UML Introduction
Software Development

CSc 3350

Dr. William Greg Johnson
Department of Computer Science
Georgia State University

UML in Software Development

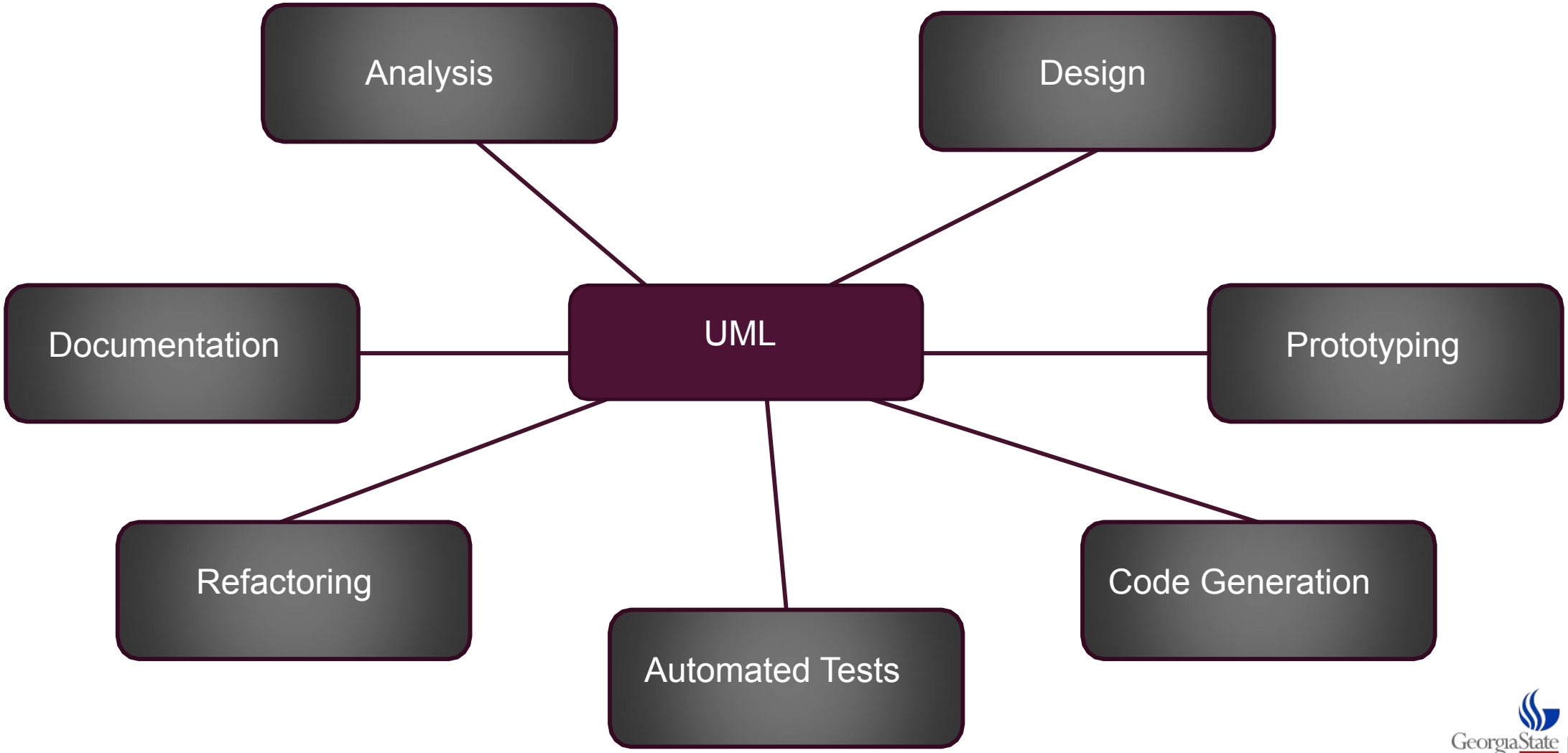
- UML = Unified Modeling Language
- A standard language for specifying, visualizing, constructing, and documenting a system in which software represents the most significant part
- Different from the other common programming languages like Python, C++, Java, SQL
- Pictorial language used to make software ‘blueprints’
- Can serve as a central notation for software development process
- Using UML helps project teams communicate, explore potential designs, and validate the architectural designs of software
- Diagrams are made using notation of things and relationships

Today's UML…

"A good 70% of UML was a useless farce to sell overpriced, clunky tools. Don't learn UML to go around annoying people with useless class diagrams. Do learn the basics so you can read a sequence diagram and learn to think this way."

Andrew Oliver
InfoWorld
"7 Books you must read to be a real software developer"
April 19, 2018

UML is Centric to Software Development



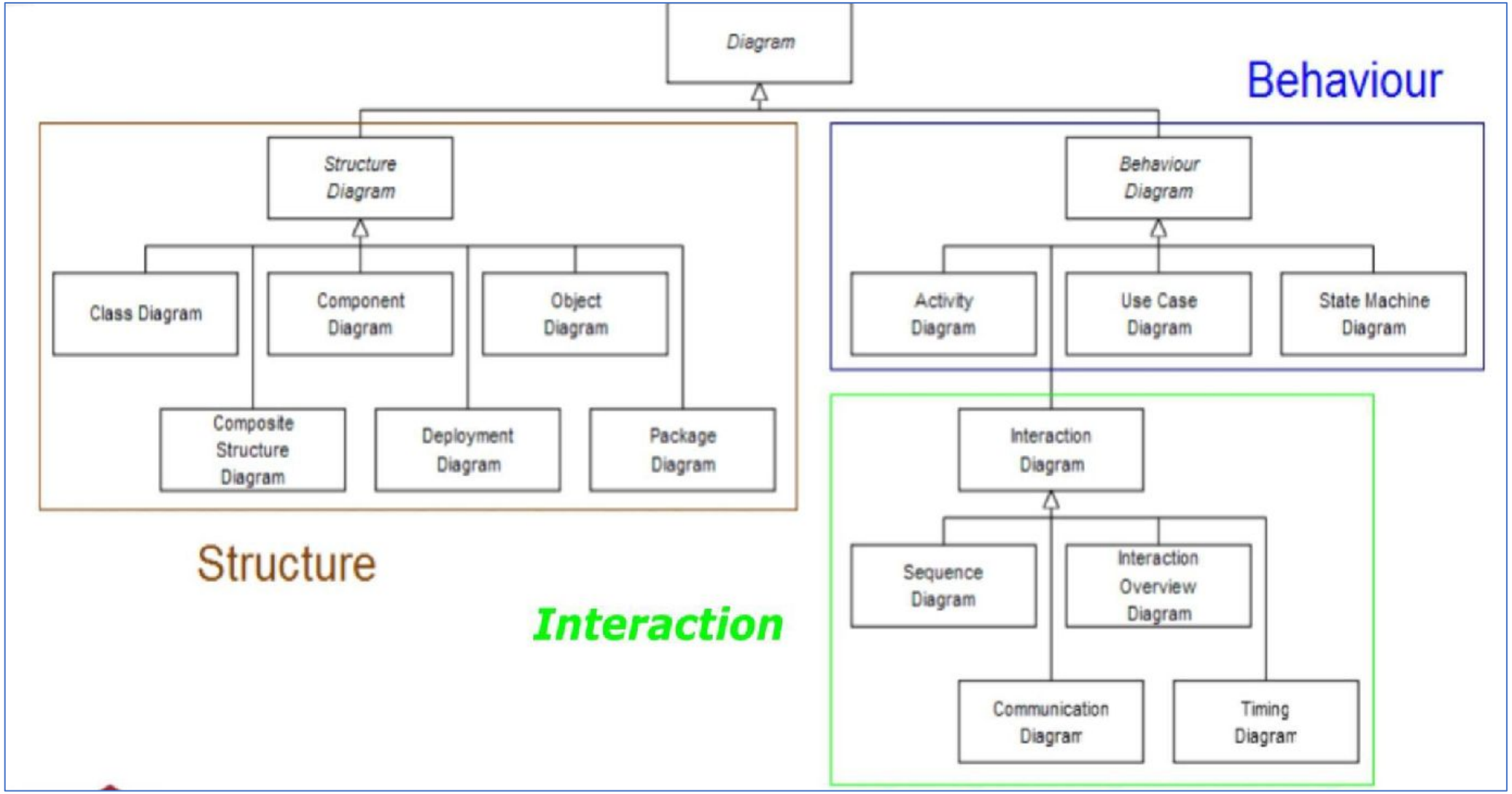
Conceptual Model

- *Conceptual model*: can be defined as a model which is made of logical relationships between software components, structures, and behaviors.
- As UML describes the real time systems it is very important to make a conceptual model and then proceed gradually.
- Conceptual models can be better understood by learning the following three major elements :
  - UML building blocks
  - Rules to connect the building blocks
  - Common mechanisms of UML

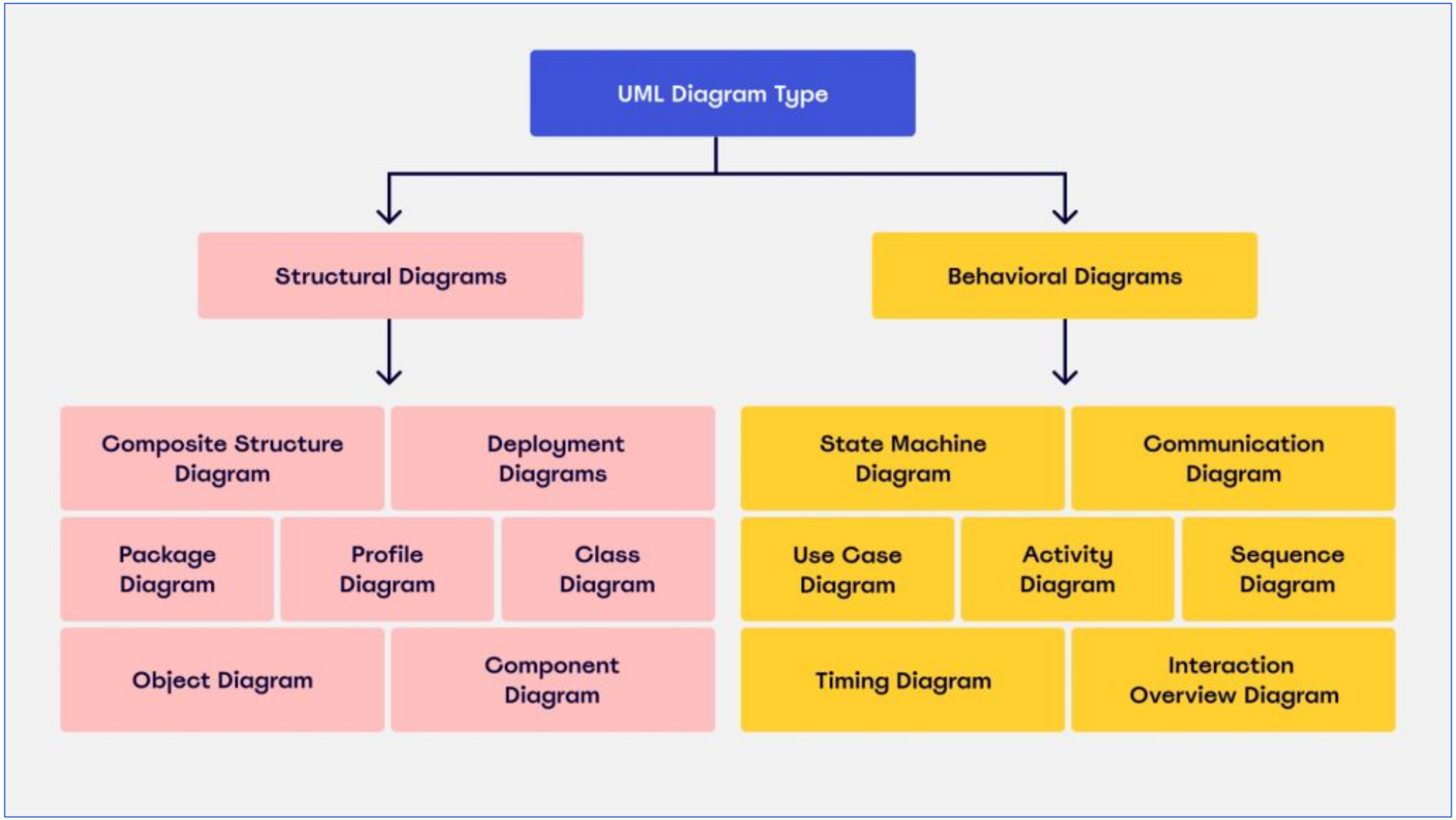
Building Blocks of UML

- The building blocks of UML can be defined as:
  - Things
  - Relationships
  - Diagrams
- These building blocks of UML and can be a category:
  - Structural
  - Behavioral
  - Interaction

Historical UML Diagram Categories



Modern UML Diagram Categories



Structural

These define the static part of the model. They represent physical and conceptual elements.

Descriptions

- Class: represents set of objects having similar responsibilities
- Interface: defines a set of operations which specify the responsibility of a class.
- Collaboration: defines interaction between elements
- Use case: represents a set of actions performed by a system for a specific goal
- Component: describes physical part of a system
- Node: can be defined as a physical element that exists at run time.

Behavioral

These consist of the dynamic parts of UML models.
Examples:

- Interaction: is defined as a behavior that consists of a group of messages exchanged among elements to accomplish a specific task
- State machine: useful when the state of an object in its life cycle is important. It defines the sequence of changes an object goes through in response to events. Events are external factors responsible for state change

Relationships

These show how elements are associated and describes the functionality of a software system.
There are four types:

1. Dependency: is a relationship between two things in which change in one element also affects the other one.
2. Association: is a set of links that connects elements of an UML model. It also describes how many objects are taking part in that relationship.
3. Generalization: can be defined as a relationship which connects a specialized element with a generalized element. It describes inheritance relationship in the world of objects.
4. Realization: can be defined as a relationship in which two elements are connected. One element describes some responsibility which is not implemented and the other one implements them. This relationship exists in case of interfaces.

UML Diagrams

These nine diagrams in UML can be used to model a system at different points of time in the SDLC:

1. Class diagram
2. Object diagram
3. Use case diagram
4. Sequence diagram
5. Collaboration diagram
6. Activity diagram
7. State diagram
8. Deployment diagram
9. Component diagram

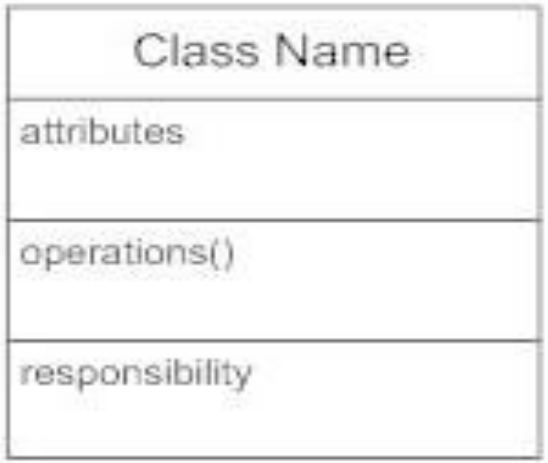
Class Diagrams

Class diagrams model the static structure of a system. They show relationship between classes, objects, attributes, and operations. They have three parts; name at the top, attributes in the middle and operations/methods at the bottom.

1. The functionalities provided by the class are termed ‘methods’ of the class.
2. Attributes uniquely identify the class.
3. The class diagram is a static view of a part of the application.

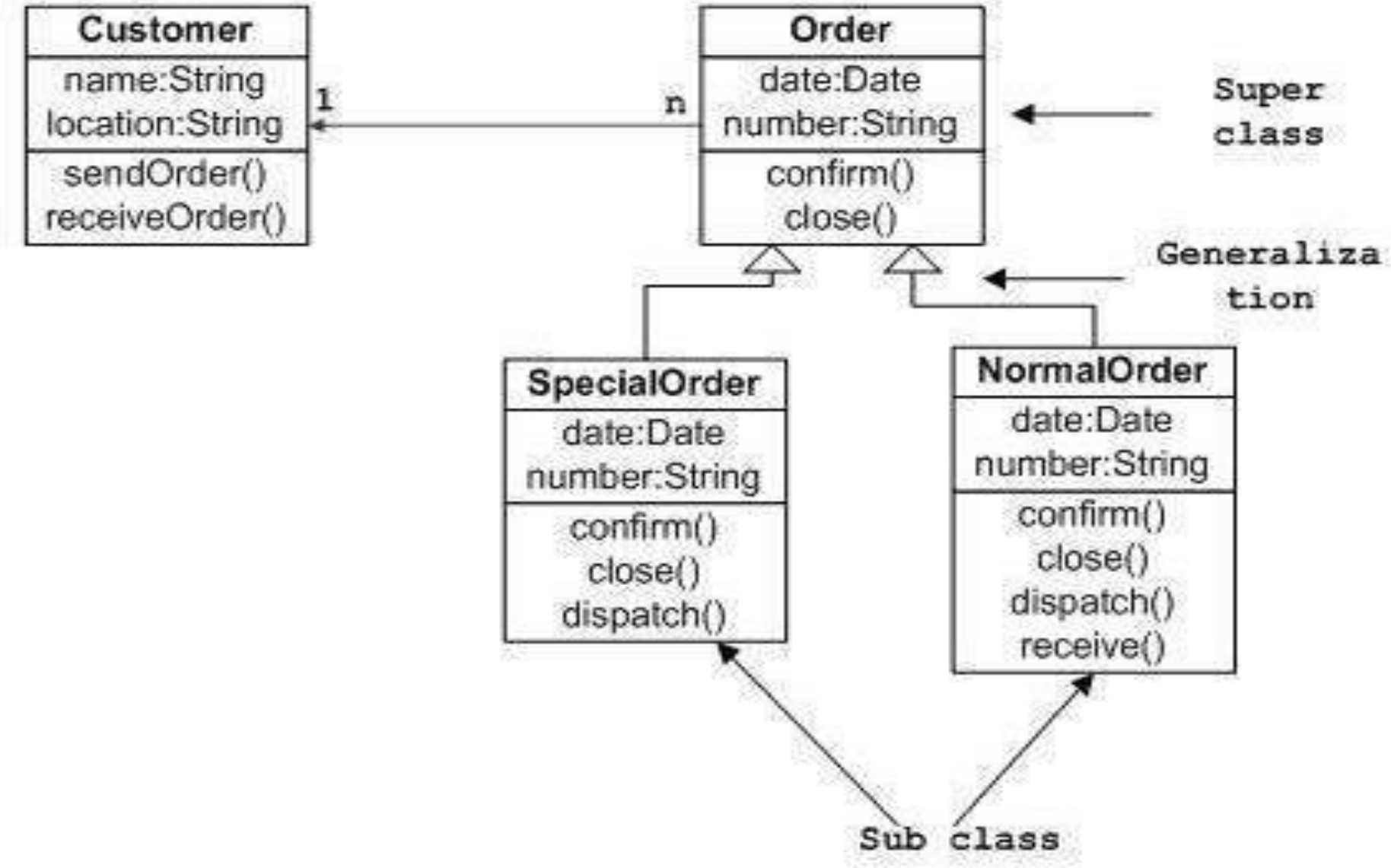
Purpose of a class diagram:

4. Analysis and design
5. Describe responsibilities
6. Basis for deployment and component diagrams



Class

Sample Class Diagram



Drawing a Class Diagram

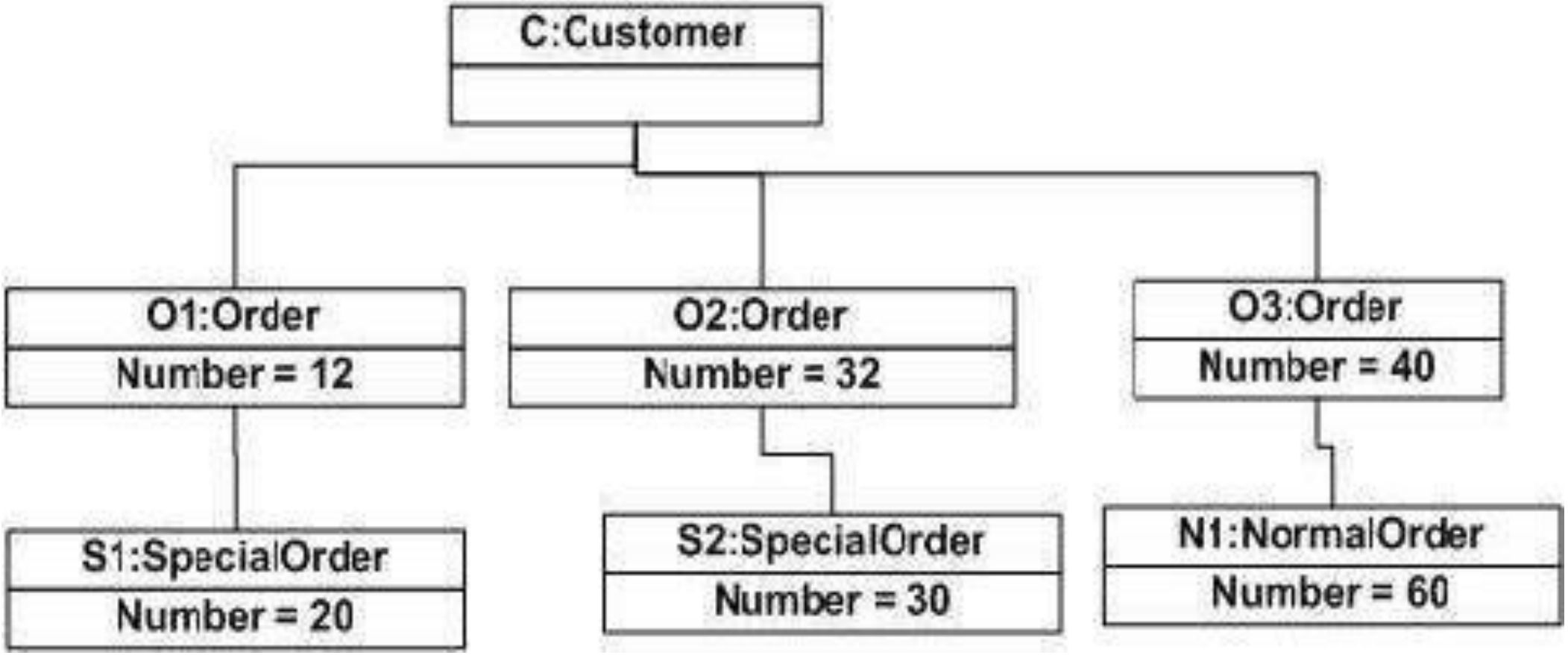
The following points should be remembered :

- The name should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified.
- For each class, a minimum number of properties should be specified because unnecessary properties will make the diagram complicated.
- Use notes when ever required to describe some aspect because at the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, it should be drawn on plain paper and reworked as many times as possible to make it correct

Object Diagram

- Object diagrams describe the static structure of a system at a particular time. They can be used to test class diagrams for accuracy.
- It is a special kind of class diagram. Recall, object is an instance of a class.
- The object diagram captures the state of different classes in the software system and the associations at a given point of time.
- Can be used for:
  - Making the prototype of a system.
  - Reverse engineering.
  - Modelling complex data structures.
  - Understanding the system from practical perspective.

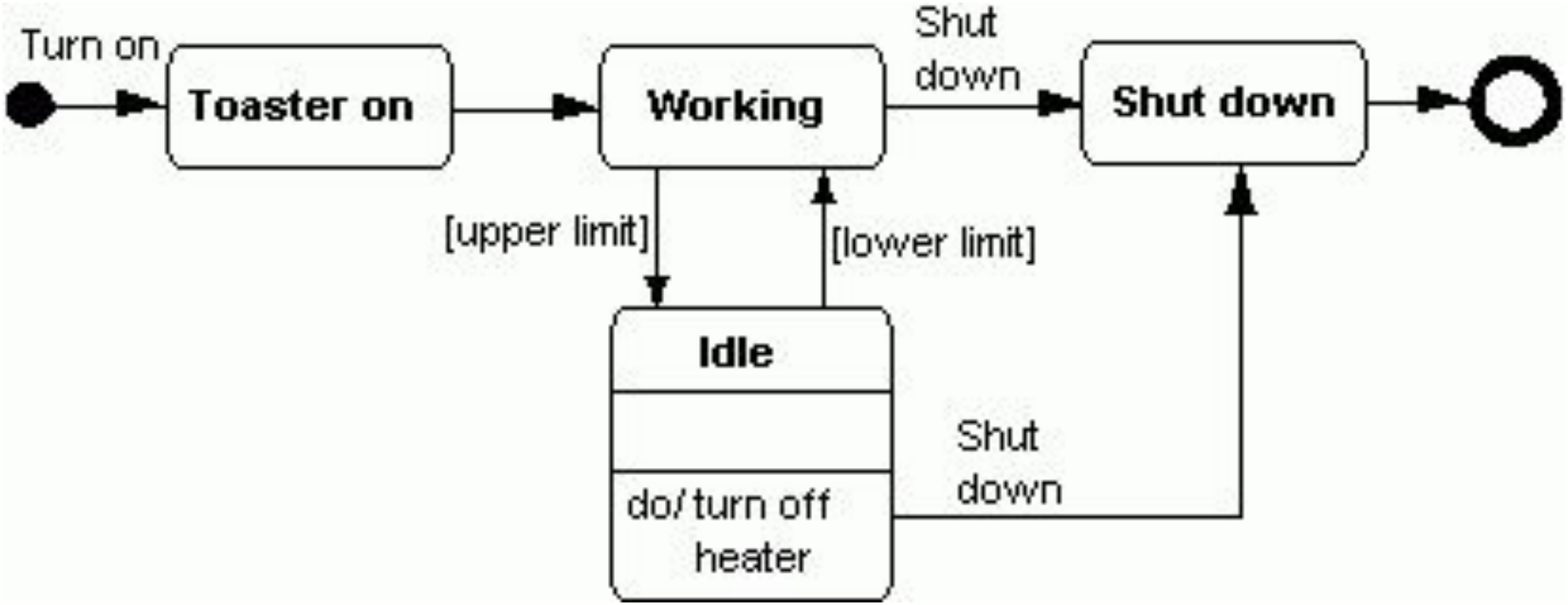
Object Diagram: Order Management System



State Diagram

- As the name suggests, it describes different states of a component in a system. The states are specific to a component/object of a system.
- Objects in the system change status in response to events. Used to model dynamic nature of a system.
- They describe the flow of control from one state to another state. States are defined as a condition in which an object exists and it changes when some event is triggered.
- Main purposes:
  - To model dynamic aspect of a system.
  - To model lifetime of a reactive system.
  - To describe different states of an object during its lifetime.
  - Define a state machine to model states of an object.

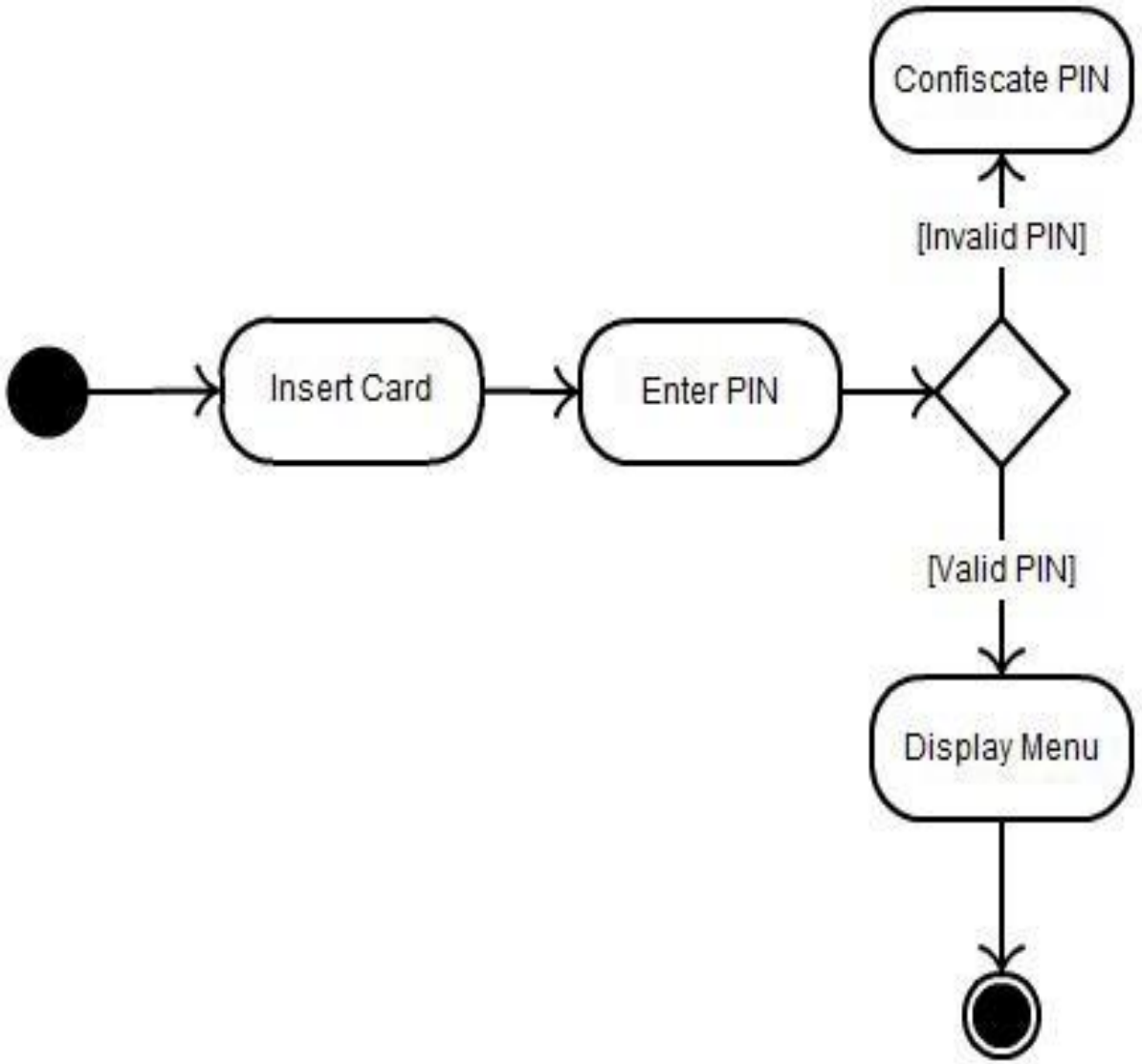
State Diagram for a toaster



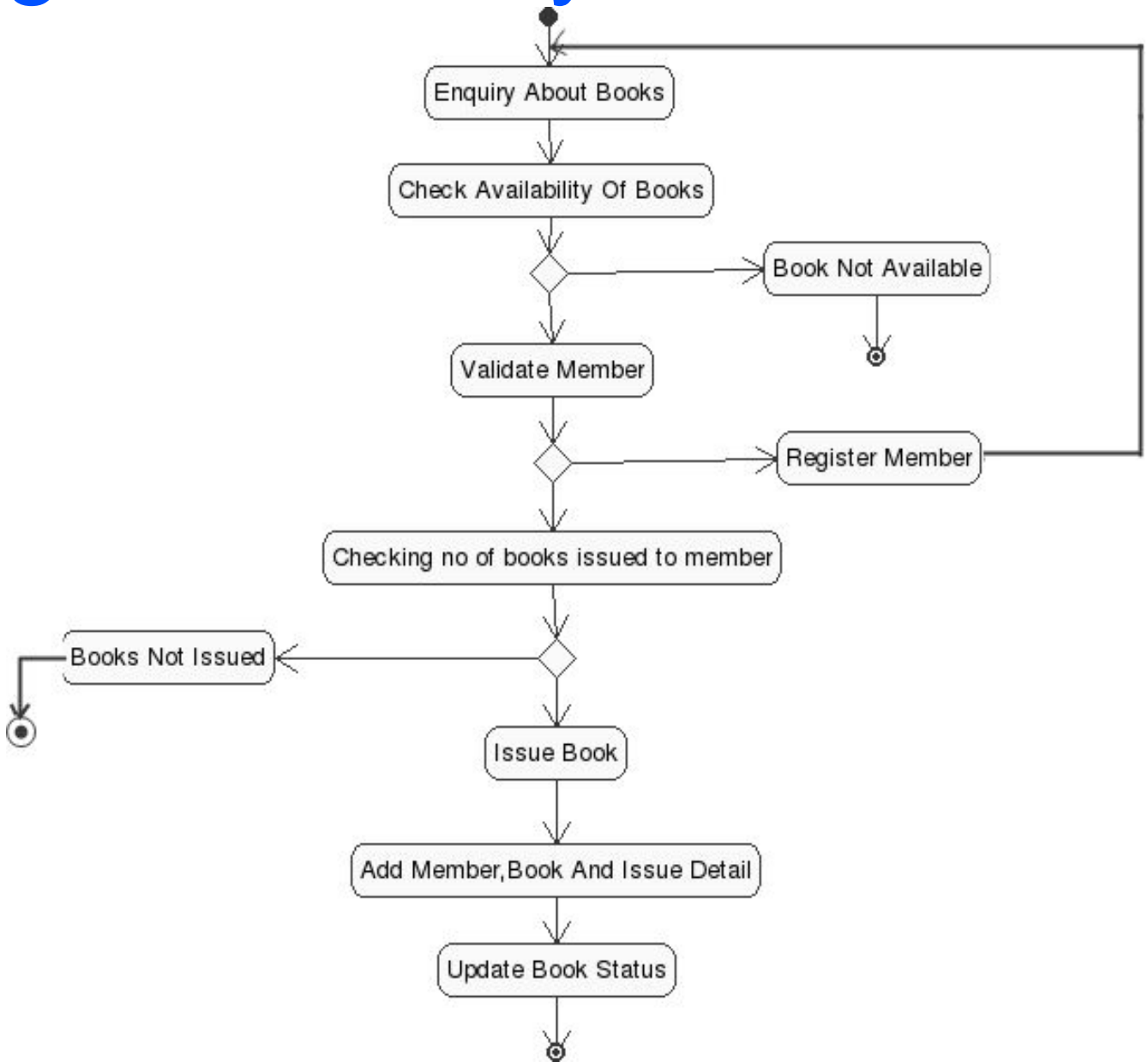
Activity Diagram

- Activity diagram is a flow chart to represent the flow form one activity to another activity. The activity can be described as an operation of the system.
- The control flow is drawn from one operation to another. This flow can be sequential, branched or concurrent. Activity diagrams deal with all types of flow control by using different elements like fork, join etc.
- Purposes can be described as:
  - Draw the activity flow of a system.
  - Describe the sequence from one activity to another.
  - Describe the parallel, branched and concurrent flow of the system.
- Before drawing an activity diagram, should identify the following elements:
  - Activities, Conditions, Associations, Constraints

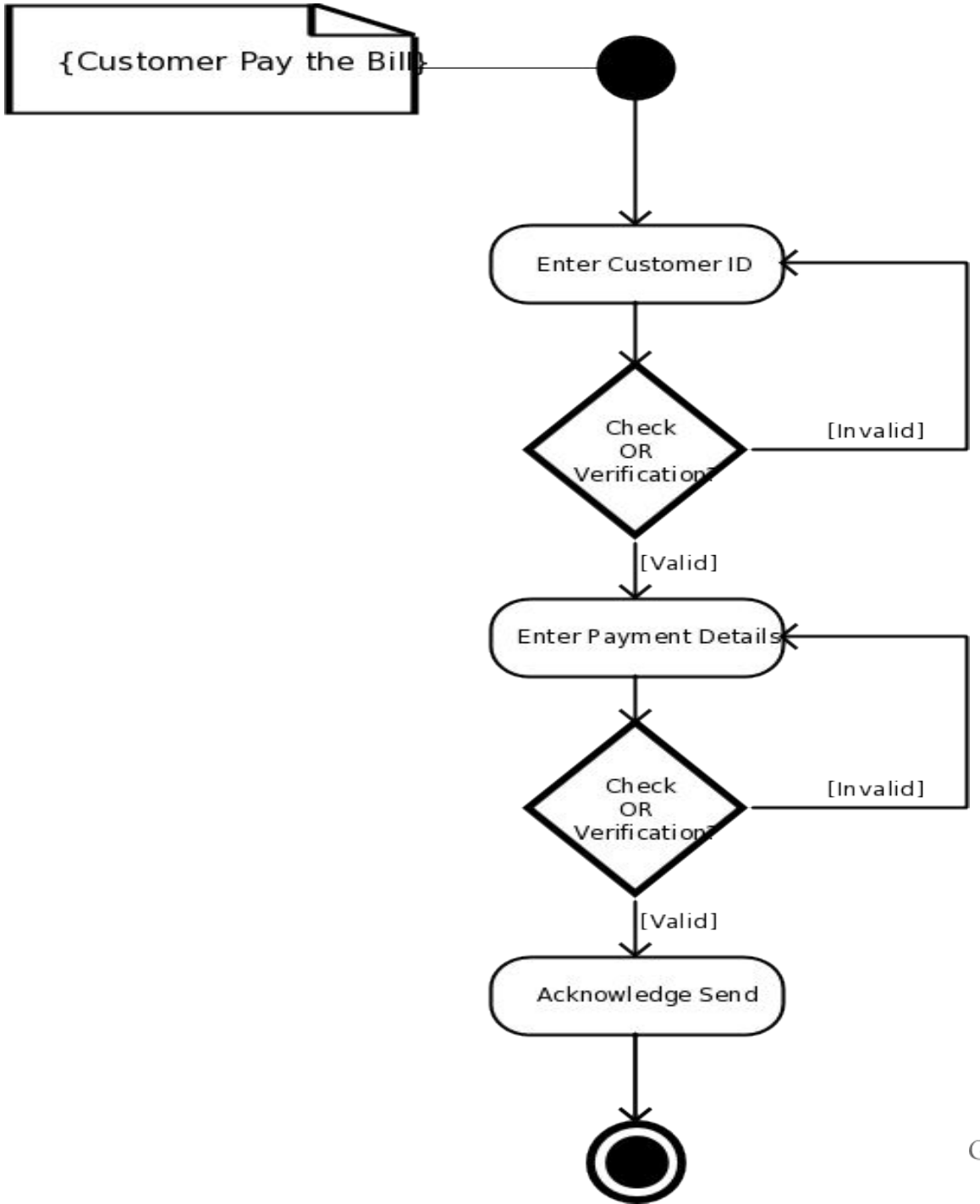
Activity Diagram: ATM



Activity Diagram: Library Checkout



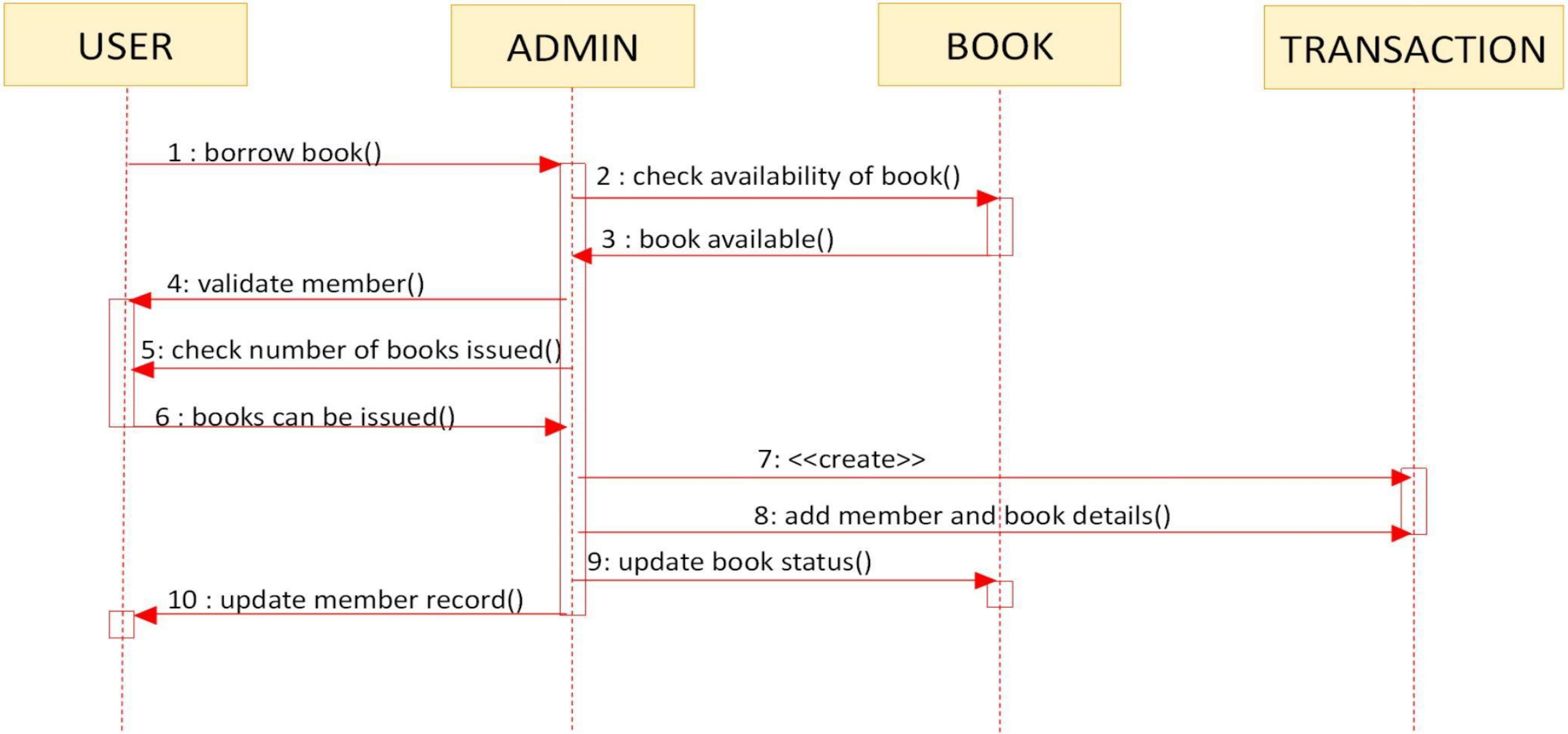
Activity Diagram: Pay Electric Bill



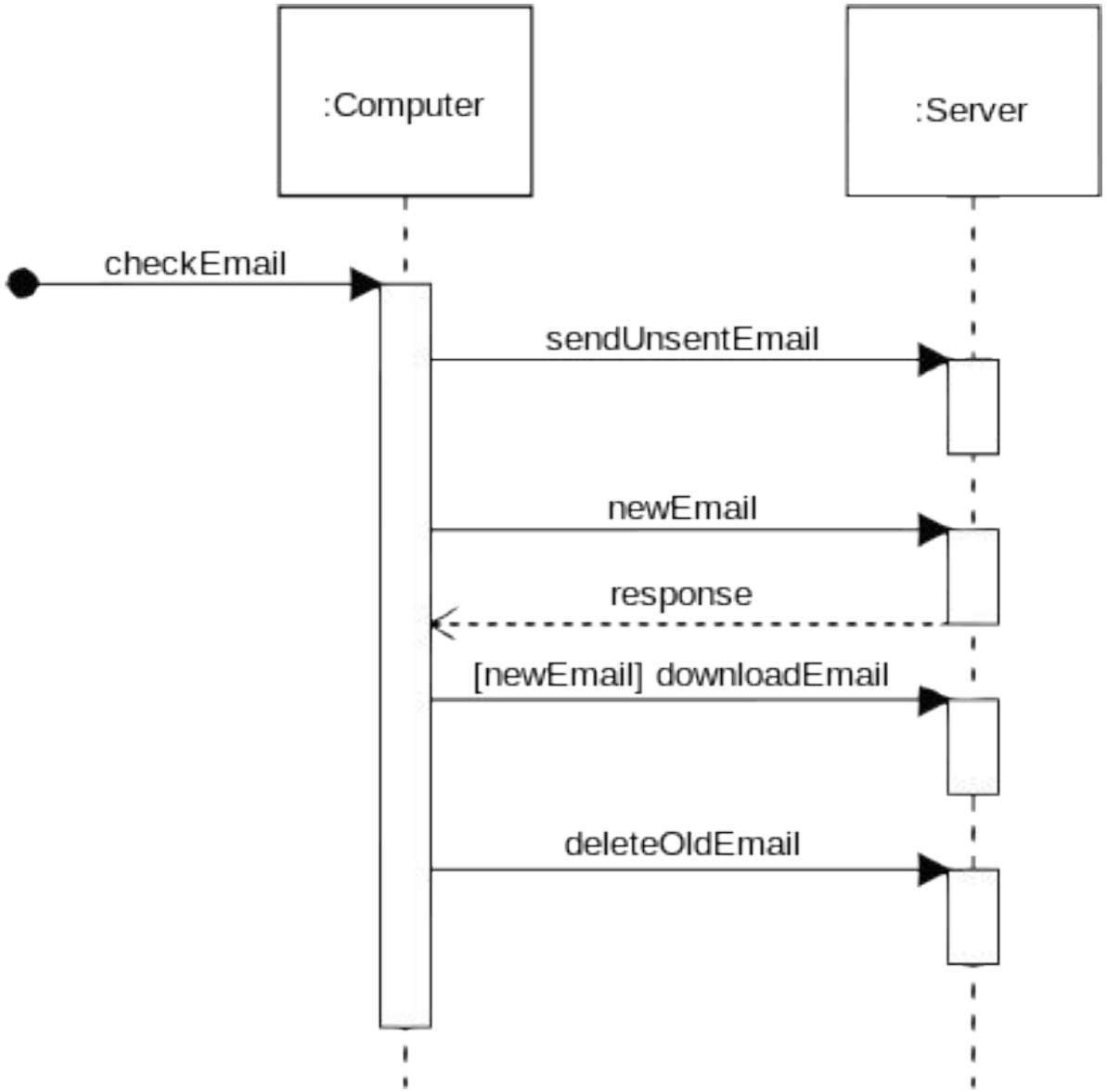
Sequence Diagram

- Used to describe some interactions among different objects/actors in the software system.
- Interactions are time-ordered.
- Purposes can be described as:
  - Capture dynamic behavior of a system.
  - Describe the message flow of a system.
  - Describe structural organization and interactions of objects
- Before drawing, identify the following elements:
  - Objects/actors taking part in the interaction
  - Message flow among these objects/actors
  - Sequence of messages flowing between
  - Object/actor organization

Sequence Diagram: Borrow Library Book



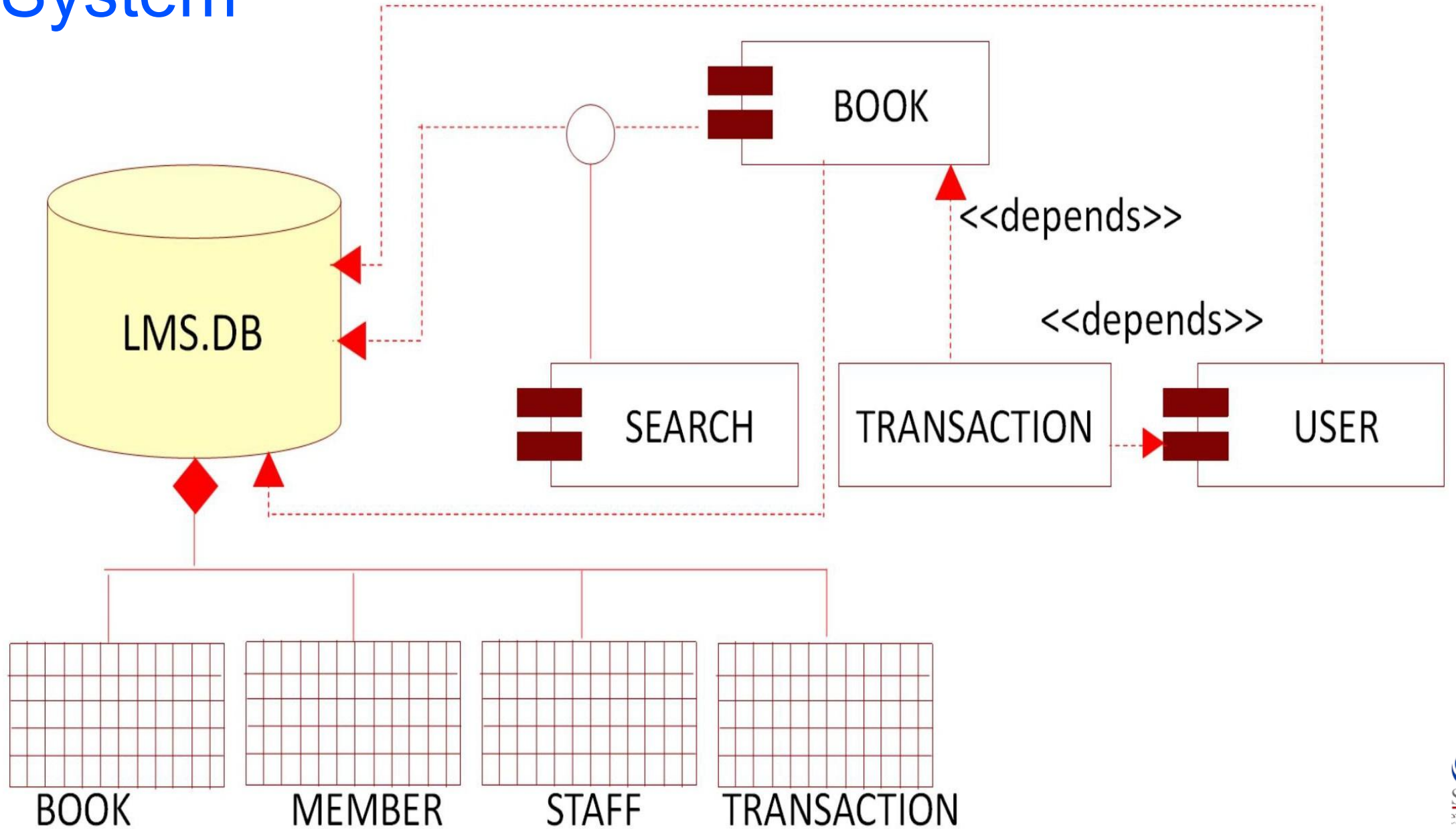
Sequence Diagram: Check Email



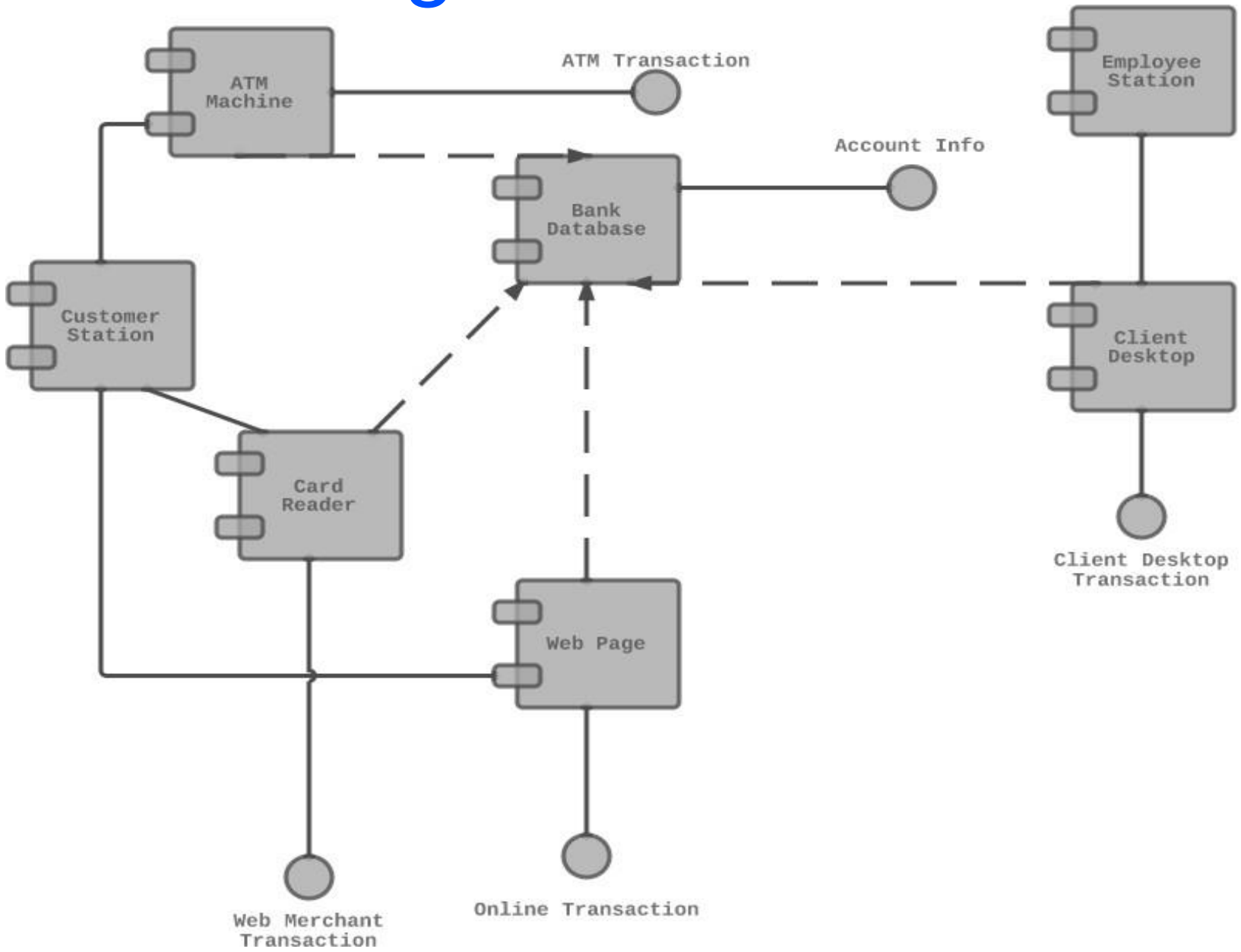
Component Diagram

- Difficult in terms of nature and behavior because they are used to model physical aspects of a system.
- Physical aspects are executables, libraries, configuration files, documents that are held in a node (run-time).
- Used to visualize the organization and relationships among components in a software system.
- Purposes can be described as:
  - Visualize components of a system
  - Construct executables by using forward and reverse engineering
  - Show the organization and relationship with components
- Before drawing, identify the following elements:
  - Files used by the system
  - Libraries and other API relevant to the system
  - Relationships among the APIs and libraries used

Component Diagram: Library Management System



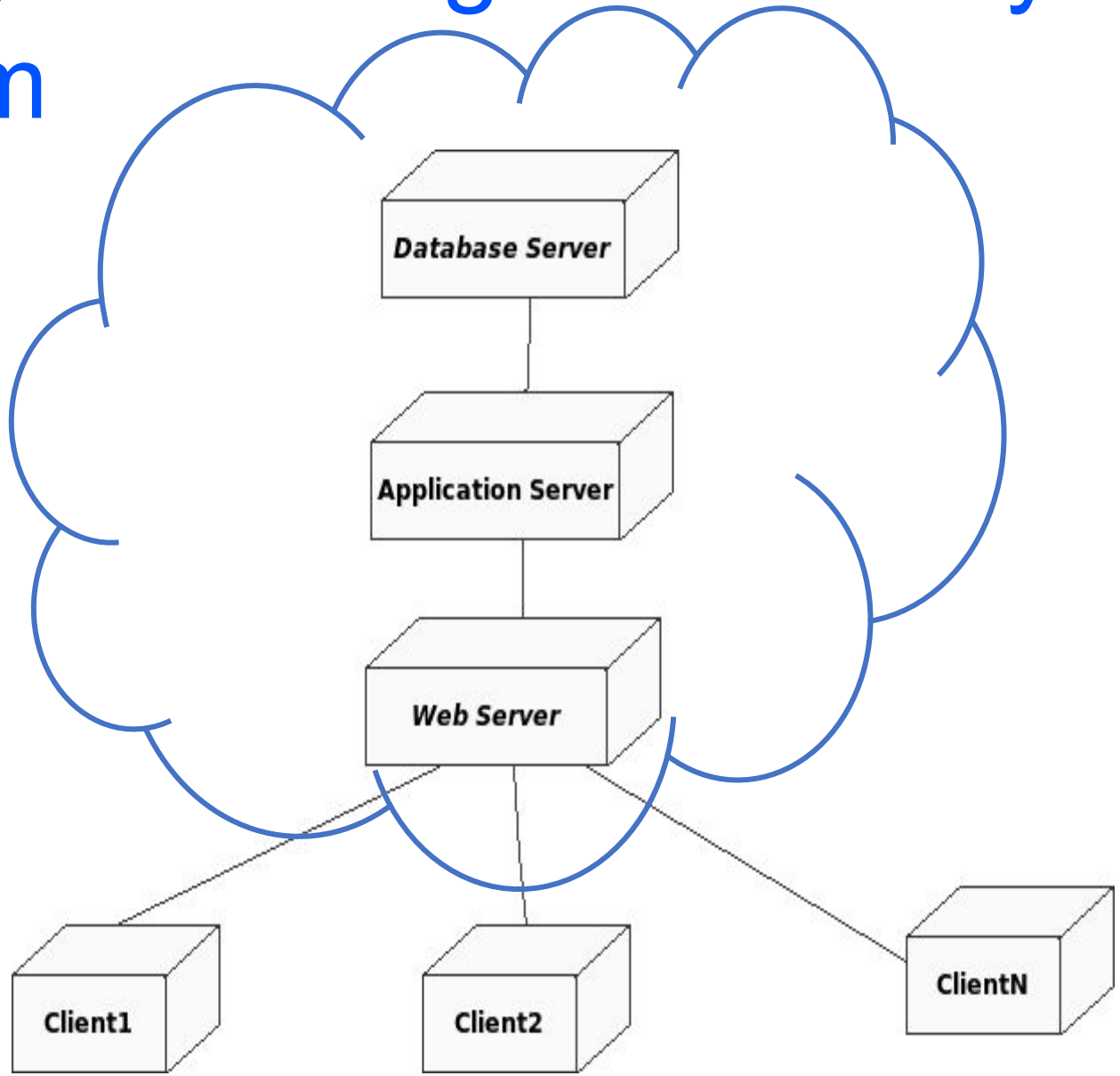
Component Diagram For ATM



Deployment Diagram

- Used to visualize the topology of the physical components of a system and where the software components are deployed. Deployment diagrams consist of nodes and their relationships.
- Purposes can be described as:
  - Visualize cloud/hosted topology of a system.
  - Describe the components used to deploy software components.
  - Describe runtime processing nodes.
- Before drawing, identify the following:
  - Nodes and relationship among themselves
- Useful for system engineers it shows control of these:
  - Performance
  - Scalability
  - Maintainability
  - Portability

Deployment Diagram: Library Management System



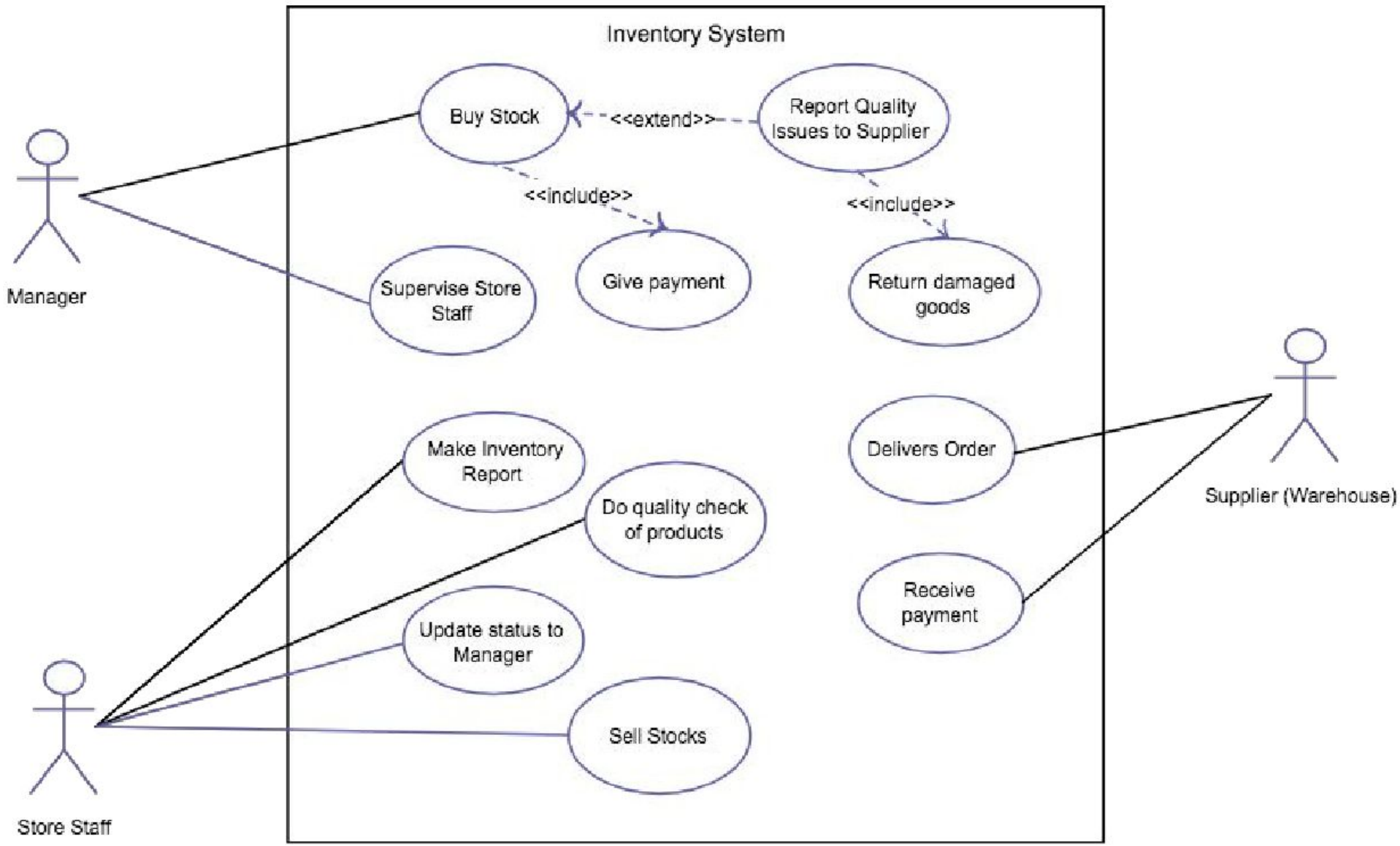
Use Case Diagram

- Use case diagrams consists of actors, components, and their relationships. The diagram is used to model a system/subsystem of an application. A single use case diagram captures a particular functionality of a system.
- Purposes can be described as:
  - Gather requirements of a system
  - Get an outside view of a system.
  - Identify external and internal factors influencing the system
  - Show interactions among the requirements of actors.

Use Case Diagram

- Guidelines for drawing:
  - The name of a use case is very important and should be chosen in such a way so that it can identify the functionalities performed
  - Give a suitable name for actors (Human or component)
  - Show relationships and dependencies clearly
  - Do not try to include all types of relationships
  - The main purpose is to identify requirements and use notations to clarify some important points
- Places where use case diagrams are used:
  - Requirement analysis and high-level design
  - Model the context of a system
  - Reverse engineering
  - Forward engineering

Use Case Diagram: Inventory System



Attributions

1. Ashita Agrawal, https://www.slideshare.net/ASHadventurelover/introduction-to-uml-diagrams
2. Nwabueze Obioma, https://www.slideshare.net/Zed4rReal/uml-and-software-modeling-toolspptx
3. Brad Beiermann, https://www.slideshare.net/BradBeiermann/the-language-of-application-architecture
4. http://uml.org Maintained by the Object Management Group, Inc.

End.

UML Introduction
Software Development

CSc 3350

Dr. William Greg Johnson
Department of Computer Science
Georgia State University