Unified
Modeling
Language

UML - Standard Diagrams

- Structural Diagrams
- Behavioral Diagrams

Structural Diagrams

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

Class diagram

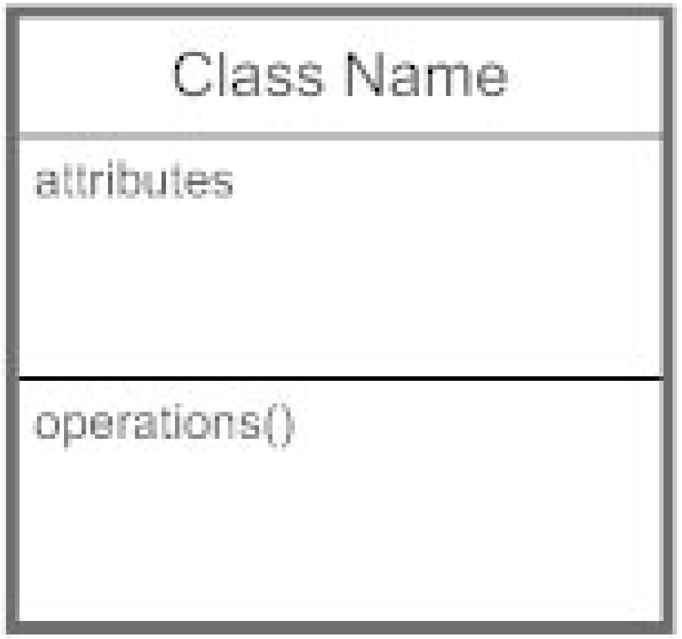
- **UML CLASS DIAGRAM** gives an overview of a software system by displaying classes, attributes, operations, and their relationships.
- This Diagram includes the class name, attributes, and operation in separate designated compartments.
- Class Diagram defines the types of objects in the system and the different types of relationships that exist among them.

Class diagram

- **Benefits of Class Diagram**
- Class Diagram Illustrates data models for even very complex information systems
- It provides an overview of how the application is structured before studying the actual code. This can easily reduce the maintenance time
- Helpful for developers and other stakeholders.

Class diagram

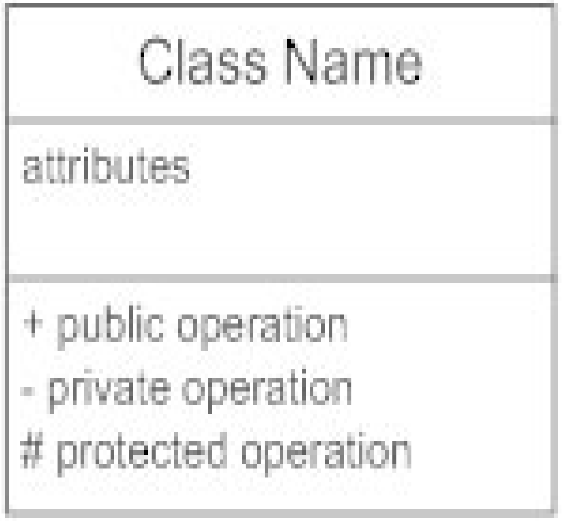
- Basic Class Diagram Symbols and Notations
- Class Name Written in Italics : Abstract Class



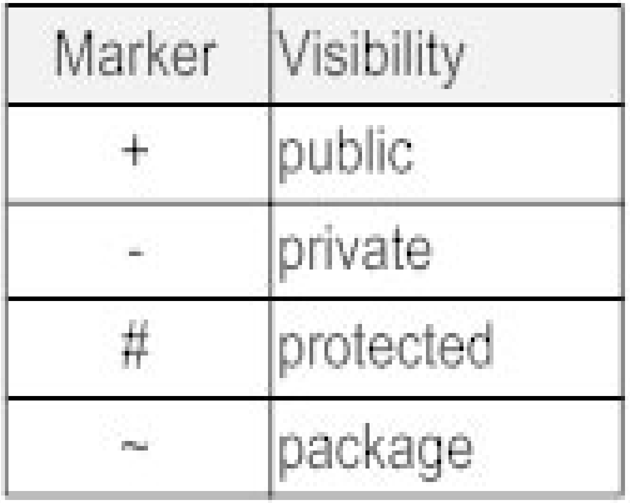
Class diagram

- Visibility
- Use visibility markers to signify who can access the information contained within a class.
- Private visibility, denoted with a - sign, hides information from anything outside the class partition.
- Public visibility, denoted with a + sign, allows all other classes to view the marked information.
- Protected visibility, denoted with a # sign, allows child classes to access information they inherited from a parent class.

Class diagram



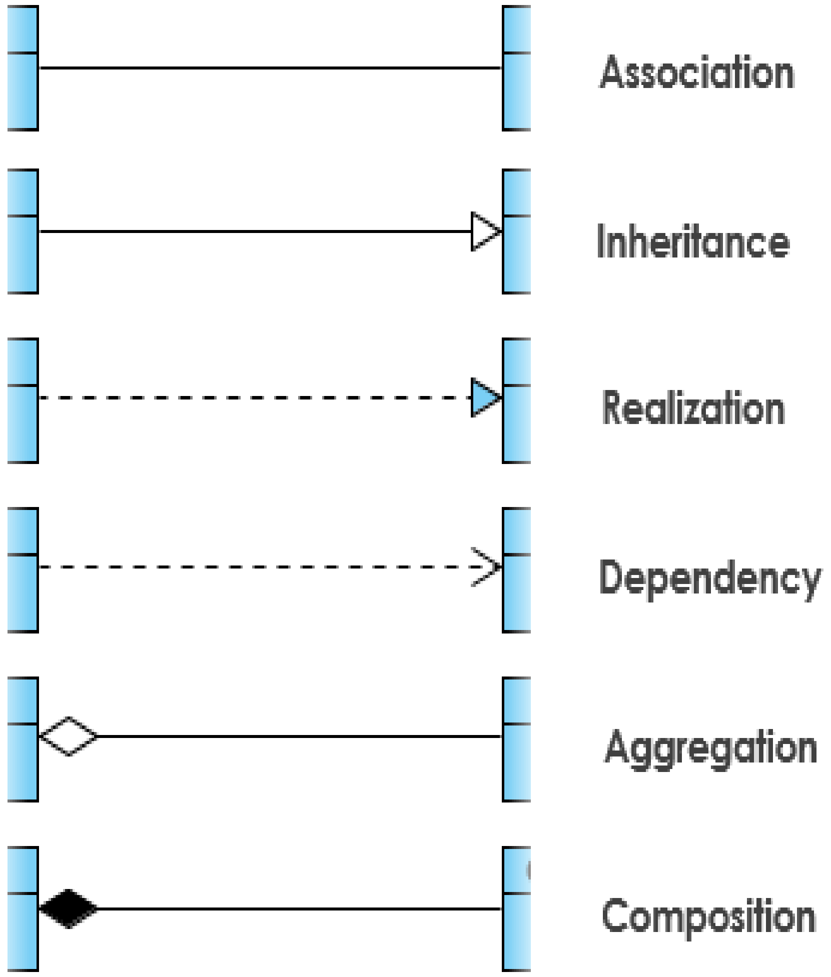
Visibility



Marker	Visibility
+	public
-	private
#	protected
~	package

Class diagram

- Relationships between classes
- UML is not just about pretty pictures.
- If used correctly, UML precisely conveys how code should be implemented from diagrams.
- If precisely interpreted, the implemented code will correctly reflect the intent of the designer.



Class diagram

- Relationships between classes

1. Association

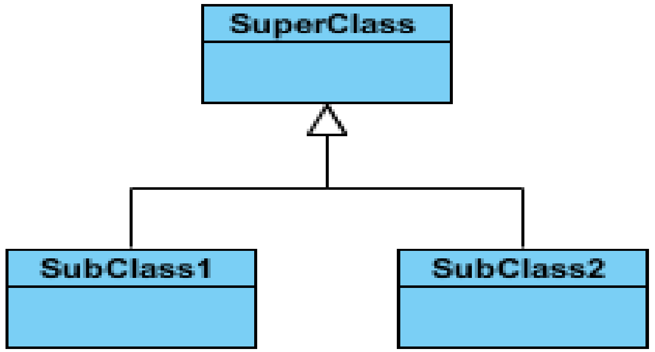
- Associations are relationships between classes in a UML Class Diagram.
- They are represented by a solid line between classes.

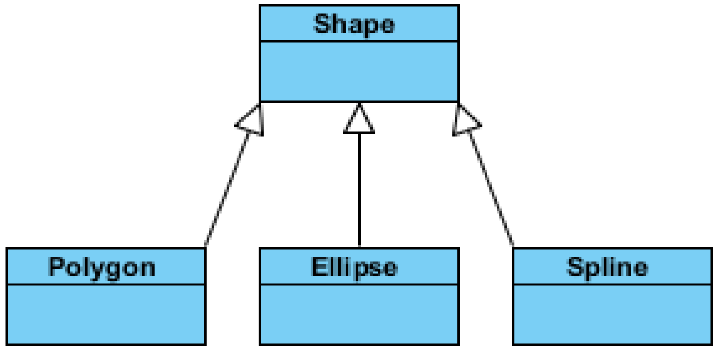
Class diagram

- Relationships between classes

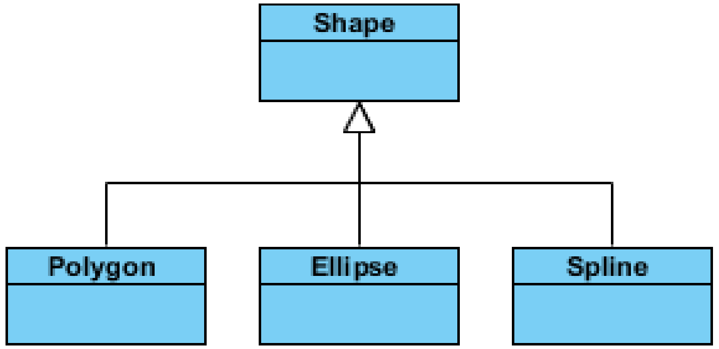
2. Inheritance (or Generalization):

- Represents an "is-a" relationship.
- An abstract class name is shown in italics.
- SubClass1 and SubClass2 are specializa





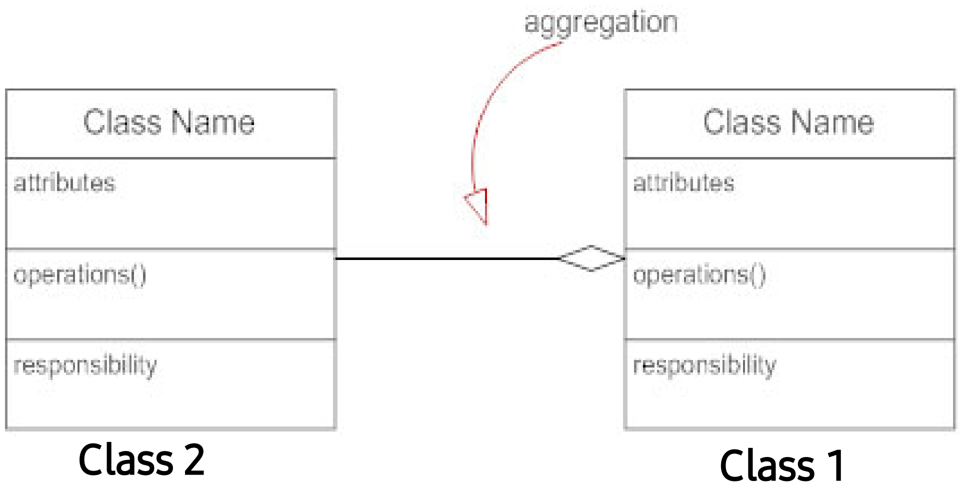
Style 1: Separate target

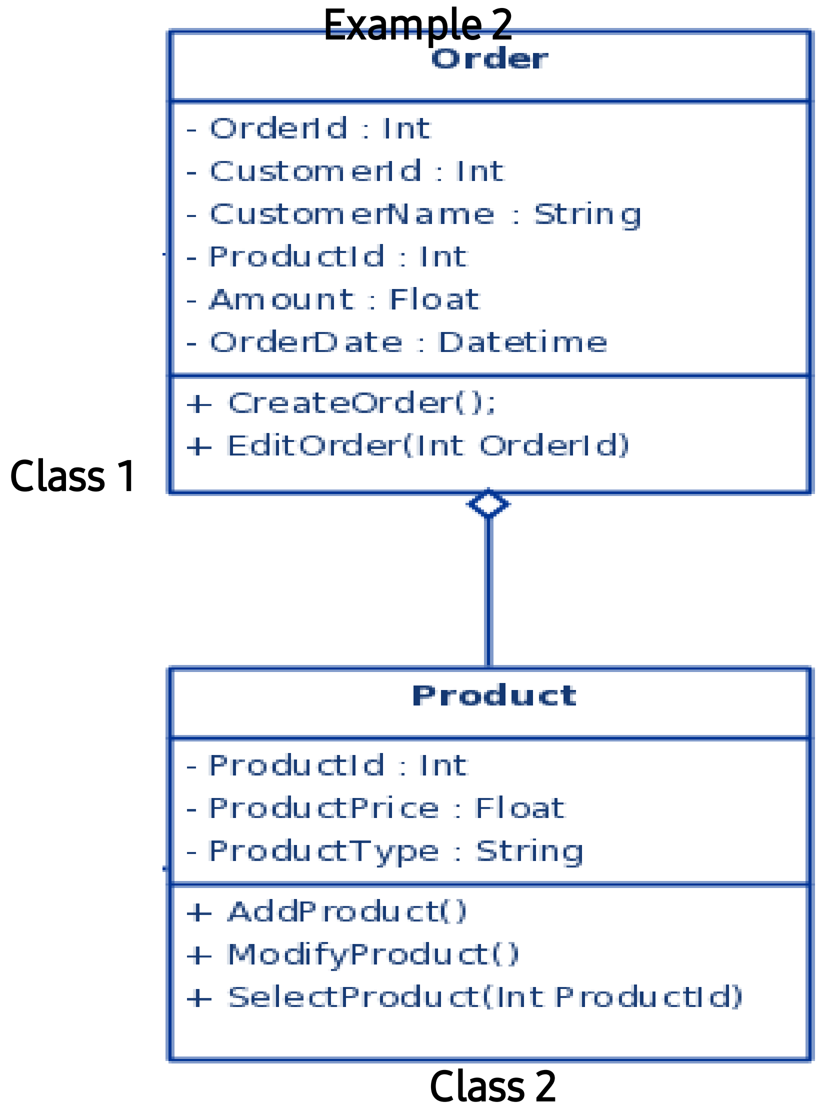


Style 2: Shared target

Class diagram

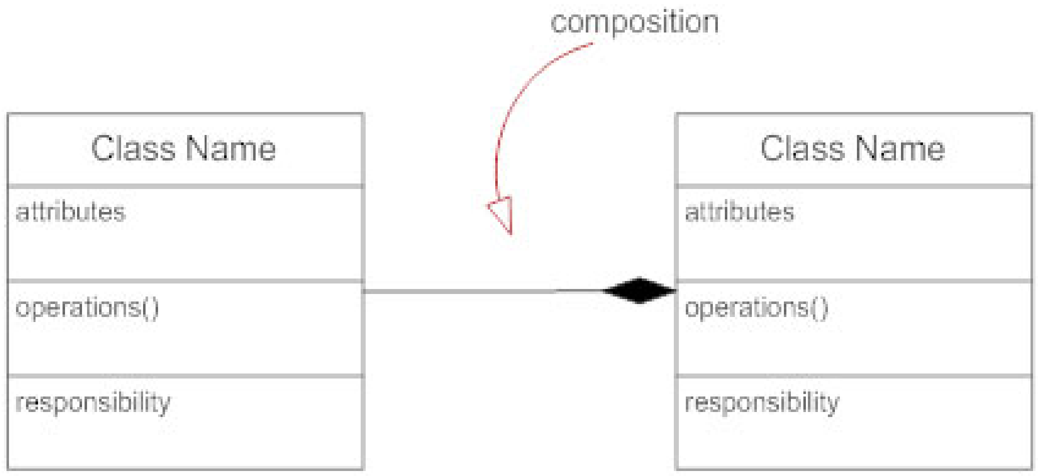
- Relationships between classes
- 3. Aggregation
- A special type of association.
- It represents a "part of" relationship.
- In the Example Class2 is part of Class1.
- Objects of Class1 and Class2 have separate lifetimes.





Class diagram

- Relationships between classes
- 4. Composition
- A special type of aggregation where parts are destroyed when the whole is destroyed.
- Objects of Class2 live and die with Class1.
- Class2 cannot stand by itself

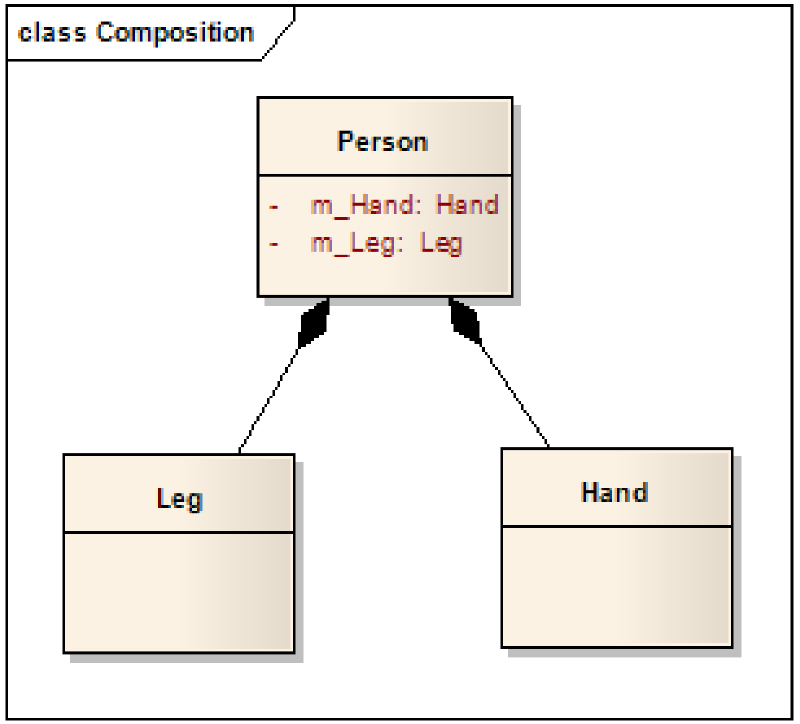


Class 2

Class 1

Example 2

Class 1

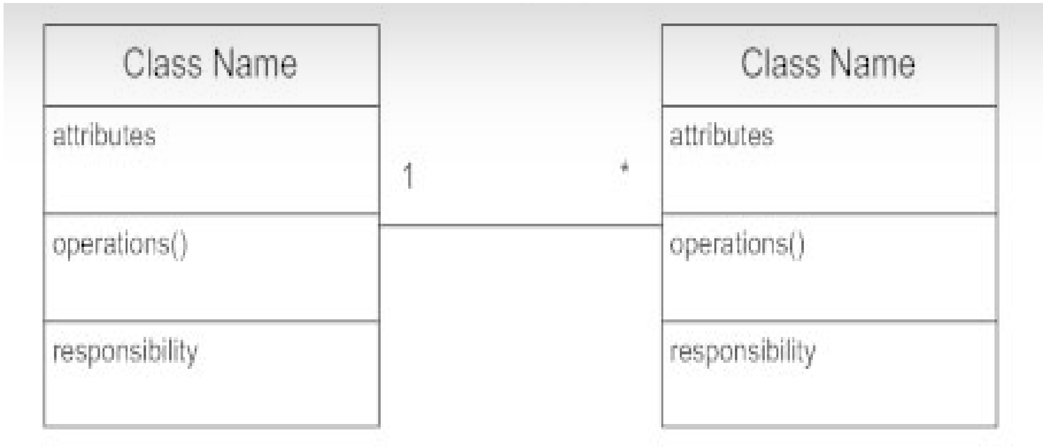


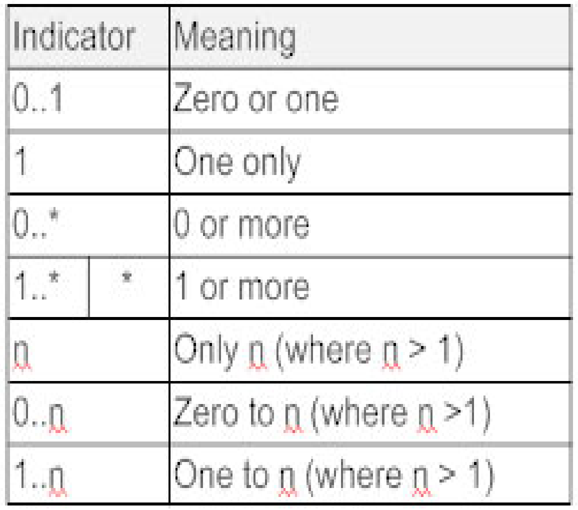
Class 2

Class diagram

- Multiplicity (Cardinality)

- Place multiplicity notations near the ends of an association.
- These symbols indicate the number of instances of one class linked to one instance of the other class.
- For example, one company will have one or more employees, but each employee works for just one company.



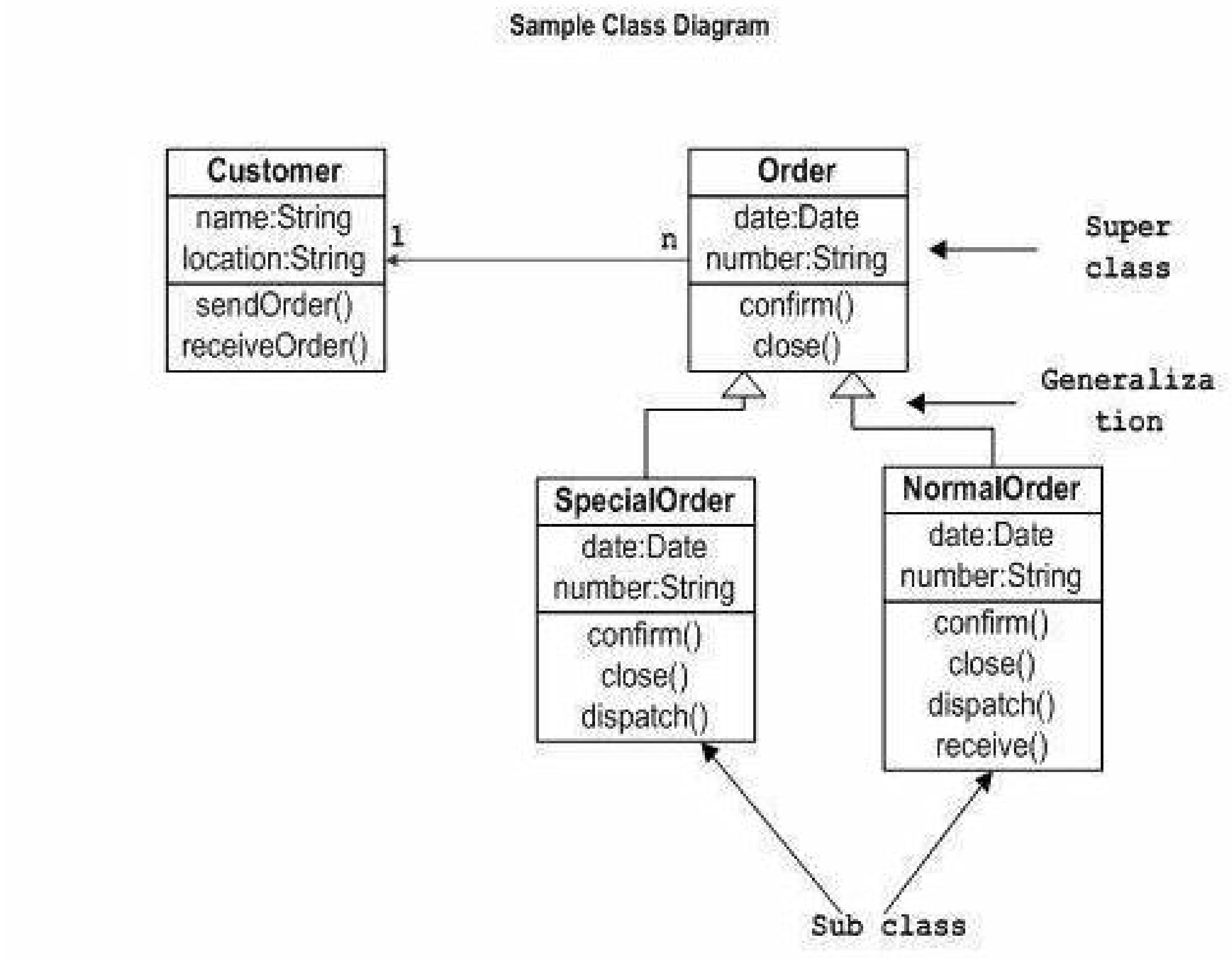


Indicator	Meaning
0..1	Zero or one
1	One only
0..*	0 or more
1..* *	1 or more
n	Only n (where n > 1)
0..n	Zero to n (where n >1)
1..n	One to n (where n > 1)

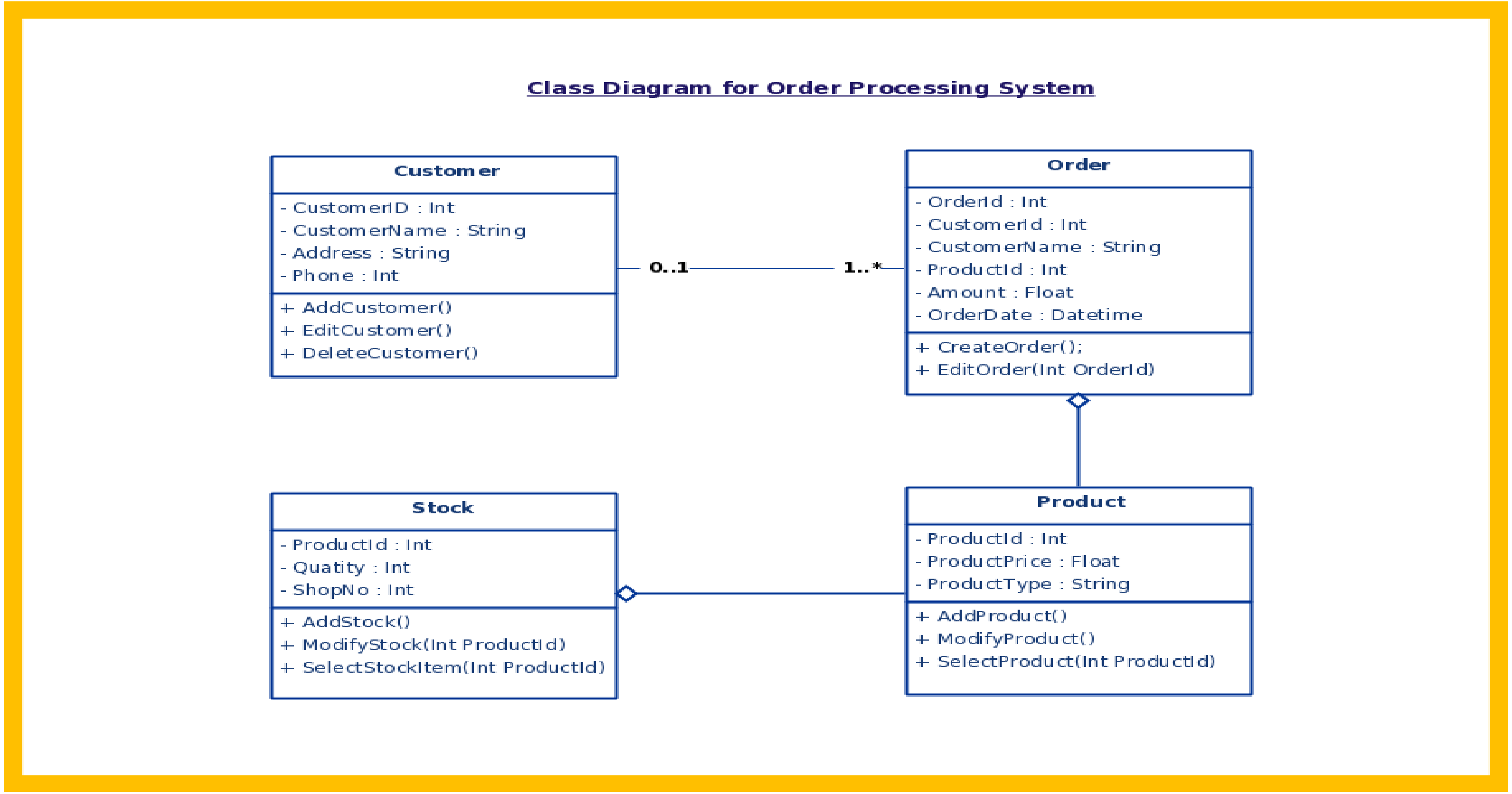
Examples

- Order System of an application

Class diagram - Static view of an application - Example 1



Class diagram - Static view of an application - Example 1

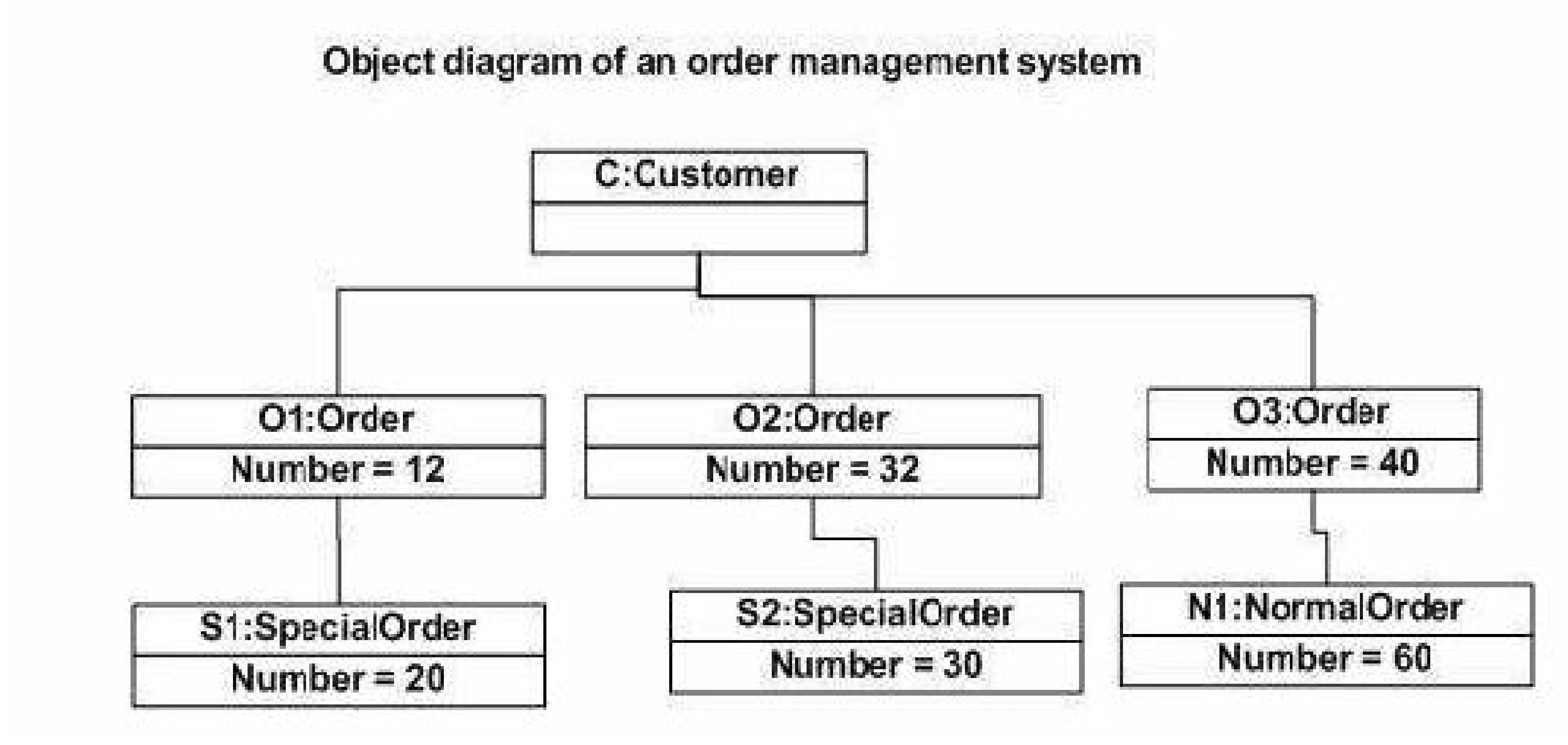


OBJECT DIAGRAM

OBJECT DIAGRAM

- Object diagrams are derived from class diagrams so object diagrams are dependent upon class diagrams.
- Object diagrams represent an instance of a class diagram.
- The basic concepts are similar for class diagrams and object diagrams.
- Object diagrams also represent the static view of a system but this static view is a snapshot of the system at a particular moment
- **Purpose of Object Diagrams**
- A class diagram represents an abstract model consisting of classes and their relationships.
- However, an object diagram represents an instance at a particular moment, which is concrete in nature.
- It means the object diagram is closer to the actual system behaviour.
- The purpose is to capture the static view of a system at a particular moment.

Object Diagrams – instance of a class diagram



OBJECT DIAGRAM

- Explanation of Example Diagram
- The above diagram is an example of an object diagram.
- It represents the Order management system.
- The following diagram is an instance of the system at a particular time of purchase.
- It has the following objects.
- Customer
- Order
- SpecialOrder
- NormalOrder

OBJECT DIAGRAM

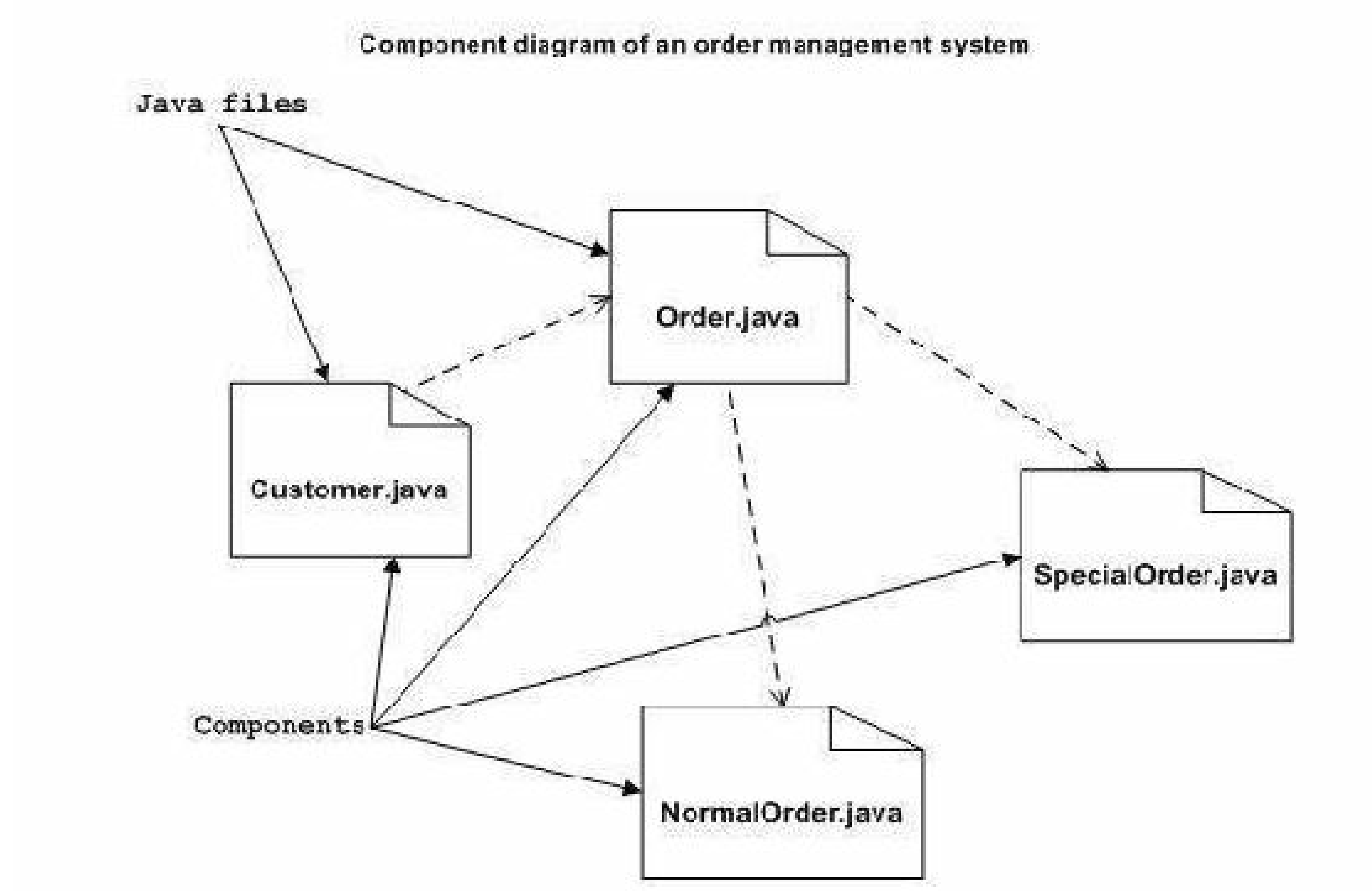
- Explanation of Example Diagram
- Now the customer object (C) is associated with three order objects (O1, O2, and O3).
- These order objects are associated with special order and normal order objects (S1, S2, and N1).
- The customer has the following three orders with different numbers (12, 32 and 40) for the particular time considered.
- The customer can increase the number of orders in future and in that scenario the object diagram will reflect that.
- If order, special order, and normal order objects are observed then you will find that they have some values.

COMPONENT DIAGRAM

Component Diagrams

- Component diagrams are used to model the physical aspects of a system.
- Component diagrams are used during the implementation phase of an application
- what are physical aspects?
- Physical aspects are the elements such as executables, libraries, files, documents, etc. which reside in a node.
- Purpose of Component Diagrams
- It does not describe the functionality of the system but it describes the components used to make those functionalities.
- It Visualize the components of a system.

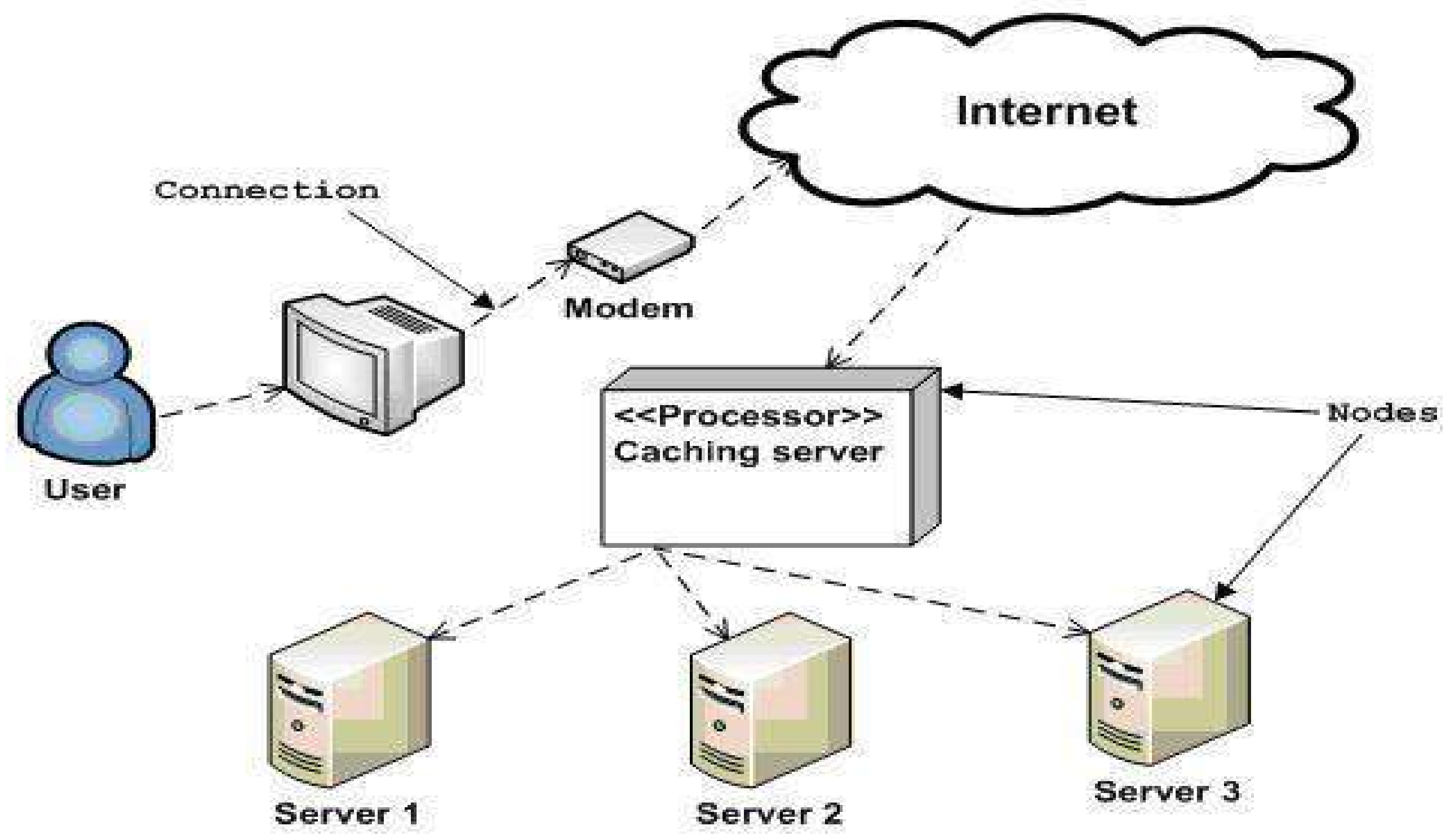
Component Diagrams



DEPLOYMENT DIAGRAM

- Deployment diagrams are used to visualize the topology of the physical components of a system, where the software components are deployed.
- Deployment diagrams are used for describing the hardware components, where software components are deployed.
- Component diagrams and deployment diagrams are closely related
- Component diagrams are used to describe the components and deployment diagrams shows how they are deployed in hardware.

Deployment Diagrams



Behavioral Diagrams – Behaviour of the system when it is running/operating

- Use case diagram
- Sequence diagram
- Collaboration diagram
- Statechart diagram
- Activity diagram

Use Case Diagrams

- use case diagram is dynamic in nature, there should be some internal or external factors for making the interaction.
- These internal and external agents are known as actors.
- Use case diagrams consists of actors, use cases and their relationships.
- A single use case diagram captures a particular functionality of a system.

Use Case Diagrams - Used to gather the requirements of a system including internal and external influences

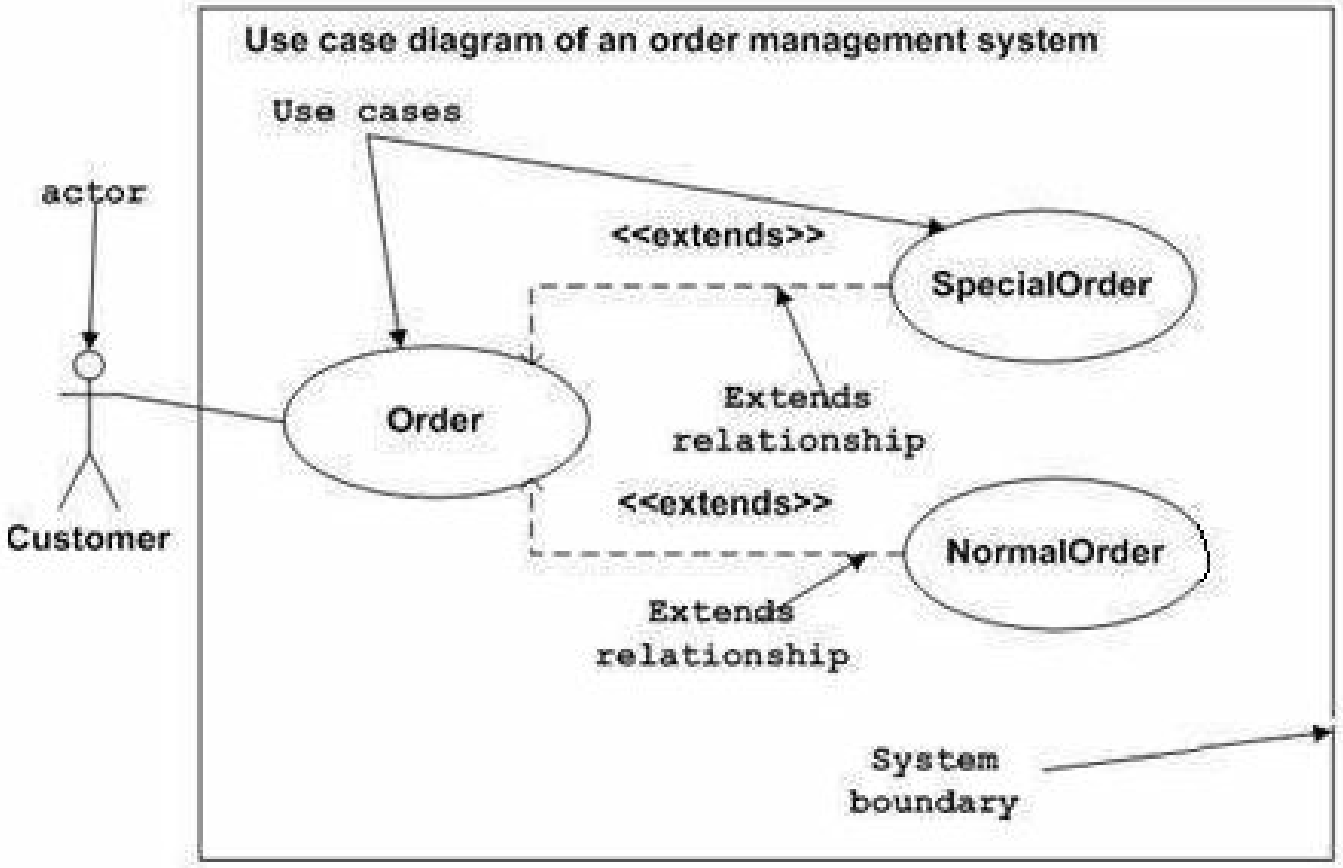
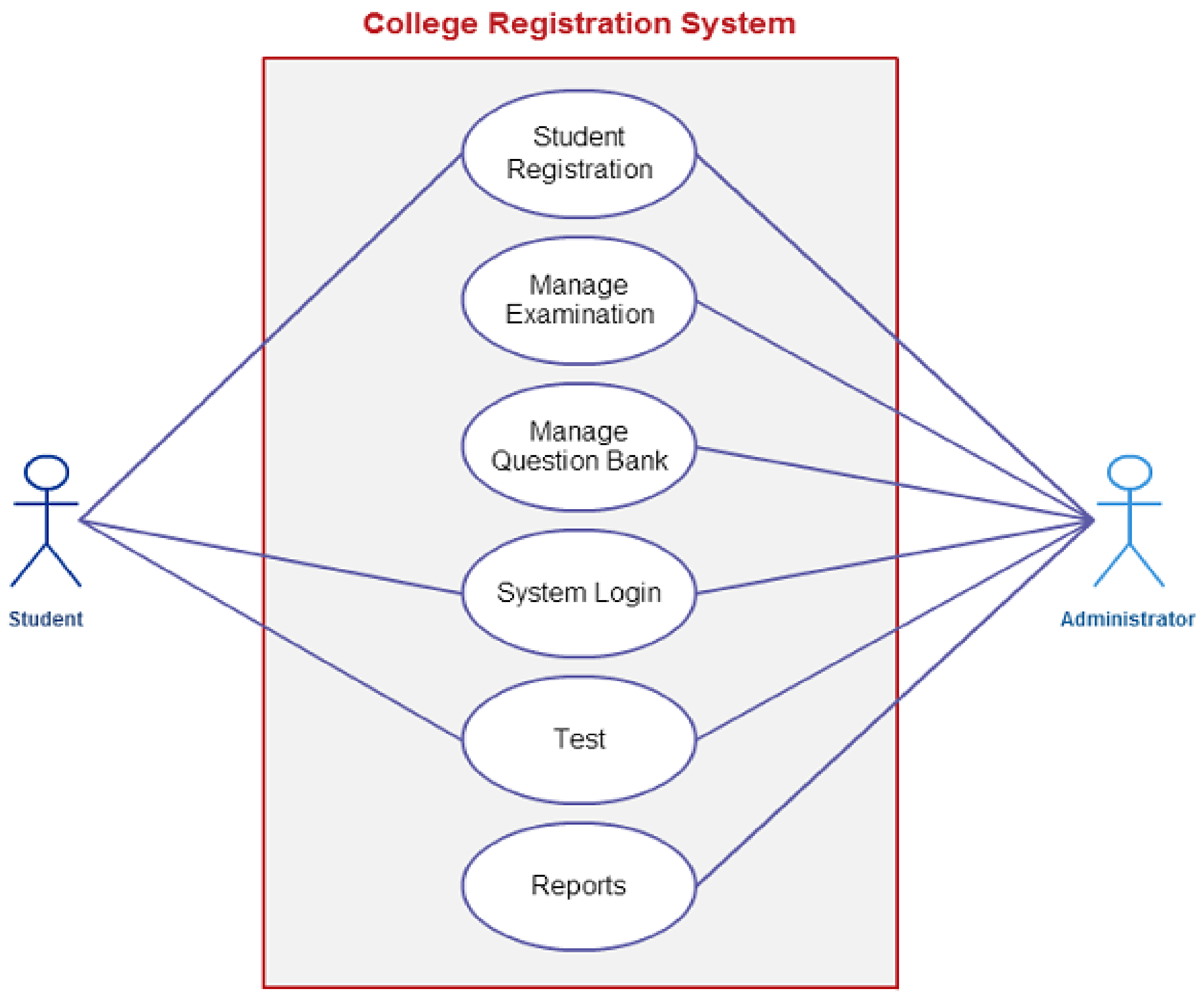


Figure: Sample Use Case diagram

Use Case Diagrams

- Sample use case diagram representing the order management system.
- Hence, if we look into the diagram then we will find three use cases **(Order, SpecialOrder, and NormalOrder)** and one actor which is the customer.
- The SpecialOrder and NormalOrder use cases are extended from *Order* use case. Hence, they have extended relationship.
- Another important point is to identify the system boundary, which is shown in the picture.
- The actor Customer lies outside the system as it is an external user of the system.

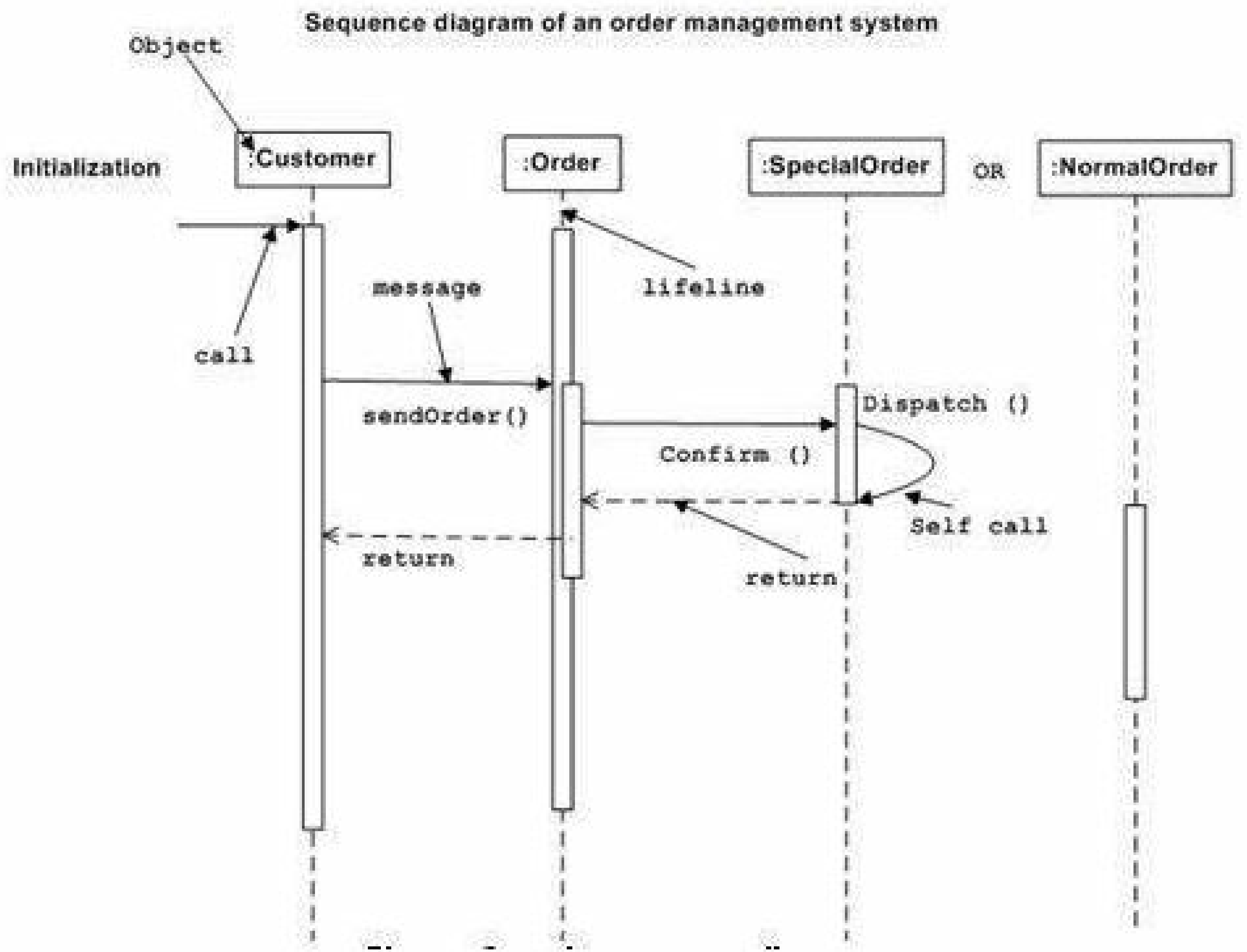
Use Case Diagrams - Example



Interaction Diagrams - Interactions among the different elements in the model

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

Sequence Diagram- Emphasizes on time sequence of messages



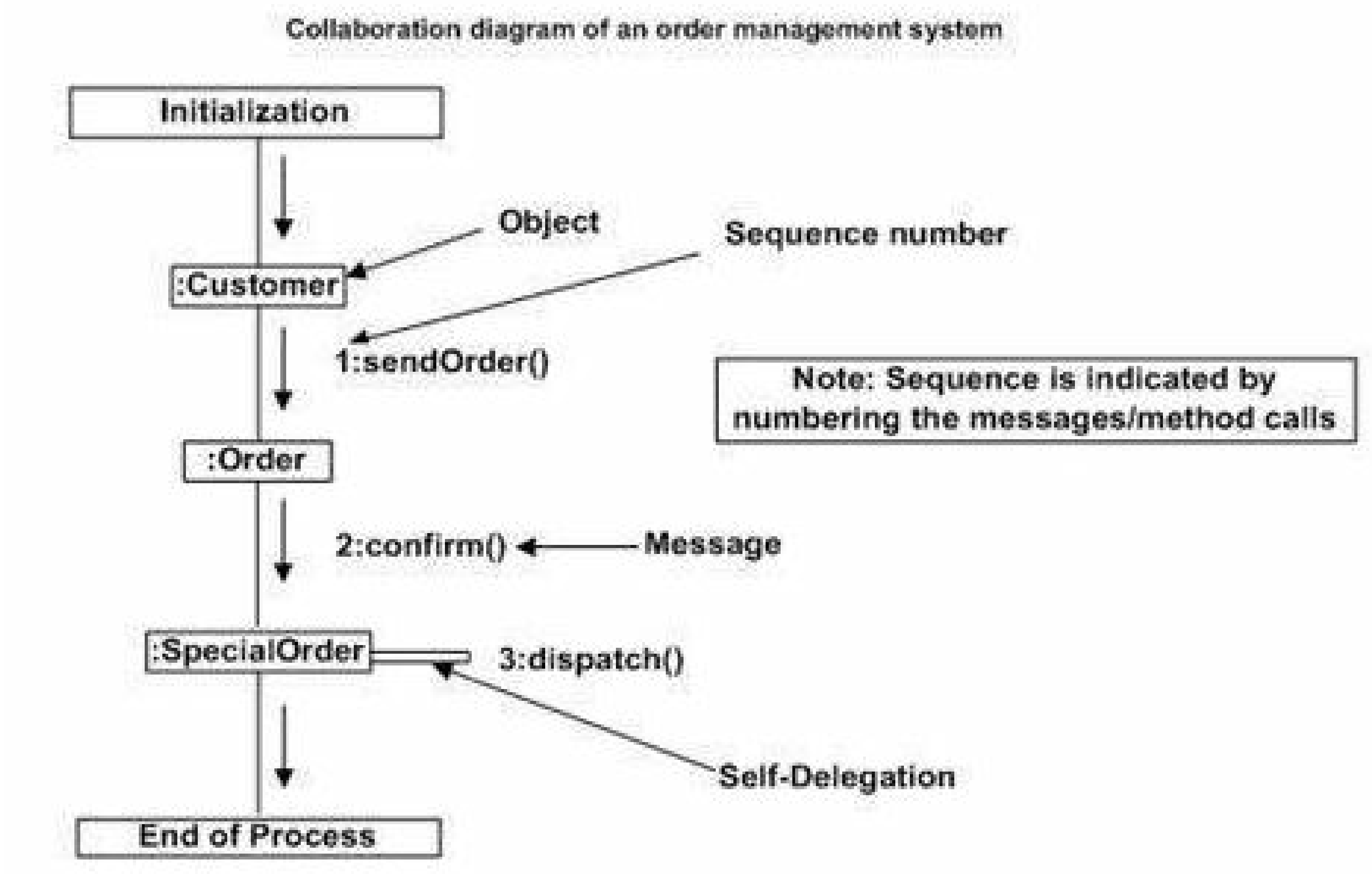
Explanation

- The sequence diagram has four objects (Customer, Order, SpecialOrder and NormalOrder).
- The diagram shows the message sequence for *SpecialOrder* object and the same can be used in case of *NormalOrder* object.
- It is important to understand the time sequence of message flows.
- The message flow is nothing but a method call of an object.
- The first call is *sendOrder ()* which is a method of *Order object*.
- The next call is *confirm ()* which is a method of *SpecialOrder* object and the last call is *Dispatch ()* which is a method of *SpecialOrder* object.
- The following diagram mainly describes the method calls from one object to another, and this is also the actual scenario when the system is running.

Collaboration diagram

- 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.

Collaboration diagram - Emphasizes on the structural organization of the objects that send and receive messages.

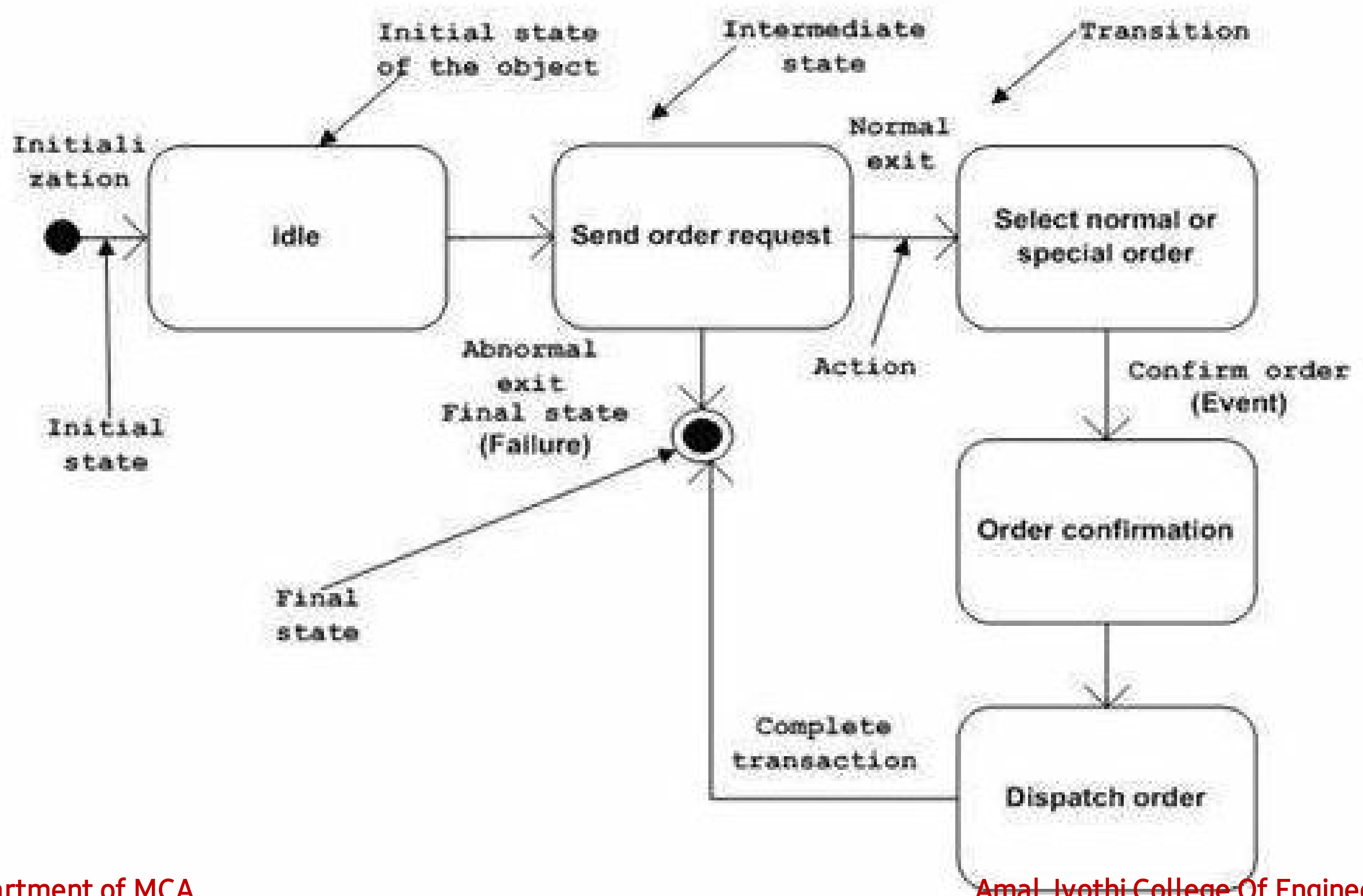


Statechart Diagrams

- A Statechart diagram describes a state machine.
- State machine can be defined as a machine which defines different states of an object and these states are controlled by external or internal events.

Statechart Diagrams

Statechart diagram of an order management system

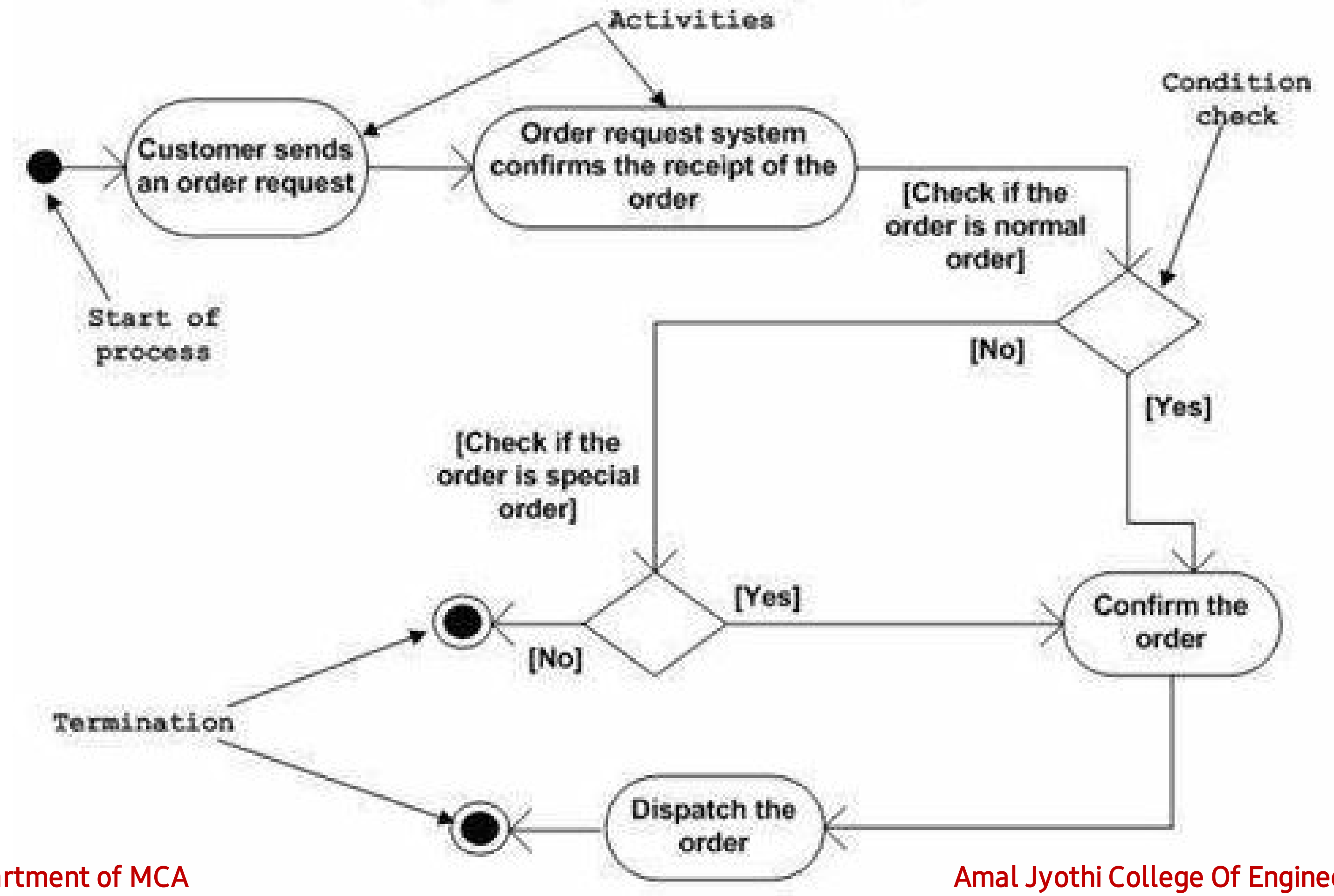


Activity Diagrams

- Activity diagram is basically a flowchart to represent the flow from one activity to another activity.
- The control flow is drawn from one operation to another

Activity Diagrams – Basically a flowchart to represent the flow from one activity to another activity

Activity diagram of an order management system



Activities
Condition check
Customer sends an order request
Order request system confirms the receipt of the order
[Check if the order is normal order]
Start of process
[No]
[Yes]
[Check if the order is special order]
[Yes]
[No]
Confirm the order
Termination
Dispatch the order

Thank You!