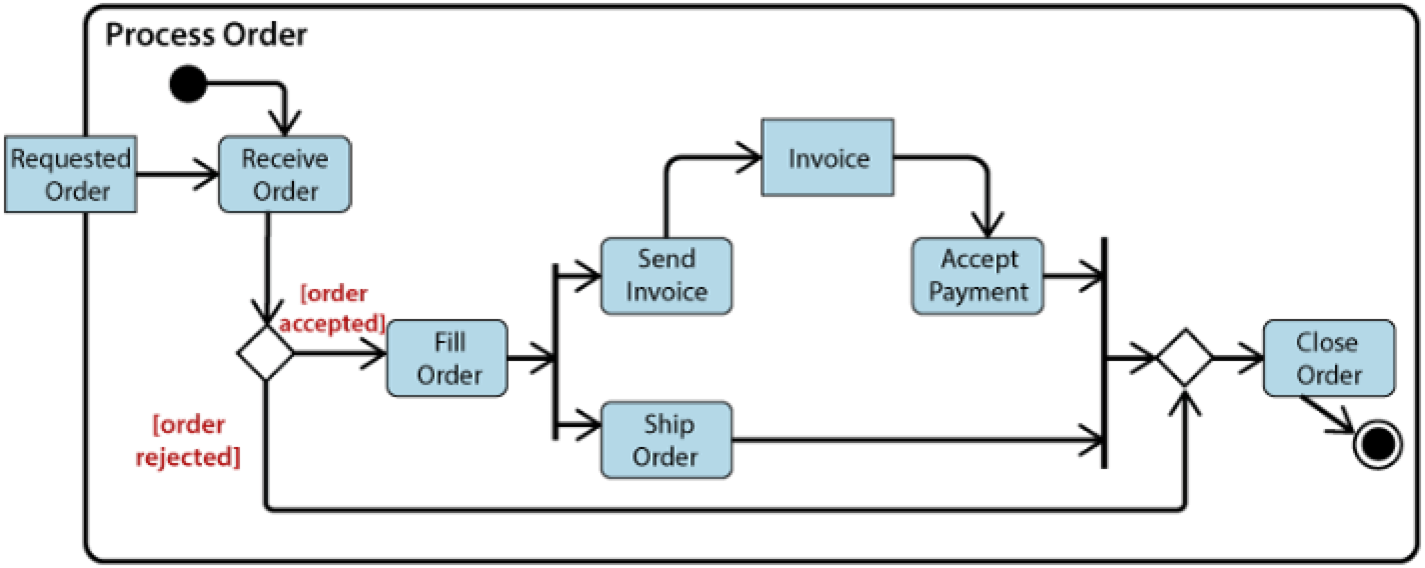
Activity diagram (contd..)

Document Management system (activity diagram)



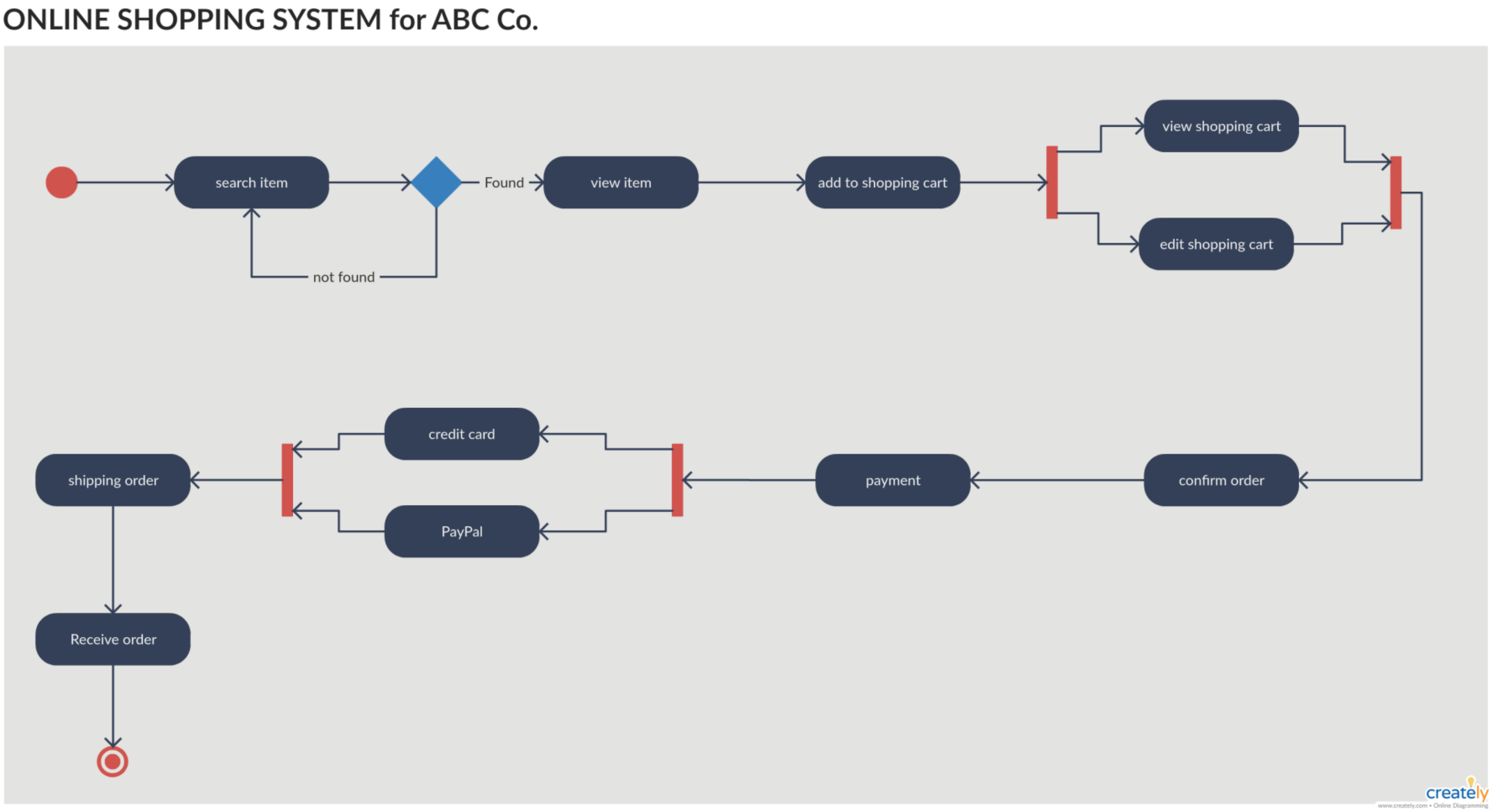
Activity diagram (contd..)

Order Processing (activity diagram)



Activity diagram (contd..)

Order Processing (activity diagram)



State Machines diagram/ State diagram

The state machine diagram is also called the Statechart or State Transition 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.

Types

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.

Protocol state machine

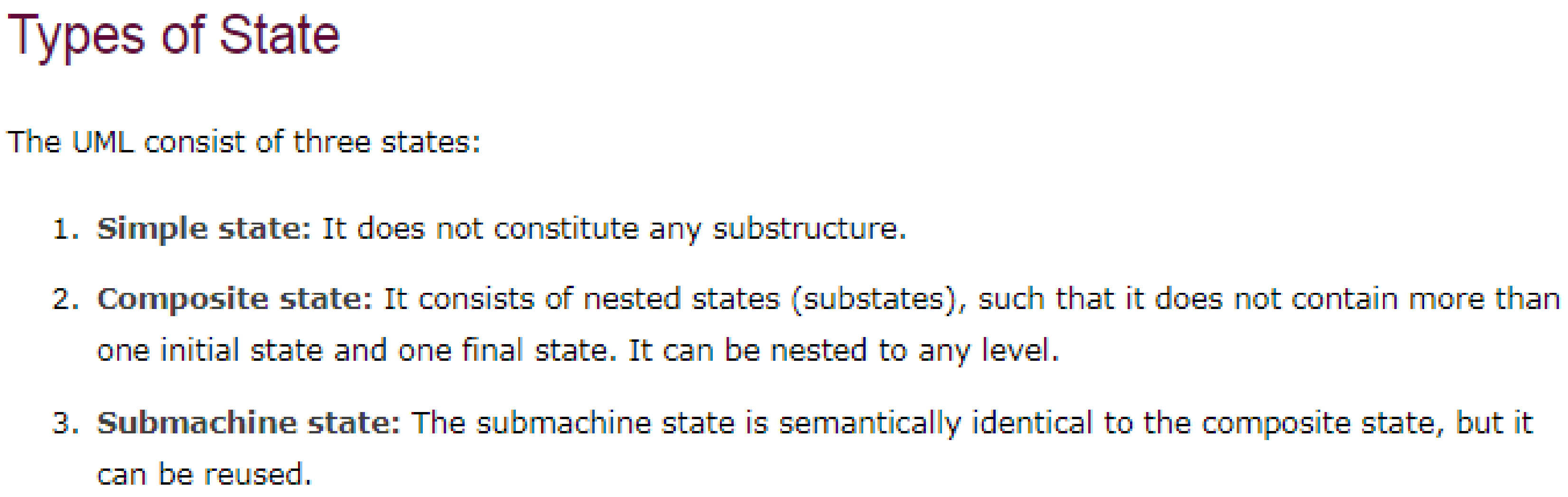
It captures the behavior of the protocol. The protocol state machine depicts the change in the state of the protocol and parallel changes within the system. But it does not portray the implementation of a particular component.

State Machines diagram/ State diagram (contd..)

Why State Machine Diagram??

As it records the dynamic view of a system, it portrays the behavior of a software application. During a lifespan, an object underwent several states, such that the lifespan exist until the program is executing. Each state depicts some useful information about the object.

It blueprints an interactive system that response back to either the internal events or the external ones. The execution flow from one state to another is represented by a state machine diagram. It visualizes an object state from its creation to its termination.

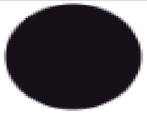


Types of State

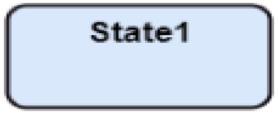
The UML consist of three states:

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.

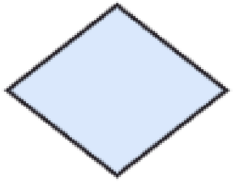
Notations of State Machines diagram



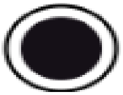
Initial state



State-box



Decision-box



Final State

a. **Initial state:** It defines the initial state (beginning) of a system, and it is represented by a black filled circle.

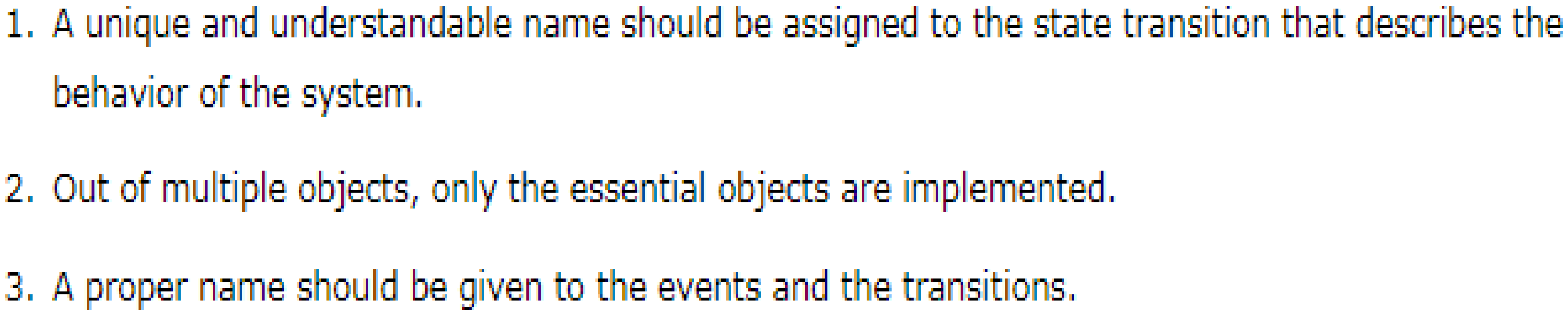
b. **Final state:** It represents the final state (end) of a system. It is denoted by a filled circle present within a circle.

c. **Decision box:** It is of diamond shape that represents the decisions to be made on the basis of an evaluated guard.

d. **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.

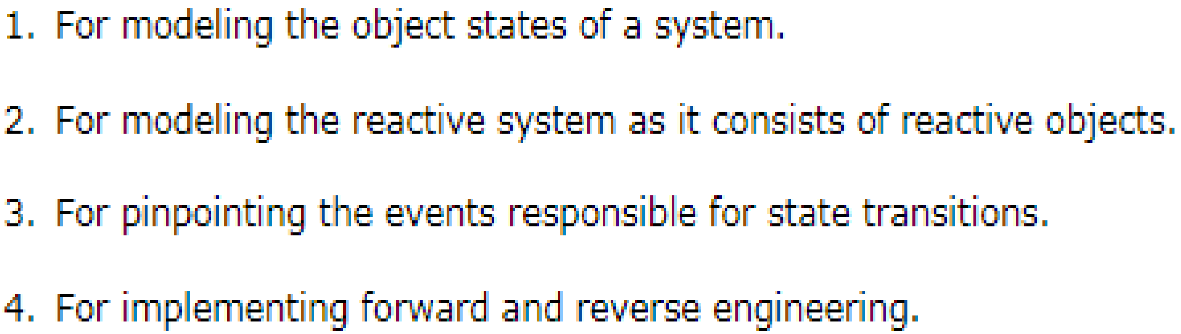
e. **State box:** It depicts the conditions or circumstances of a particular object of a class at a specific point of time. A rectangle with round corners is used to represent the state box.

Steps to draw a State Machines diagram



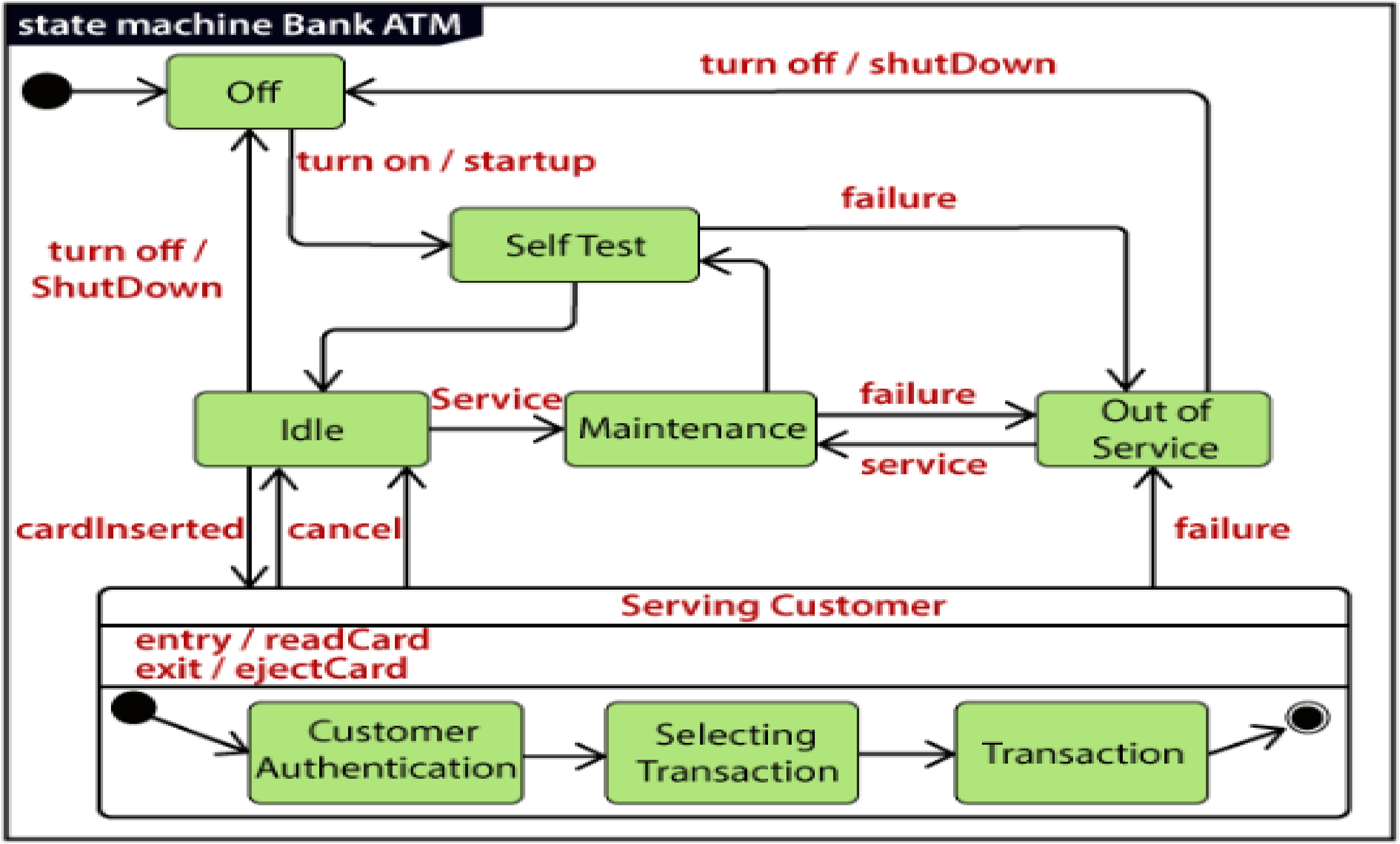
1. A unique and understandable name should be assigned to the state transition that describes the behavior of the system.
2. Out of multiple objects, only the essential objects are implemented.
3. A proper name should be given to the events and the transitions.

When to use a State Machines diagram

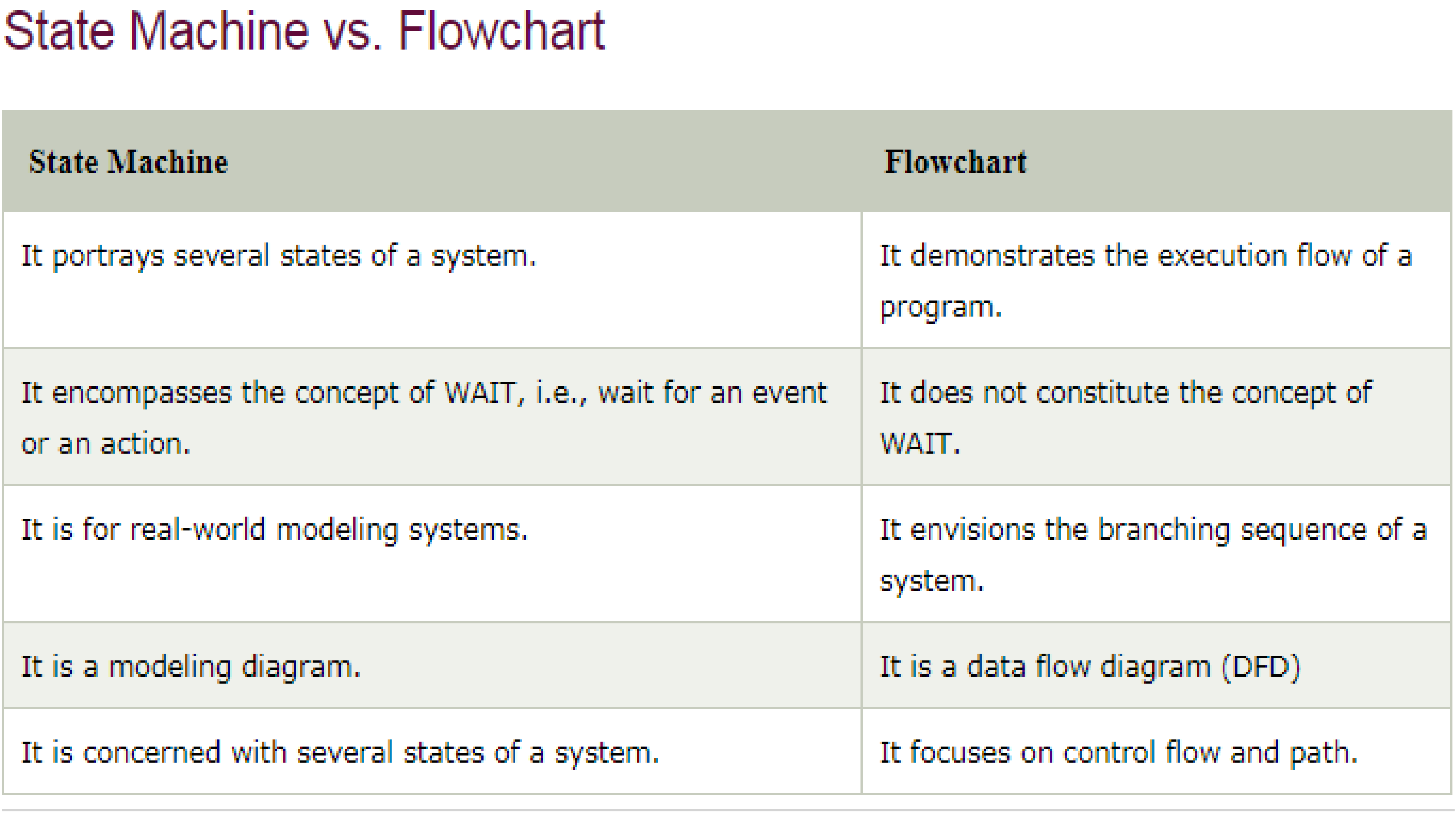


1. For modeling the object states of a system.
2. For modeling the reactive system as it consists of reactive objects.
3. For pinpointing the events responsible for state transitions.
4. For implementing forward and reverse engineering.

State Machines diagram for **Bank Automated Teller Machine (ATM)**



State Machines diagram (contd..)



State Machine vs. Flowchart

State Machine	Flowchart
It portrays several states of a system.	It demonstrates the execution flow of a program.
It encompasses the concept of WAIT, i.e., wait for an event or an action.	It does not constitute the concept of WAIT.
It is for real-world modeling systems.	It envisions the branching sequence of a system.
It is a modeling diagram.	It is a data flow diagram (DFD)
It is concerned with several states of a system.	It focuses on control flow and path.

Process and Thread

Process:

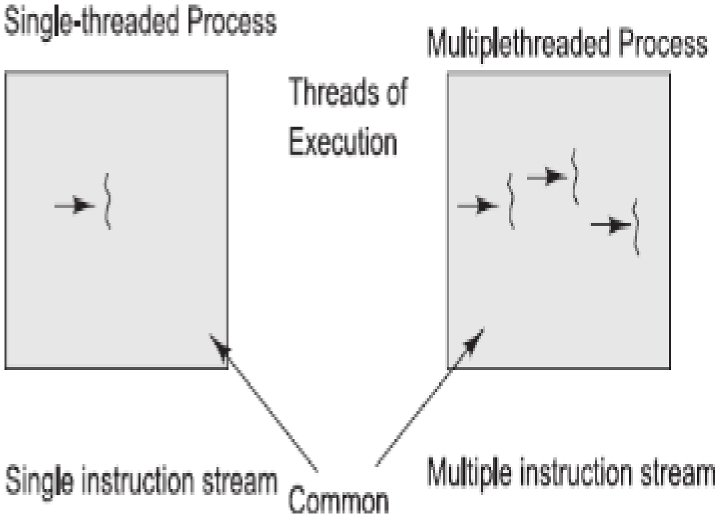
Process means any program is in execution. Process control block controls the operation of any process. Process control block contains information about processes for example Process priority, process id, process state, CPU, register, etc. A process can creates other processes which are known as **Child Processes**. Process takes more time to terminate and it is isolated means it does not share memory with any other process.
The process can have the following states like new, ready, running, waiting, terminated, suspended.

Thread:

Thread is the segment of a process means a process can have multiple threads and these multiple threads are contained within a process. A thread have 3 states: running, ready, and blocked.
Thread takes less time to terminate as compared to process and like process threads do not isolate.

Process and Thread

To understand multithreading, the concepts process and thread must be understood. A process is a program in execution. A process may be divided into a number of independent units known as threads. A thread is a dispatchable unit of work. Threads are light-weight processes within a process . A process is a collection of one or more threads and associated system resources. The difference between a process and a thread is shown in Fig.14.1. A process may have a number of threads in it. A thread may be assumed as a subset of a process.

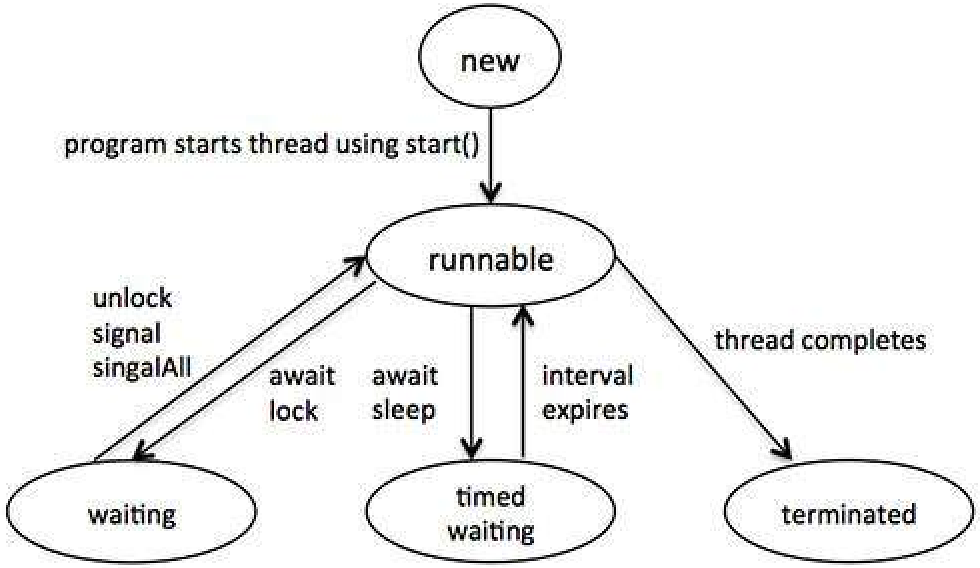


If two applications are run on a computer (MS Word, MS Access), two processes are created. Multitasking of two or more processes is known as process-based multitasking. Multitasking of two or more threads is known as thread-based multitasking. The concept of multithreading in a programming language refers to thread-based multitasking. Process-based multitasking is totally controlled by the operating system. But thread-based multitasking can be controlled by the programmer to some extent in a program.

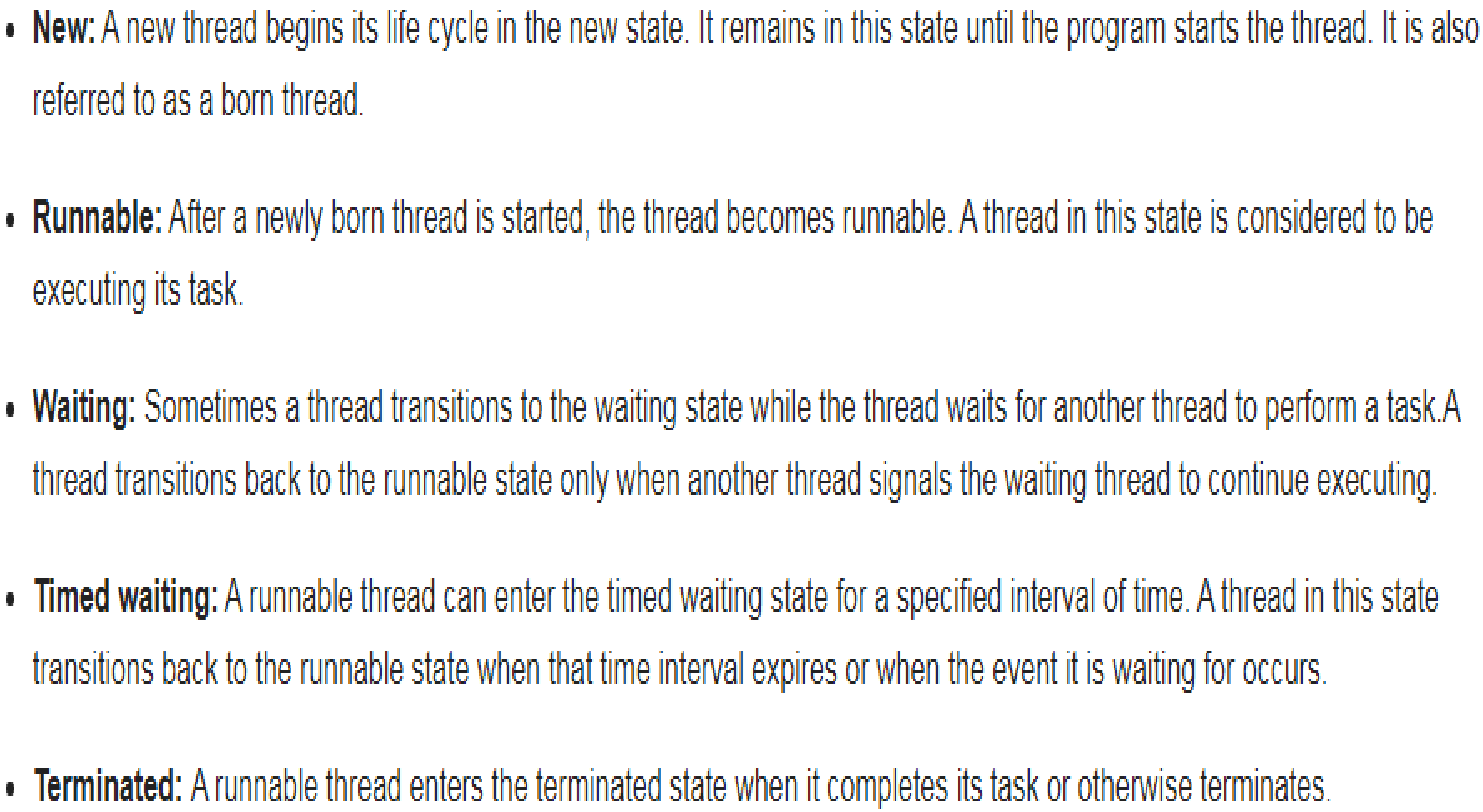
Process and Thread (contd..)

Life Cycle of a Thread:

A thread goes through various stages in its life cycle. For example, a thread is born, started, runs, and then dies. Following diagram shows complete life cycle of a thread.

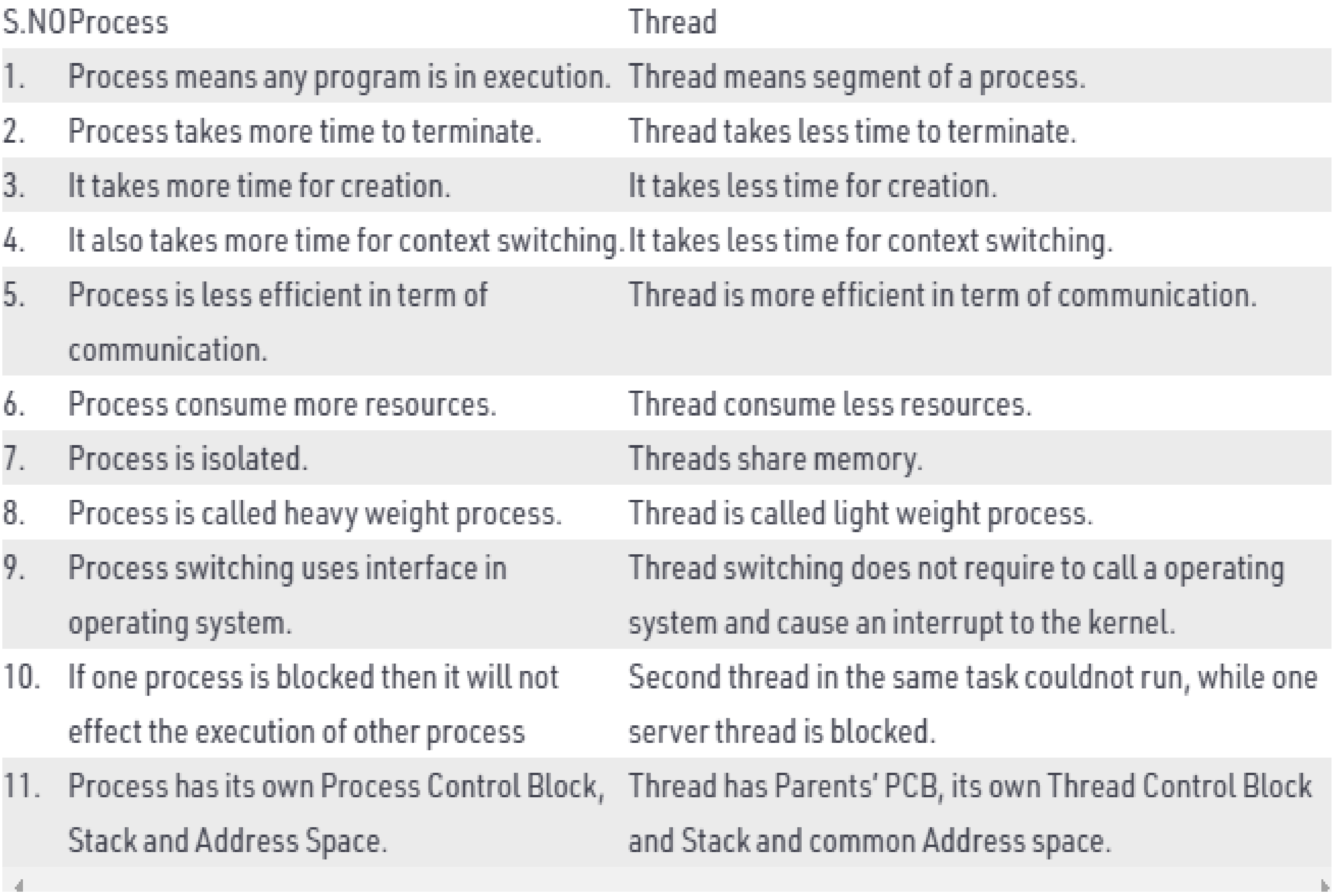


Process and Thread (contd..)



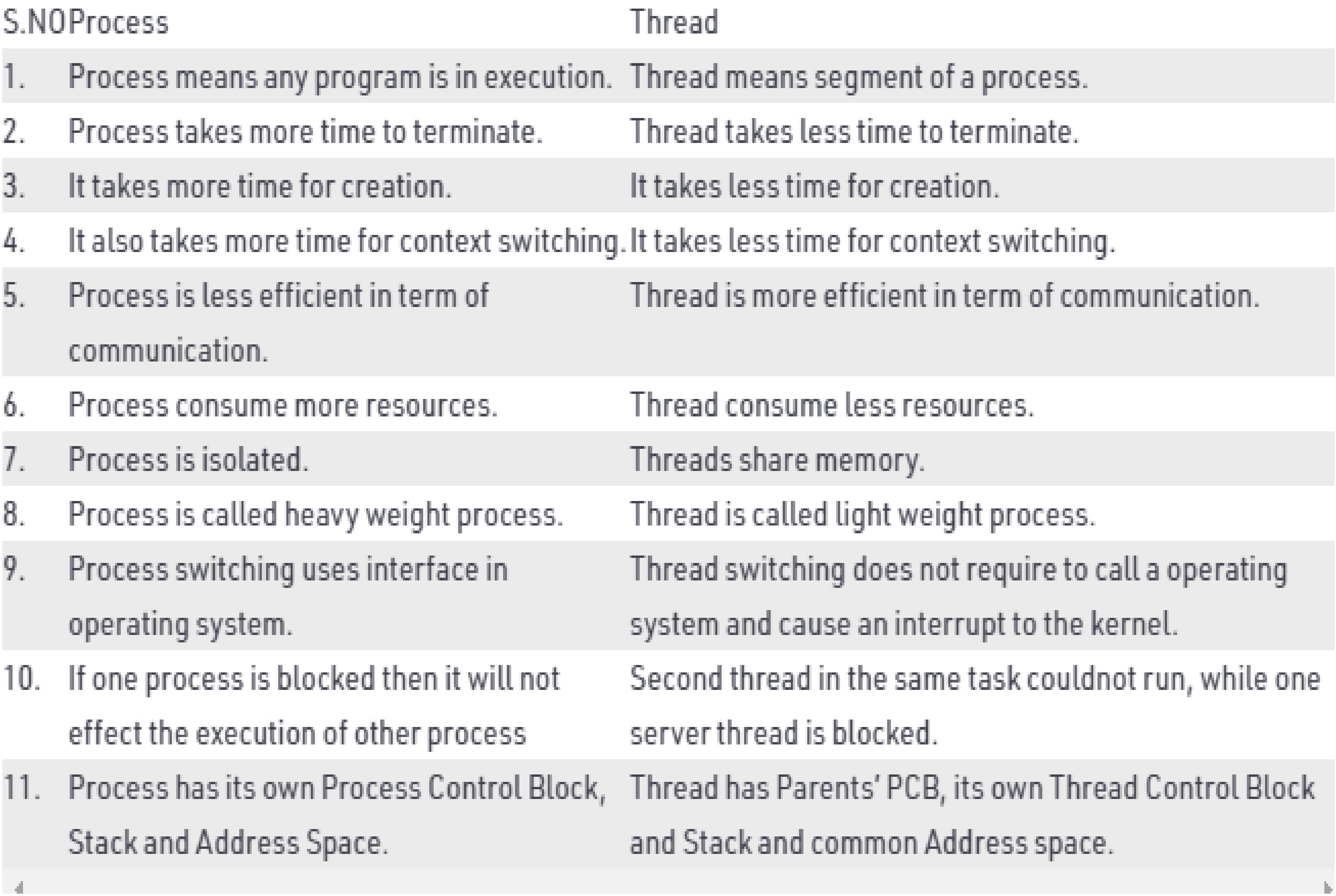
- **New:** A new thread begins its life cycle in the new state. It remains in this state until the program starts the thread. It is also referred to as a born thread.
- **Runnable:** After a newly born thread is started, the thread becomes runnable. A thread in this state is considered to be executing its task.
- **Waiting:** Sometimes a thread transitions to the waiting state while the thread waits for another thread to perform a task.A thread transitions back to the runnable state only when another thread signals the waiting thread to continue executing.
- **Timed waiting:** A runnable thread can enter the timed waiting state for a specified interval of time. A thread in this state transitions back to the runnable state when that time interval expires or when the event it is waiting for occurs.
- **Terminated:** A runnable thread enters the terminated state when it completes its task or otherwise terminates.

Difference between Process and Thread



S.NO	Process	Thread
1.	Process means any program is in execution.	Thread means segment of a process.
2.	Process takes more time to terminate.	Thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	Process is less efficient in term of communication.	Thread is more efficient in term of communication.
6.	Process consume more resources.	Thread consume less resources.
7.	Process is isolated.	Threads share memory.
8.	Process is called heavy weight process.	Thread is called light weight process.
9.	Process switching uses interface in operating system.	Thread switching does not require to call a operating system and cause an interrupt to the kernel.
10.	If one process is blocked then it will not effect the execution of other process	Second thread in the same task couldnot run, while one server thread is blocked.
11.	Process has its own Process Control Block, Stack and Address Space.	Thread has Parents' PCB, its own Thread Control Block and Stack and common Address space.

Difference between Process and Thread



S.NO	Process	Thread
1.	Process means any program is in execution.	Thread means segment of a process.
2.	Process takes more time to terminate.	Thread takes less time to terminate.
3.	It takes more time for creation.	It takes less time for creation.
4.	It also takes more time for context switching.	It takes less time for context switching.
5.	Process is less efficient in term of communication.	Thread is more efficient in term of communication.
6.	Process consume more resources.	Thread consume less resources.
7.	Process is isolated.	Threads share memory.
8.	Process is called heavy weight process.	Thread is called light weight process.
9.	Process switching uses interface in operating system.	Thread switching does not require to call a operating system and cause an interrupt to the kernel.
10.	If one process is blocked then it will not effect the execution of other process	Second thread in the same task couldnot run, while one server thread is blocked.
11.	Process has its own Process Control Block, Stack and Address Space.	Thread has Parents' PCB, its own Thread Control Block and Stack and common Address space.

End