

UML

Unified Modeling Language (UML) is a general purpose modeling language designed to provide a standard way to visualize the design of a system.

UML as a Sketch

Most common use of UML is to use it to:

- help communicate some aspect of a system and to better understand it in a graphical view.
- Used for both forward engineering and reverse engineering.
- Strives to be informal and dynamic.
- Only emphasizes those **classes**, **attributes**, **operations**, and **relationships** that are of interest.
- Most concerned with selective communication than complete specification.

UML as a Blueprint

- Goal is completeness
 - Start the diagram and add as many details as you can.
 - Definitive details.
- It is more definitive, while the sketch approach is more explorative.
- Used to describe a detailed design for a programmer to follow in writing source code.
- Notation should be sufficiently complete so that a software engineer can follow it.
- It can be used by a designer to develop blueprint-level models that show interfaces of subsystems or classes.
- As a reverse engineered product diagrams convey detailed information about the source code that is easier for software engineers to understand.

UML as a Programming Language

- Specifies the complete system in UML so that code can be automatically generated.
 - Help create initial code from it. Gives a headstart.
 - Gives a headstart that lets developers jump from the boilerplate and right into the project.
- Looks at UML from a software perspective rather than a conceptual perspective

- Diagrams are compiled directly into executable code so that the UML becomes the source code.
- Challenge is making it more productive to use UML, rather than some another programming language.
- Another concern is how to model behavioral logic.
 - *Done with interaction diagrams, state diagrams, and activity diagrams.*

Ways of Using Comparison

UML sketches are useful with both forward and reverse engineering in both conceptual and software perspectives.

- Conveys **what and how to build**.
- What classes, attributes, operations and relationships are needed.
Detailed forward engineering blueprints are difficult to do well and slow down the development effort.
- Actual implementation of interfaces will reveal the needs for changes.
The value of reversed engineered blueprints depends on the CASE tool
- A dynamic browser would be very helpful a thick document wastes time and resources.
UML as a programming language will probably never see significant usage
- Graphical forms have not shown to be store productive code for most programming tasks.

Types of UML Diagrams

- Activity - models procedural and parallel behavior.
- Class - models classes, attributes, operations and relationships.
- Communication - models interaction between objects.
- Component - models structure and connection of components.
- Composite structure - models runtime decomposition of a class.
- Deployment - models deployment of artifacts to nodes.
- Interactive overview - Mixes the sequences and activity diagram.
- Object - module example configuration of instances.
- Package - models compile-time hierarchical structure.
- Sequence - models sequence interaction between objects.
- State Machine - models how events change an object over its life.
- Timing - models timing interaction between objects.

- Use Case - models how users interact with a system.

Let's take a closer look

UML Diagrams can be split into two families.

1) *Structure Diagrams*

2) *Behavioral Diagrams*

💡 Structural diagrams are about how you are implementing and creating objects.

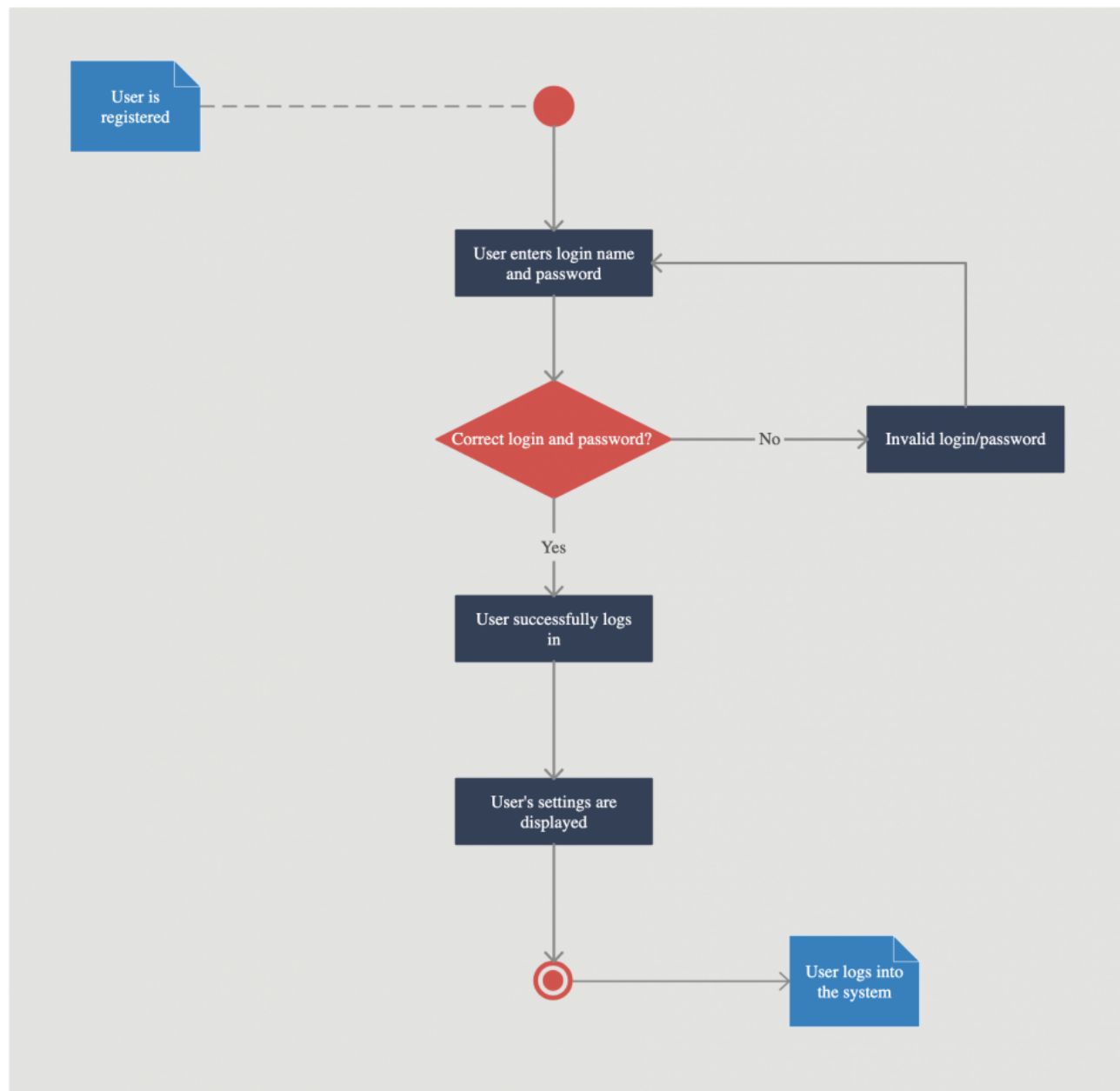
💡 Behavioral diagrams are about how different objects/users will interact and how the system will *behave*.

Activity UML

Activity Diagrams are behavioral diagrams that represent workflows in a graphical way. For example, an activity UML can describe a business workflow. Here's an example below:

A user login system for an app can be represented as:

USER LOGIN SYSTEM for XYZ App



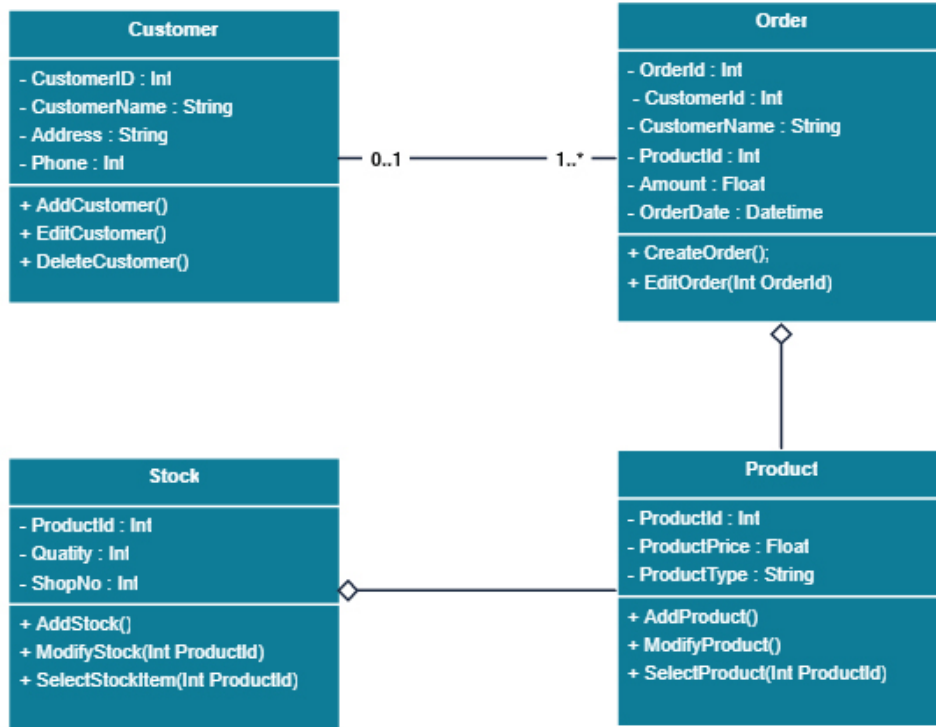
Class UML

Class UMLs are structure diagrams that are key to OOP solutions. It shows the classes in a program, their attributes and operations, and the relationship between classes.

- There's three parts to class UMLs: the name at the top, the attributes in the middle, and the methods (operations) at the bottom.
Here's an example of a class UML diagram below:

Class diagram for order processing system:

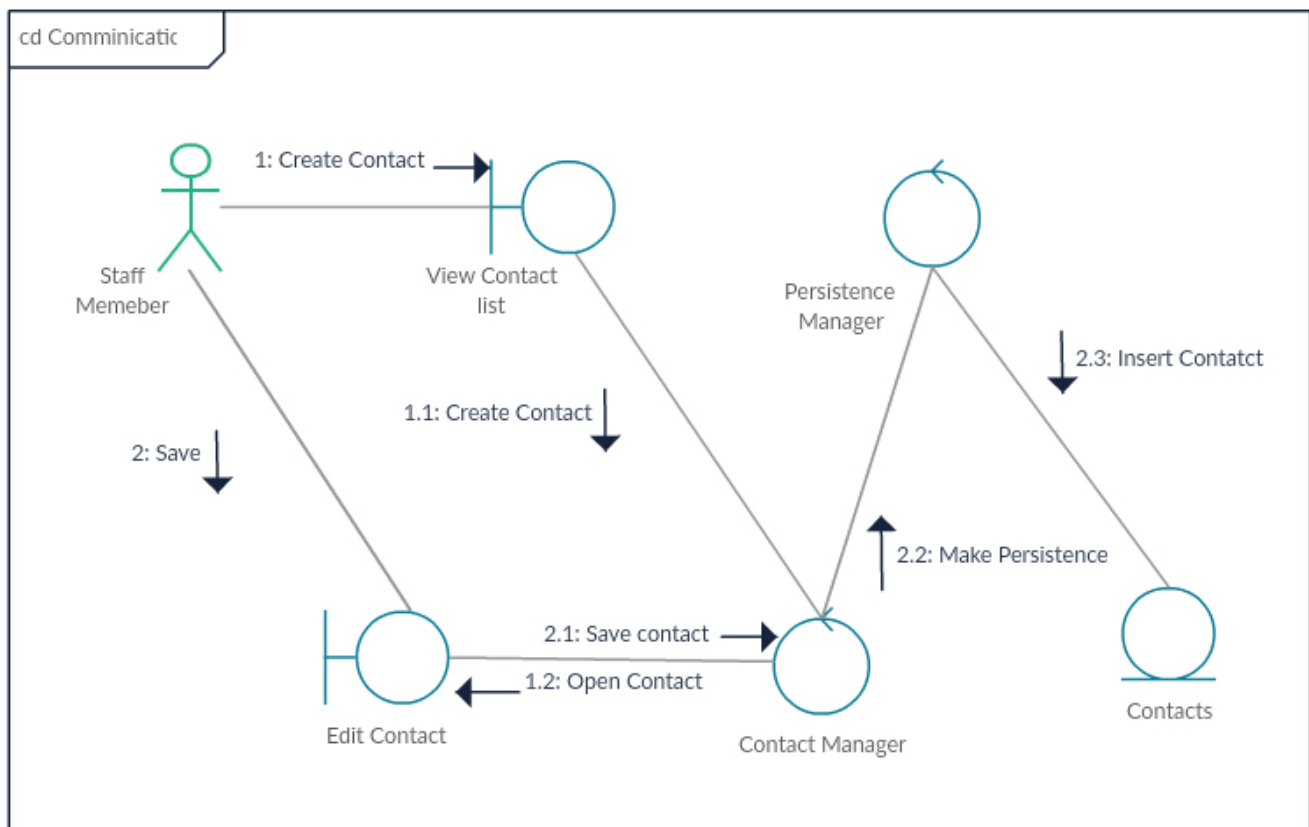
Class Diagram for Order Processing System



Communication UML

Communication UMLs are behavioral diagrams that focus on the communication of messages between objects.

Here's an example below showing just that:



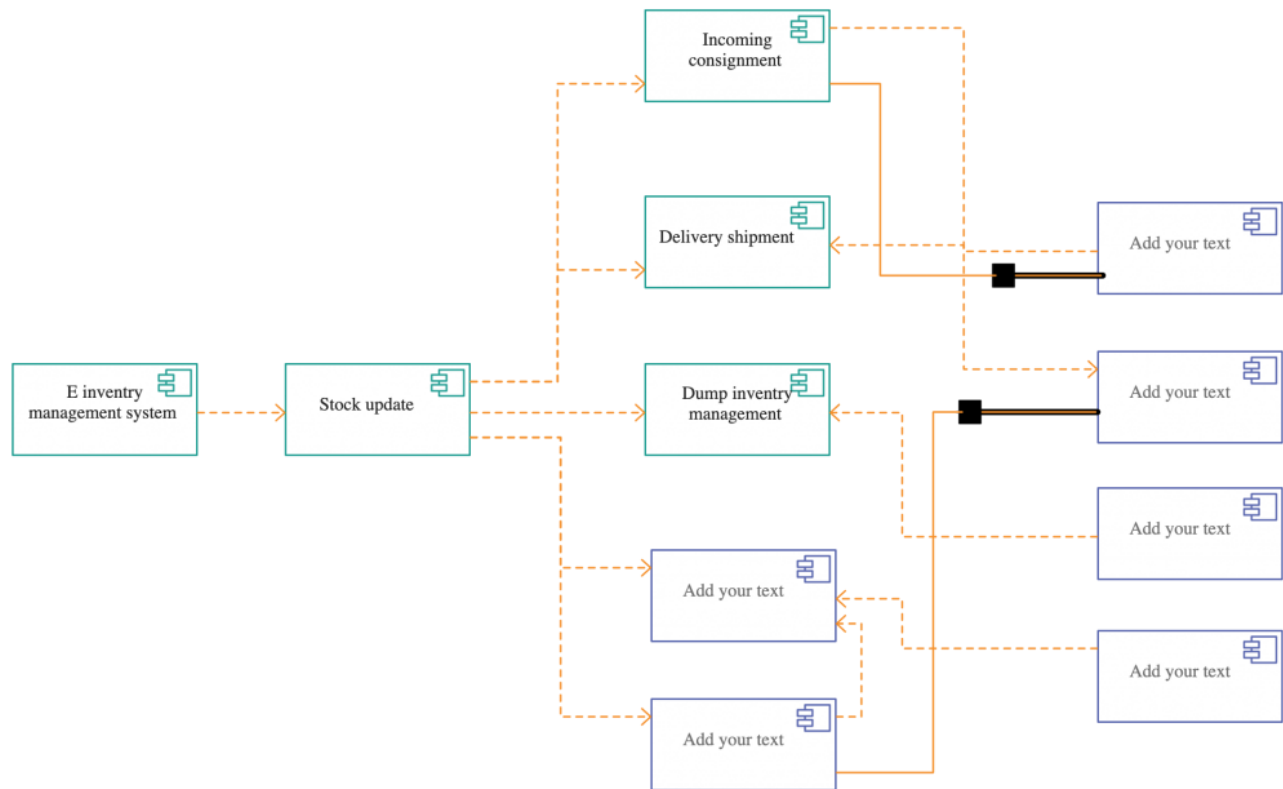
Component UML

A component UML is a structure diagram that displays the structural relationship of components in a software system.

- Mostly used when working on complex systems wiht many components.
- Components communication with each other through **interfaces**. Which are linked using connectors.

Take a look at the Component UML below depicting a Inventory Management System:

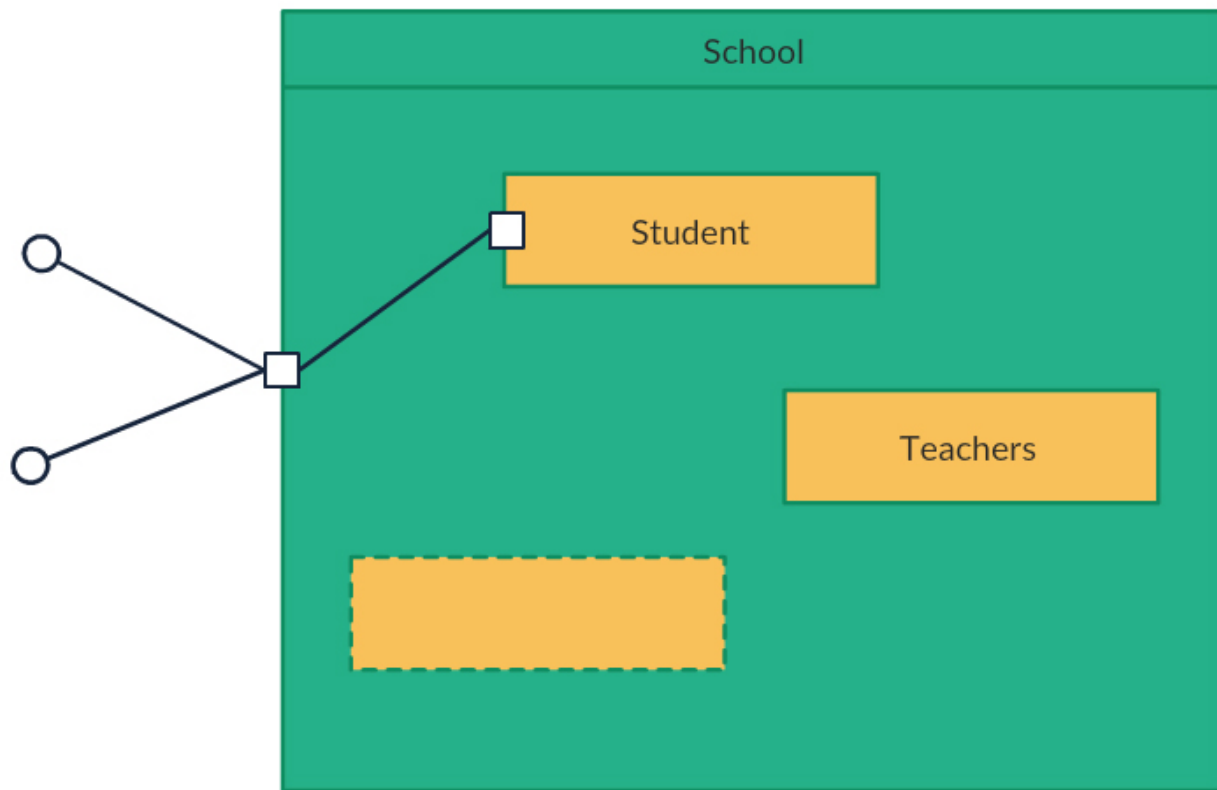
INVENTORY MANAGEMENT SYSTEM



Composite Structure UML

Composite structure diagrams, are structural UMLs used to illustrate the internal structure of a class. For example the Composite Structure UML below of a school *Class*

that has instances of Students and Teachers in it.



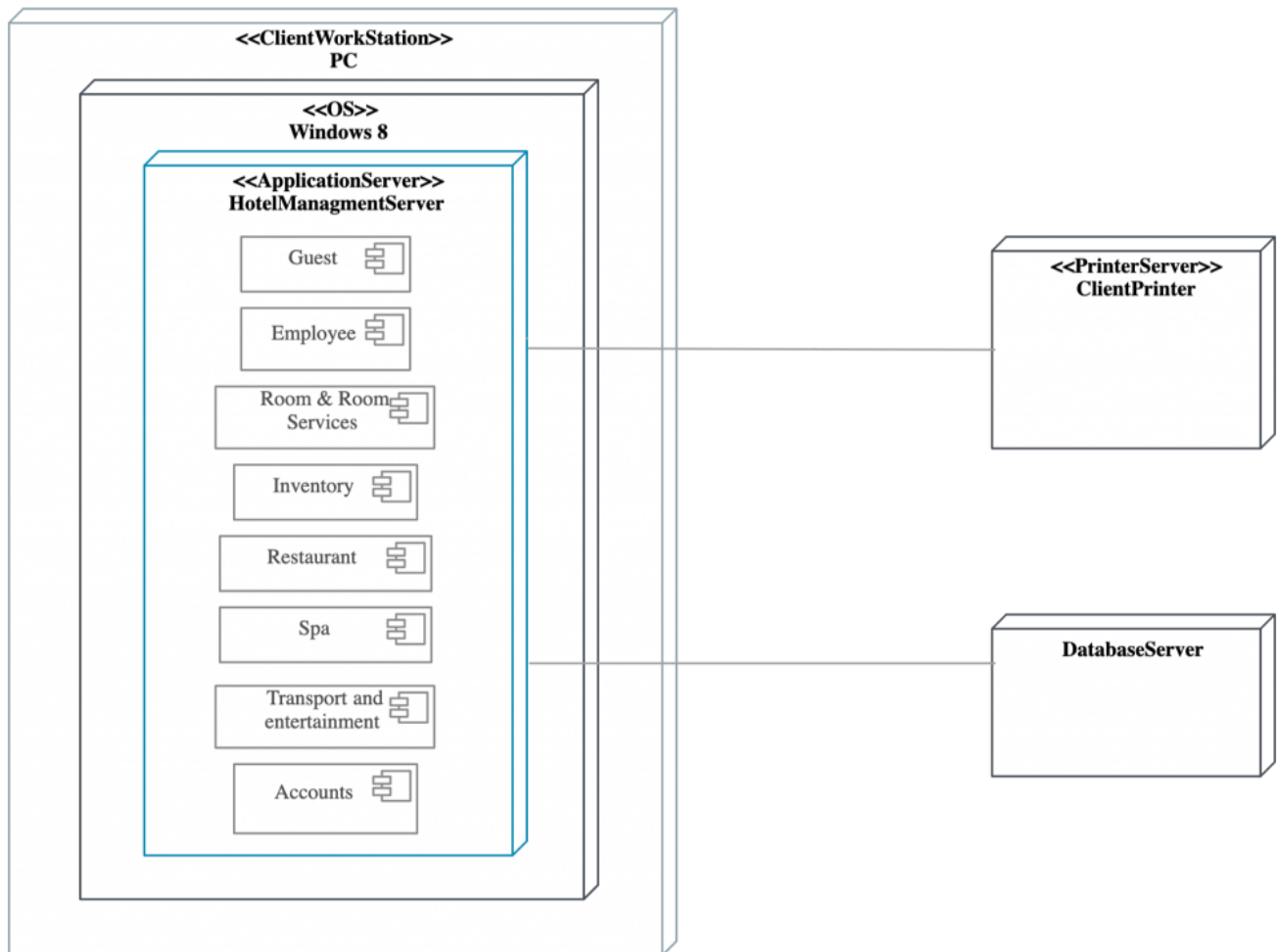
Deployment UML

A deployment UMLs are structural diagrams that show the hardware of your system and the software in that hardware.

- Useful for when software is deployed across multiple machines with unique configs.
For example, check out the Deployment UML of a Hotel Management System below:

Deployment UML of a hotel management system:

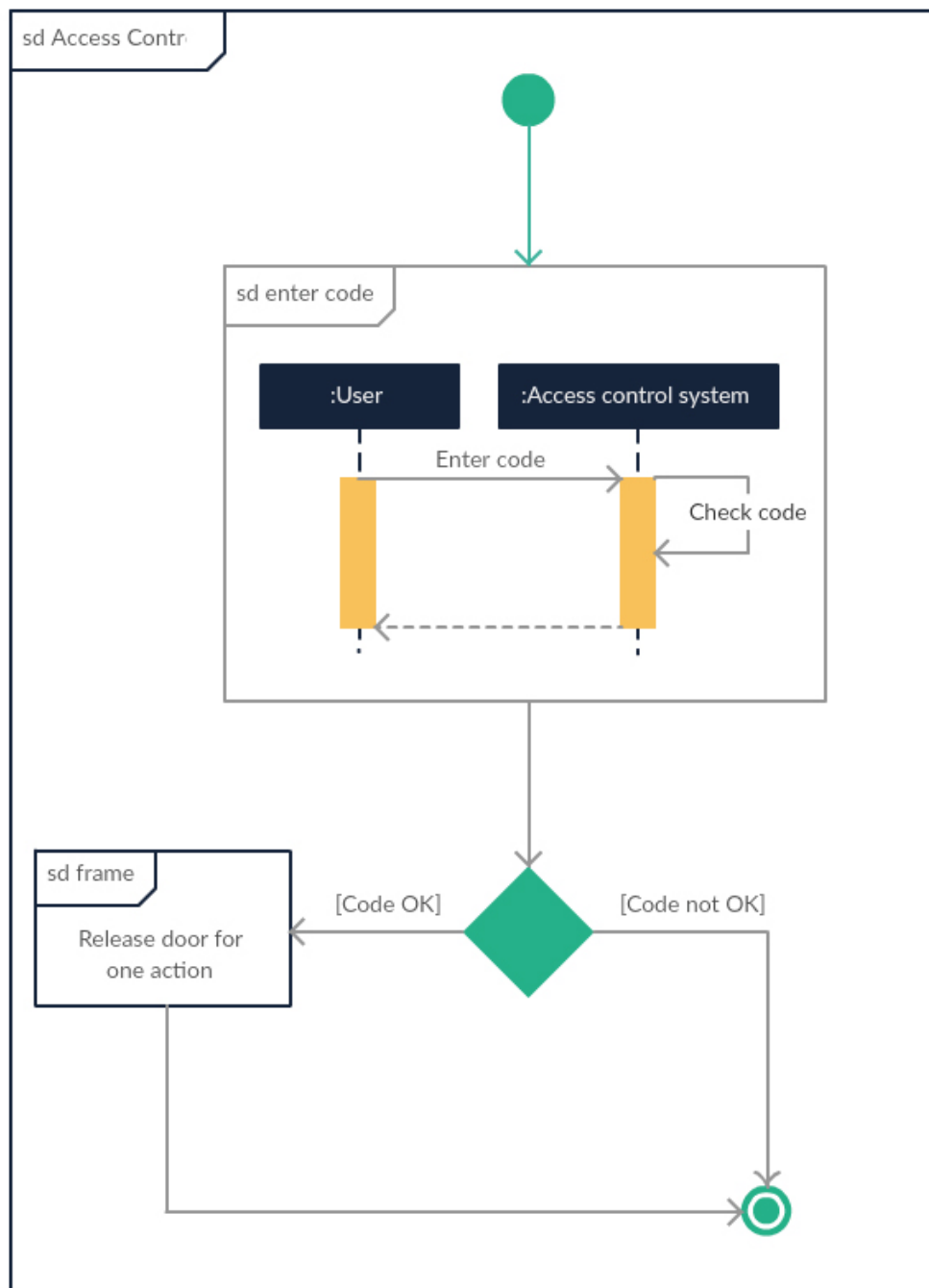
HOTEL MANAGEMENT SYSTEM



Interaction Overview UML

Interaction overview diagrams are behavioral diagrams that are very similar to activity diagrams. While activity diagrams show a sequence of processes, Interaction overview UMLs **show a sequence of interaction diagrams**

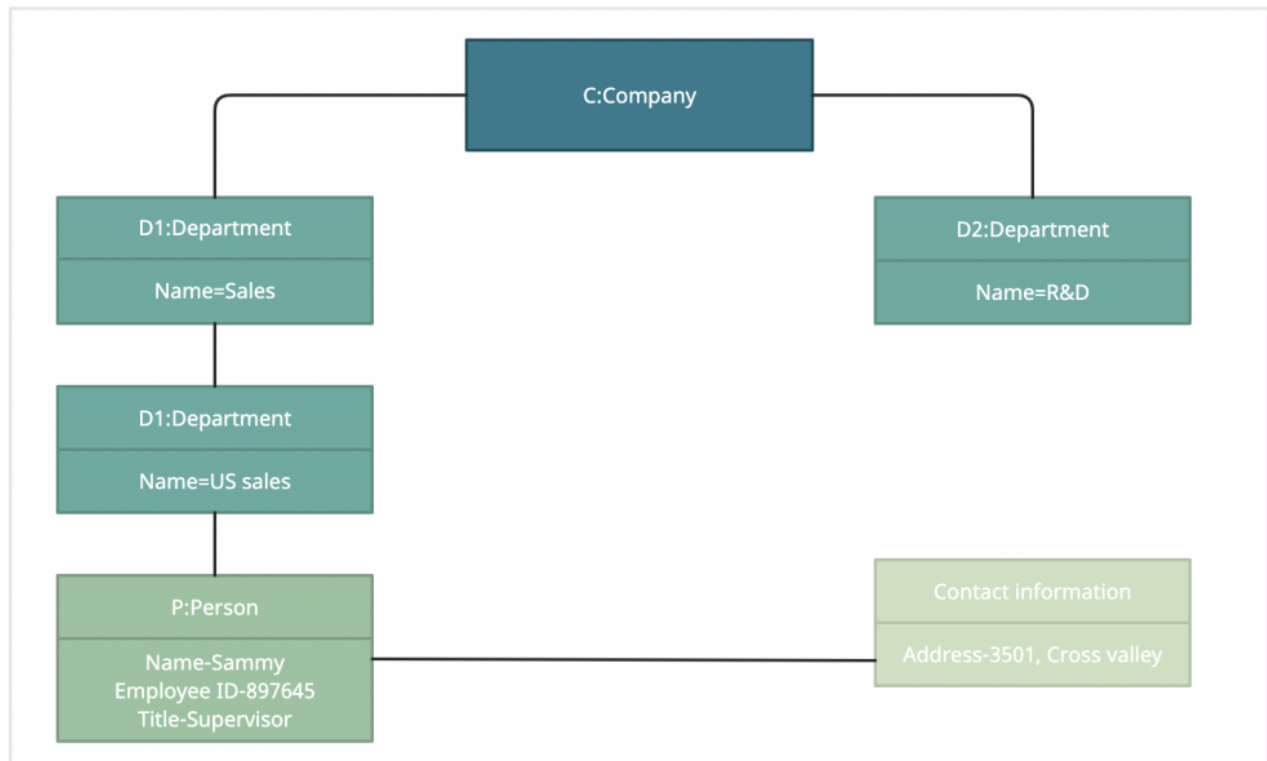
- Collection of interaction diagrams and their order.
Take a look at the example below:



Object UML

An object UML is a structural diagram, also known as an Instance diagram. Similar to class diagrams. They show the relationship between objects using **real world** examples.

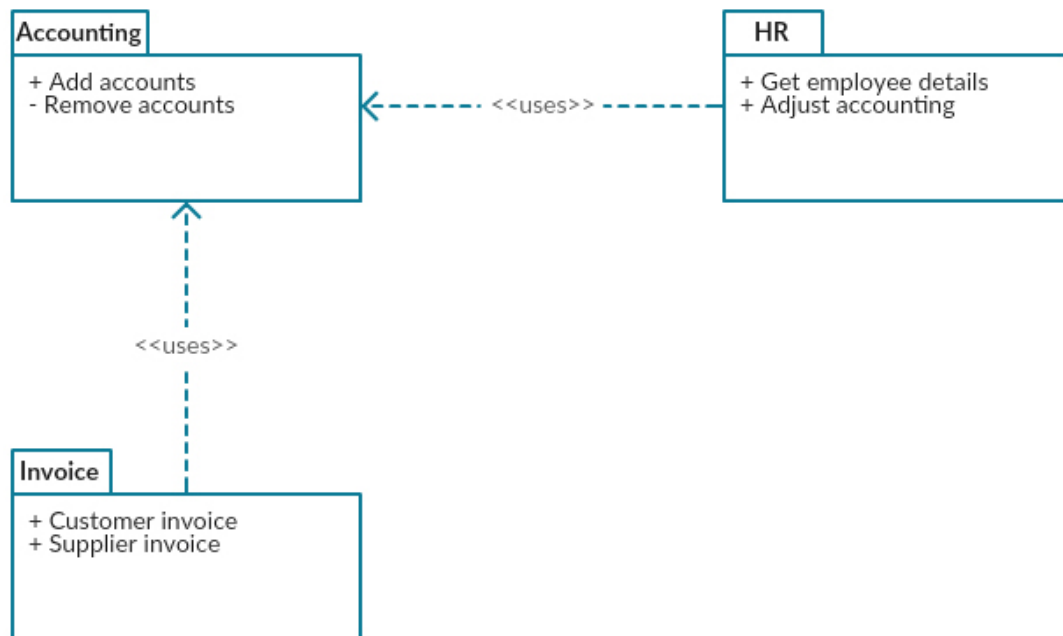
- Show what a system will look like at a given time, since there is data in the objects.
Below is an example of a Object UML:



Package UML

A package diagram shows the dependencies between different packages in a system. These are an example of structural UML diagrams.

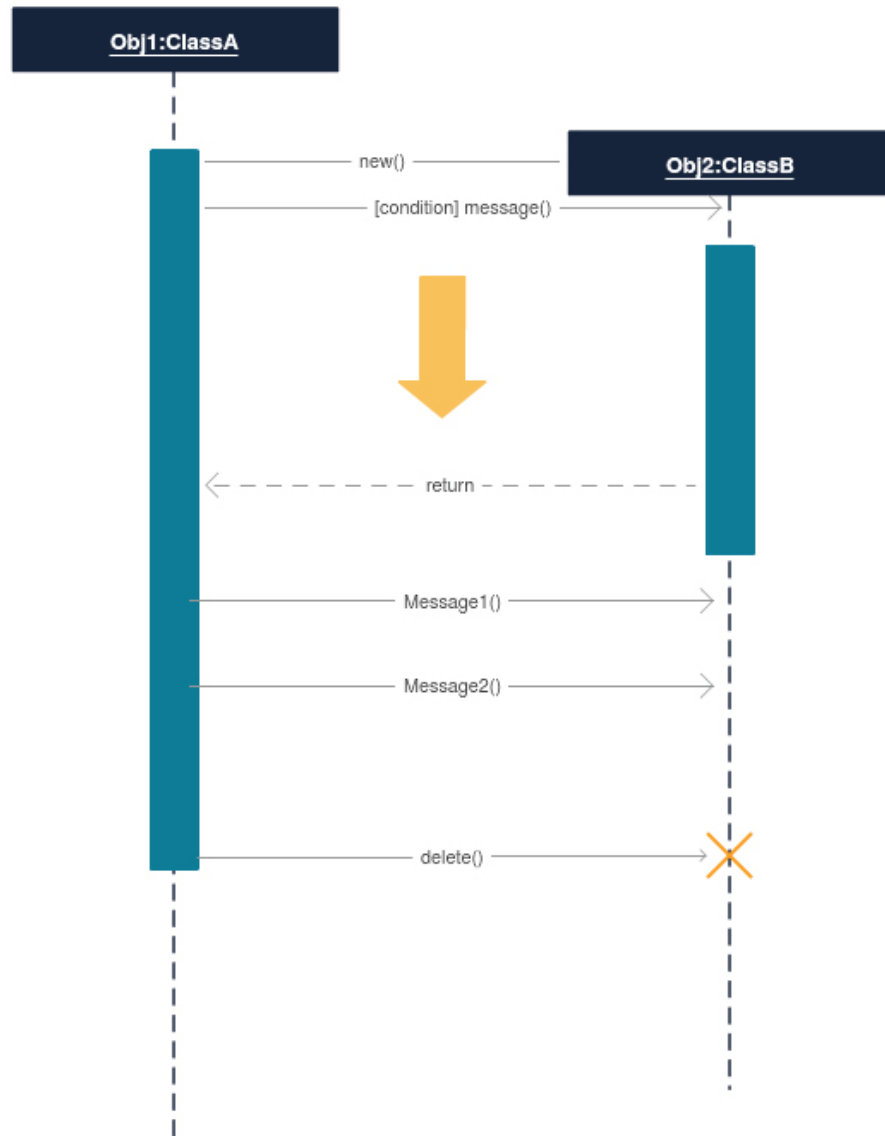
Below is an example of a Package UML Diagram for the packages of a company:



Sequence UML

Sequence UMLs are behavioral diagrams that show how objects interact with one another and the order those interactions occur.

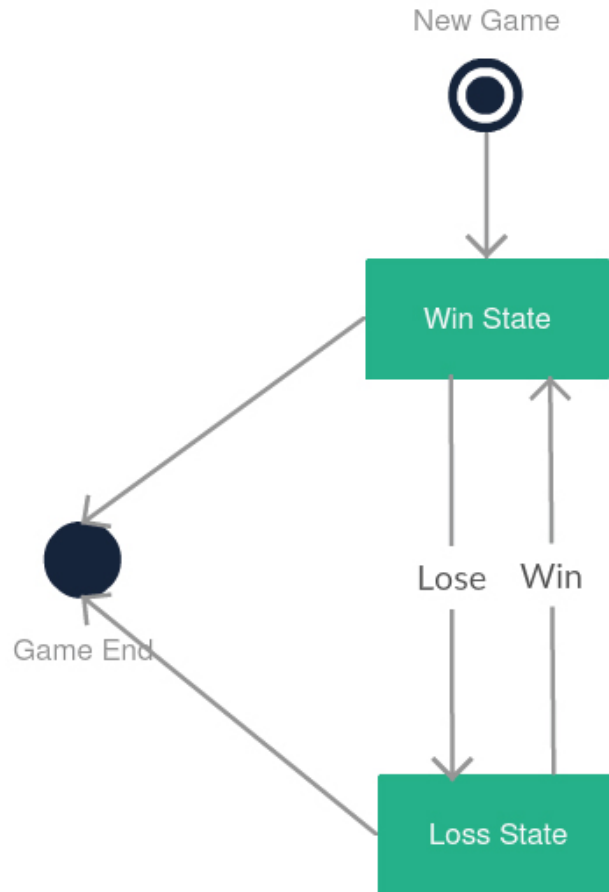
- Processes are represented vertically and interactions are shown as arrows. For example, the Sequence UML diagram depicted below?



State Machine UML

A state machine UML is a behavioral diagram similar to an activity UML. Useful for describing the behavior of objects that act differently according to the state they are in at the moment.

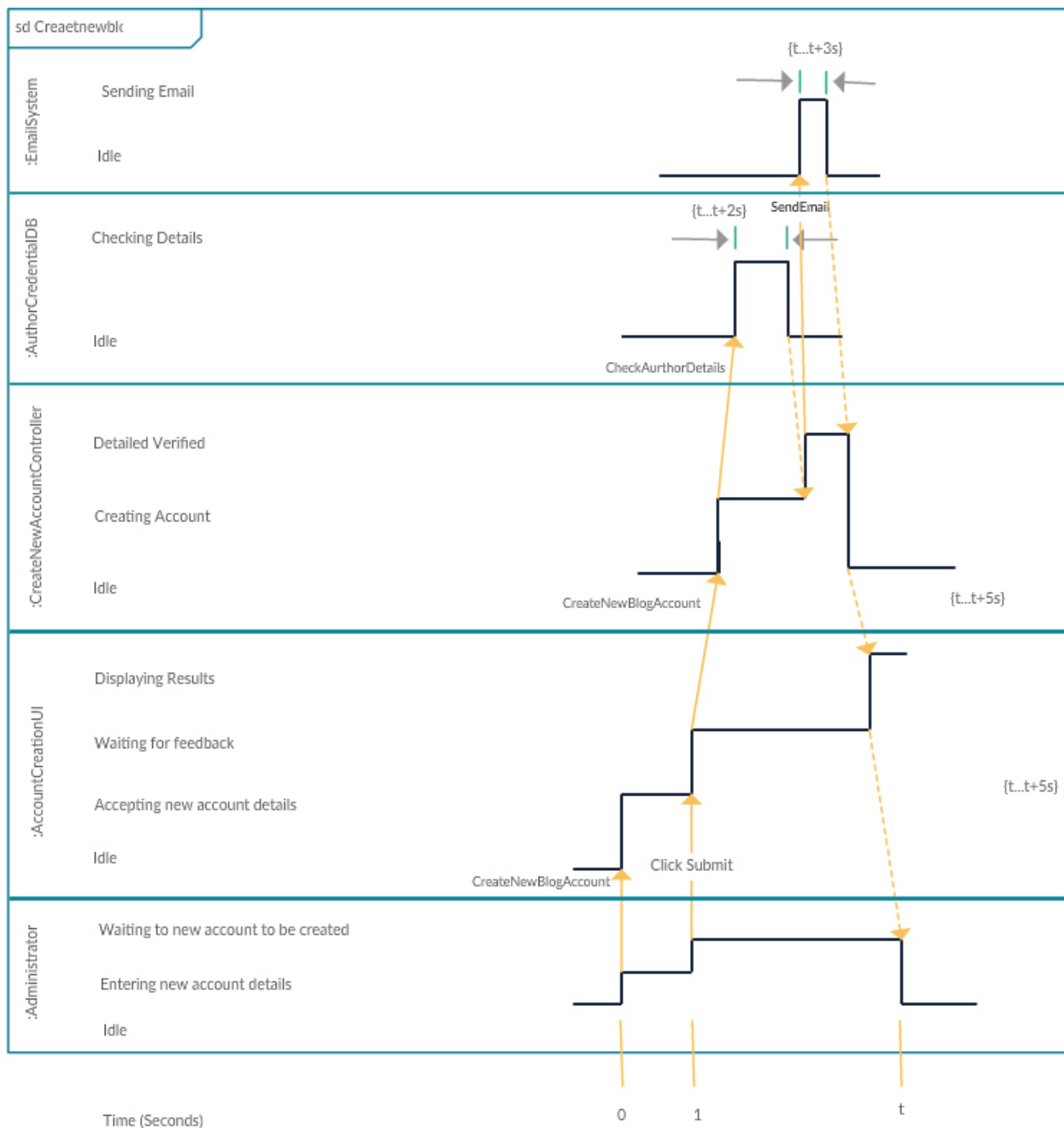
For example, the State Machine below shows the basic states and actions of a game.



Timing UML

A timing UML is a behavioral diagram similar to sequence diagrams. They represent the behavior of objects in a given time frame.

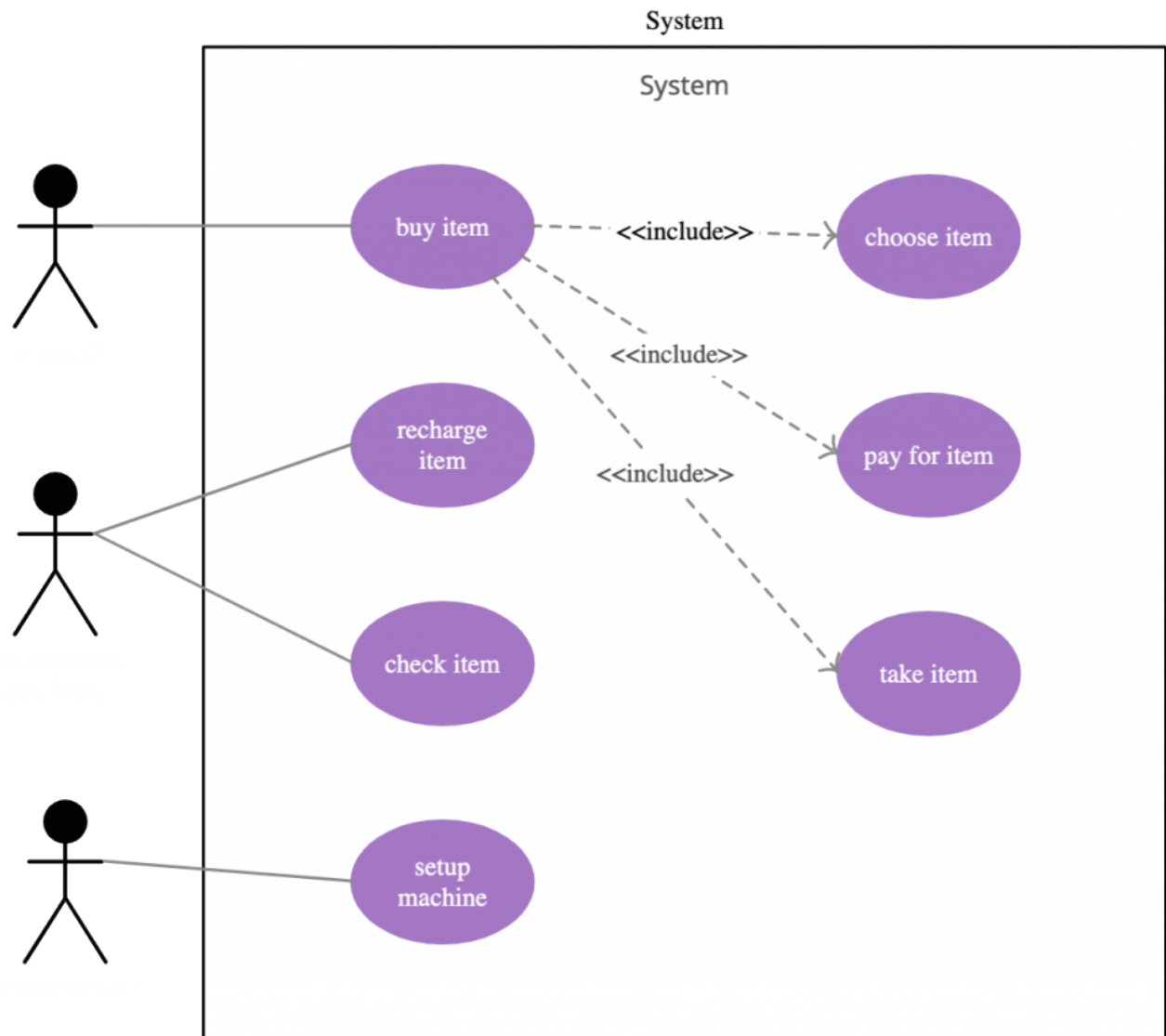
Take a look at an example here:



Use Case UML

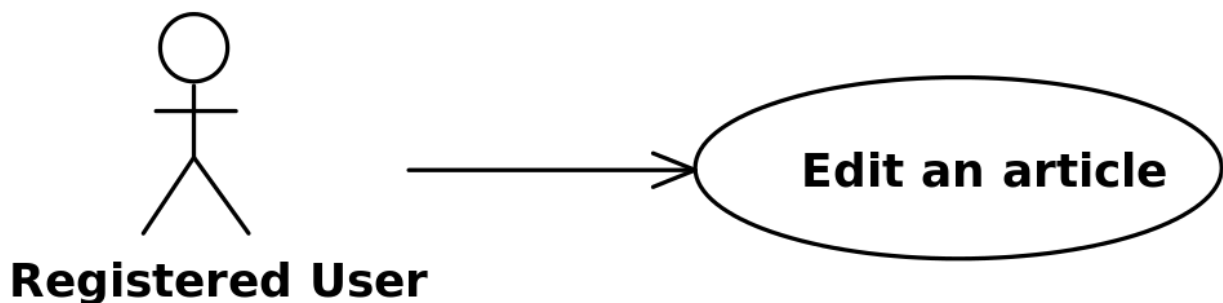
Use Case diagrams are behavioral and give a graphic overview of the actors involved in a system, different functions needed by those actors and how these different functions interact.

Check out the example below:



Scenario-Based Modeling

- **Use case** (scenario) defines how a user uses a system to accomplish a particular goal. For example, the basic Use-Case diagram for a wiki system below:

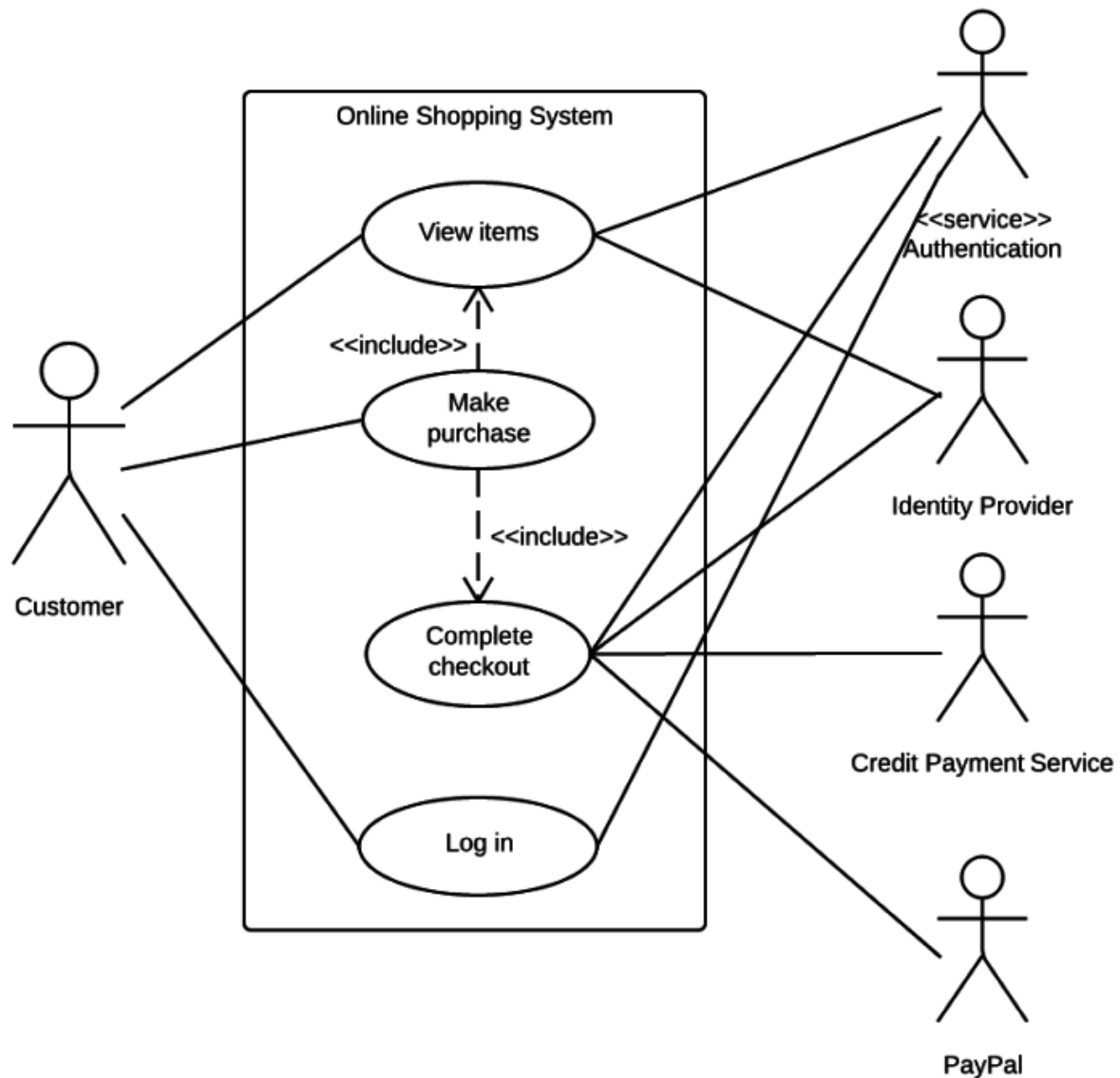


- A modeling technique that defines the features to be implemented and the resolution of any errors that may be encountered.
- A methodology used in system analysis to identify, clarify, and organize system requirements.
- A set of possible sequences of interactions between systems and users in a particular environment and related to a particular goal.

"These are the things for you to consider when developing a Use-Case"

1. What the main tasks or functions that are performed by an actor?
2. What system information will the actor acquire, produce or change?
3. Will the actor have to inform the system about changes in the external environment?
4. What information does the actor desire from the system?
5. Does the actor wish to be informed about unexpected changes?

Here's an example of a Use-Case diagram for an Online Shopping System:



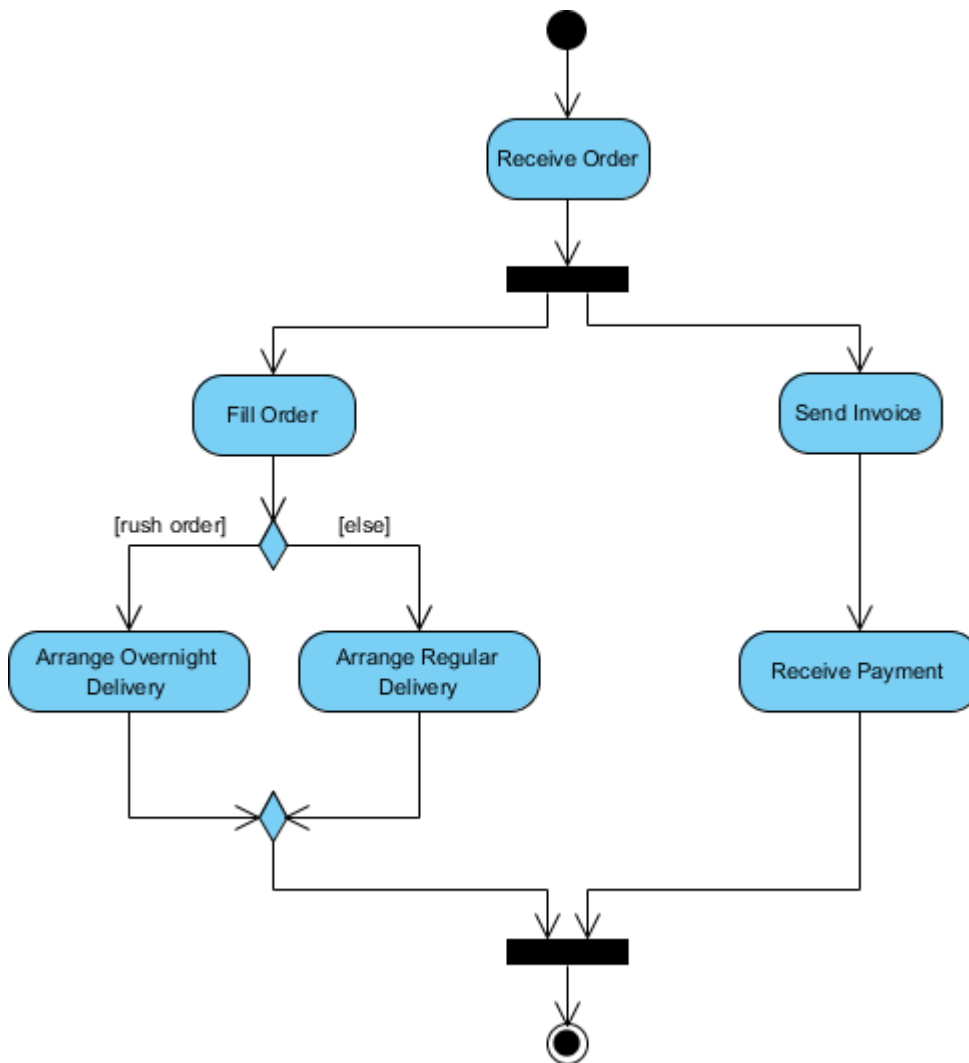
Exceptions

Exceptions that may cause a system to make unusual behavior.

1. Describe situations that may cause the system to exhibit unusual behavior.
2. Brainstorm to derive a reasonably complete set of exceptions for each use case.
3. Are there cases where a validation function occurs for the use case?
4. Handling exceptions may require the creation of additional use cases.

Activity Diagram

- 💡 Supplements the use case by providing a graphical representation.
- 💡 The **flow** of interaction between actor and the system within a specific scenario.



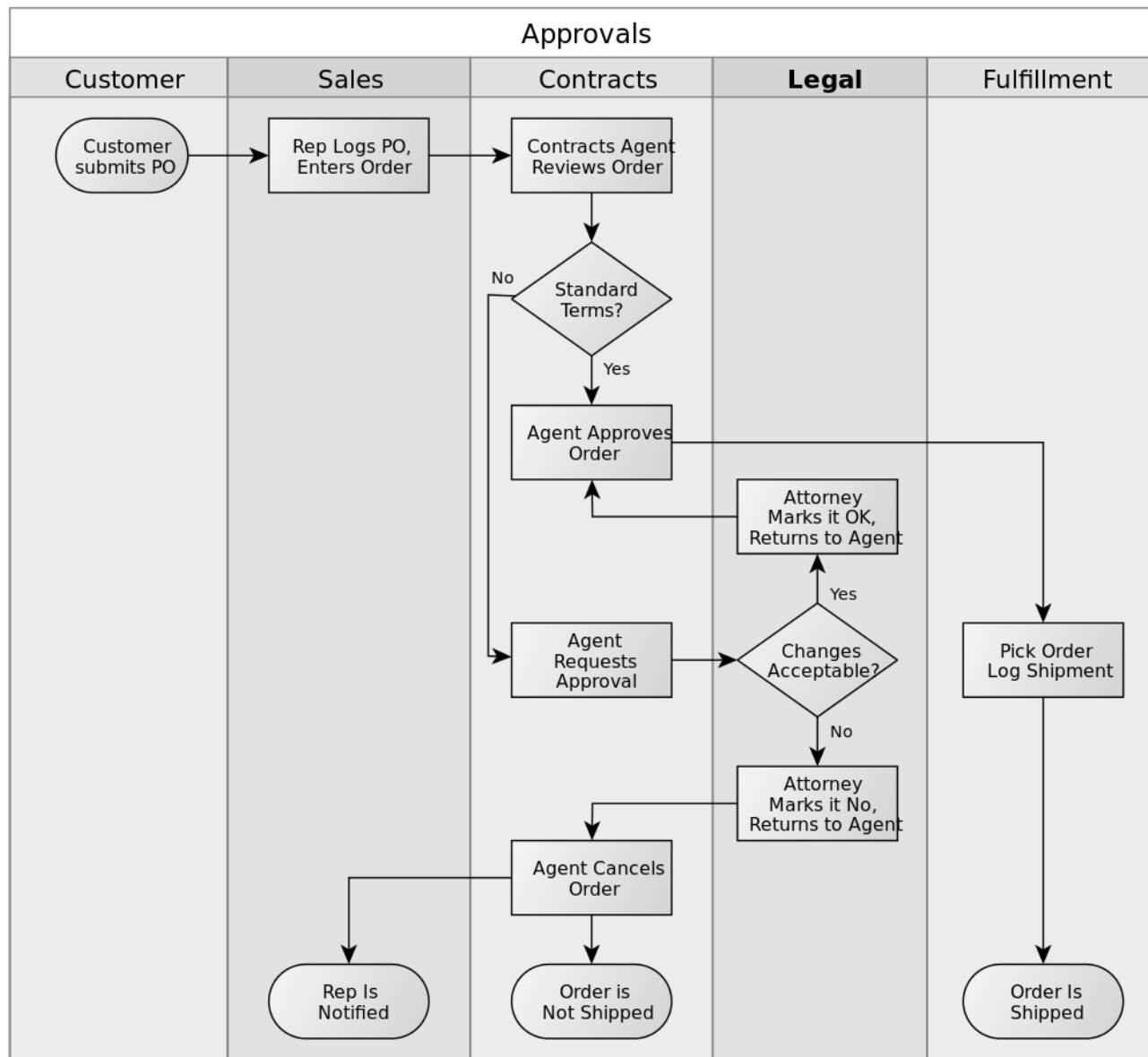
Like a flow chart, you can see the progress from one activity to another. You also clearly see the dynamic aspects of the system through the various paths between activities.

- Activity diagrams describe **how activities are coordinated** to provide a service which can be at different levels of abstractions.

Swimlane Diagram

A swimlane diagram is a type of flowchart that:

- Allows the modeler to represent the flow of activities described by the use-case
- Indicates which actor or analysis class has responsibility for the action described by an activity rectangle.



Class-Based Modeling


Class-based modelling represents:


- Objects that the system will manipulate operations (methods), relationships between the objects, collaborations that occur between the classes.


The elements of a class-based model include:

- Classes and objects
- Attributes and operations (methods)
- Collaboration diagrams and packages.


Identifying Classes

 Classes are determined by underlining each noun or noun phrase and entering it into a simple table.

 If a class is required to implement a solution, then it is part of the solution space.

 If a class is necessary only to describe a solution, it is part of the problem space.

Manifestations of Classes

 External entities - produce or consume informations.


e.g. other systems, devices, people.

 Things - are a part of the information domain for the problem


e.g. reports, displays, letters, signals.

 Occurrences or events - occur within the context of system operation.


e.g. completion of actions.

 Roles - played by people who interact with the system.

e.g. manager, engineer, salesperson.

 Organizational units - that are relevant to an application.

e.g. division, group, team.

 Places - establish the context of the problem and the overall function.

e.g. manufacturing floor.

 Structure - define a class of objects or related classes of objects.

e.g. sensors, computers.

Defining Operations

1. An operation is a method or function that can be performed by a class
2. Operations define the behavior of a class -- what a class can do.
3. Operations can perform computation, take an action, call another method, etc.

Class Types

1. **Entity Classes**, also called model or business classes, are extracted directly from the statement of the problem.
2. **Boundary classes**, are used to create the interface that the user sees and interacts with the software.
3. **Controller classes**, manage a "unit of work" from start to finish.

Responsibilities

- System intelligence should be distributed across classes to best address the needs of the problem.
- Each responsibility should be stated as generally as possible.

- Information and the behavior related to it should reside within the same class.
- Information about one thing should be localized with a single class, not distributed across multiple classes.
- Responsibilities should be shared among related classes when appropriate.

Collaborations

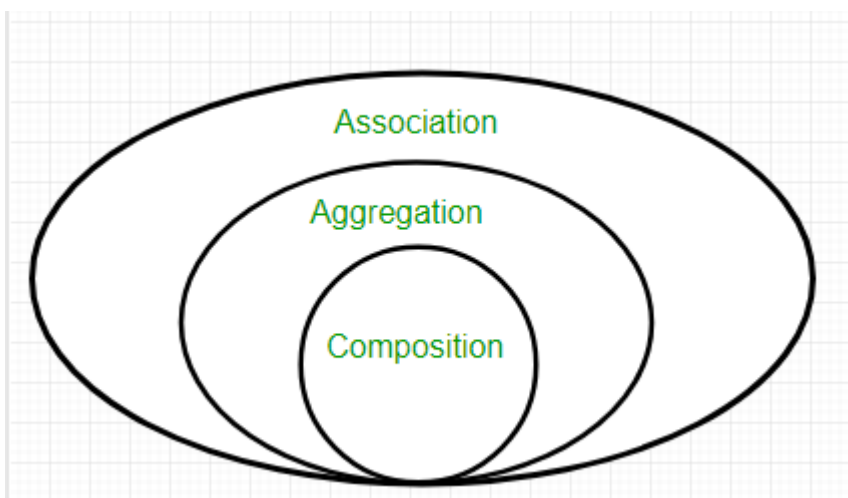
Classes fulfill their responsibilities in one or two ways:

1. A class can use its own operations to fulfill a particular responsibility.
2. A class can collaborate with other classes.

Collaborations identify relationships between classes.

Collaborations are identified by determining whether a class can fulfill each responsibility independently.

Association, Aggregation, and Composition



Association is a relationship between two classes where one class uses another.

- there is a **subclass** and **superclass**.
- Association is non-directional.



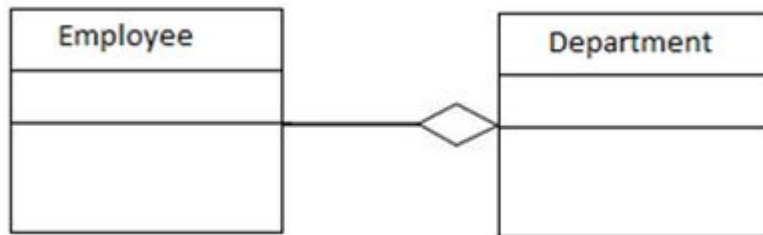
UML Representation of Association



Aggregation and Composition are two forms of **Association**

Aggregation is the directional relationship between two classes.

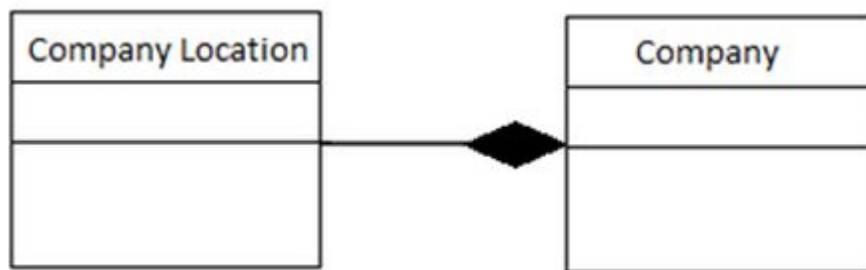
- Specifies that a object contains another object. (**HAS-A relationship**)
- Mutual dependencies between objects.



Composition can be recognized as a special type of aggregation.

- The child object does not have their own life cycle. The child object's life depends on the parent's life cycle.

An Example of a composition can be seen below:



Multiplicity

Can be a set for attributes, operations, and associations in a UML class diagram, and for associations in a use-case diagram.

Multiplicity is an indication of **how many objects** may participate in the given relationship or the allowable number of instances of the element.

Dependencies

Where one object is dependent on another object.

A dependency exists between two elements if changes to the definition of one element may cause changes to the other.

Behavioral Modeling

- The behavioral model indicates how software will respond to external events.
- Evaluate all use-cases to fully understand the sequence of interaction within the system.

- Identify events that drive the interaction sequence and understand how these events relate to specific objects.
- Create a sequence for each use-case.
- Build a state diagram for the system.
- Review the behavioral model to verify accuracy and consistency.

State Diagrams

In the context of behavioral modeling, two different characterizations of states must be considered:

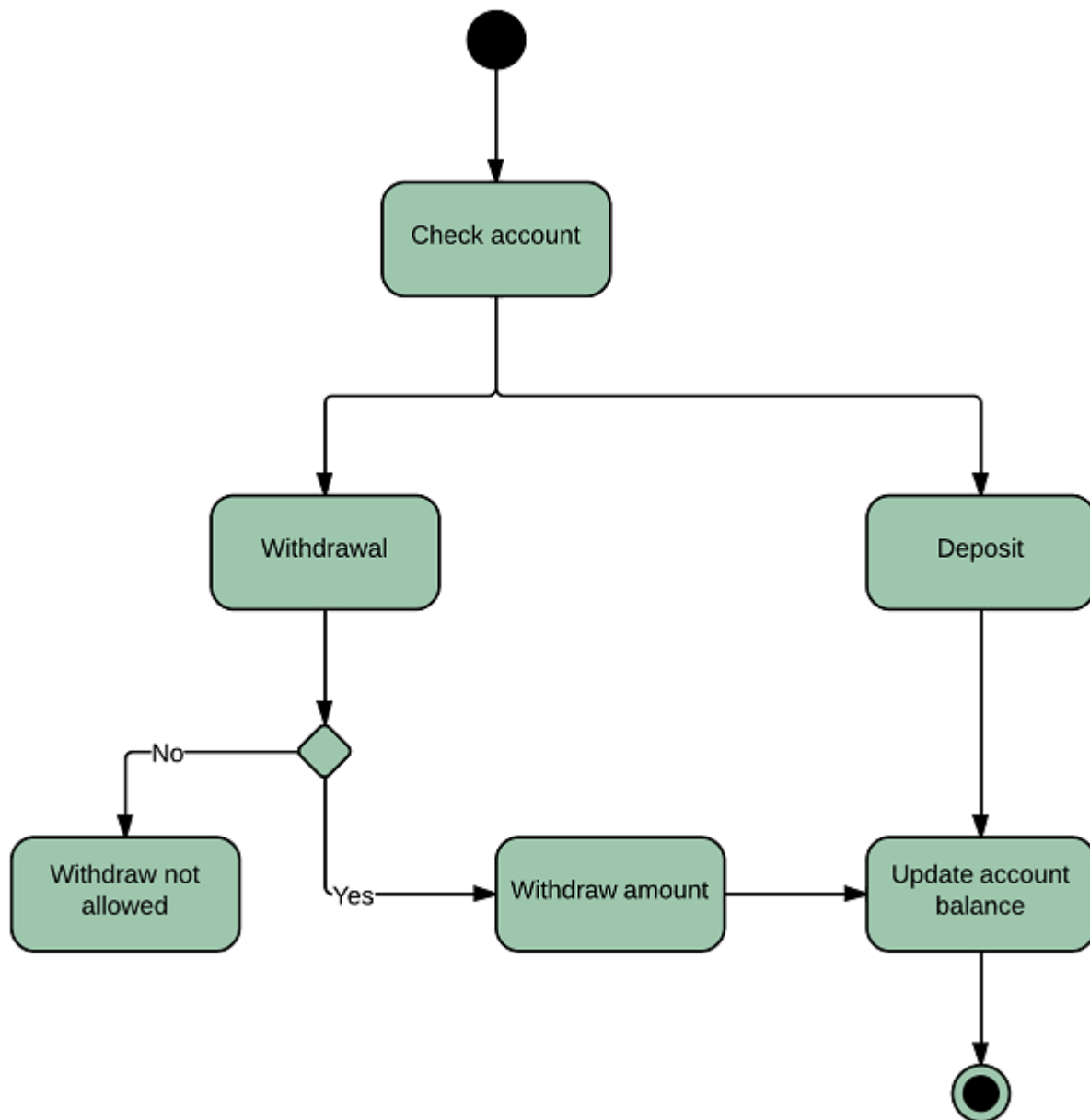
1. The state of each class as the system performs its function
2. The state of the system as observed from the outside as the system performs its function.
The state of a class takes on both passive and active characteristics:
3. A passive state is simply the current status of all of an object's attributes.
4. The active state of an object indicates the current status of the object as it undergoes a continuing transformation of processing.

The States of a System

1. State - a set of observable circumstances that characterizes the behavior of a system at a given time.
2. State transition - the movement from one state to another.
3. Event - an occurrence that causes the system to exhibit some predictable form of behavior.
4. Action - process that occurs as a consequence of making a transition.

Activity Diagrams

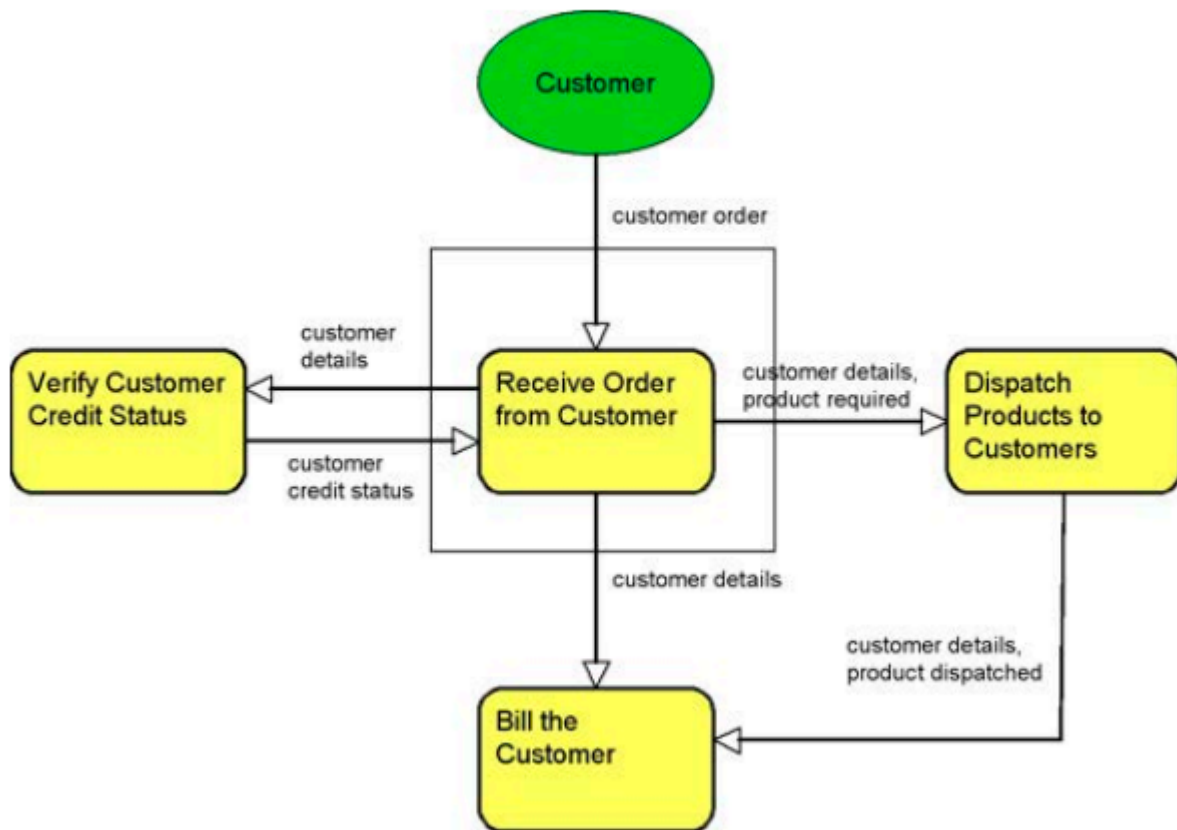
A **flowchart** to represent the flow from one activity to another activity.
The activity can be described as an operation of the system.



Data Flow Diagram

- A way of representing a flow of a data of a process or a system.
- The DFD also provides information about the output and input of each entity and the process itself.

- **No control flow**, there are no decision rules and no loops.



Object Diagram

- Represents a snapshot of the objects in a system at a point in time.
- Shows instances rather than classes, therefore it is sometimes called an instance diagram.
- When to use object diagrams:
 1. To show examples of objects and how they're related based on a specific multiplicity number.
 2. To show instances with values for their attributes.

Package Diagram

- Used to take any construct in UML and group its elements together into higher-level units.
- Used most often to group classes.

- Corresponds to packages in Java.
- Represented by a tabbed folder, where the tab contains the package name.
- Can show dependencies between packages.

Deployment Diagram

- Shows the physical architecture of the hardware and software of the deployed system.
- Typically contain components of packages.
- Physical relationships among software and hardware in a delivered systems.
- Explains how a system interacts with the external environment.

In Summary the key diagrams we learned today are purposed for:

Use Case Diagram - helps describe how people interact with the system.

Activity Diagram - shows the context for uses cases and details of how they work.

Class Diagram - shows the attributes and operations in domain classes and relationships between classes.

- Drawn from the software perspective, shows design classes, their attributes and operations, and their relationships with the domain classes.

State Diagram - shows the various states of a domain class and events that change that states.

Sequence Diagram - helps combine use cases **in order** to see what happens in the software.

Deployment Diagram - shows the physical layout of the software.