GANPAT UNIVERSITY

U. V. PATEL COLLEGE OF ENGINEERING

2CEIT502

SOFTWARE ENGINEERING

UNIT 8

UNIFIED MODELLING LANGUAGE (UML)

Prepared by: Prof. Y. J. Prajapati (Asst. Prof in IT Dept. , UVPCE)

Outline

- Overview of object oriented concepts
- Advantage of OOD
- UML diagrams
- Use Case Diagram
- Class Diagram
- Sequence Diagram
- collaboration Diagram
- Activity Diagram
- State chart Diagram

Overview of object oriented concepts

- What is Object-Orientation?
- It means that we organize software as a collection of discrete objects that incorporate both data structure and behavior
- **Object = Data structure + Behavior**

  (*attributes*) (*operations*)

- Four aspects of OO approach (characteristics) :
  - Identity
  - Classification
  - Inheritance
  - polymorphism

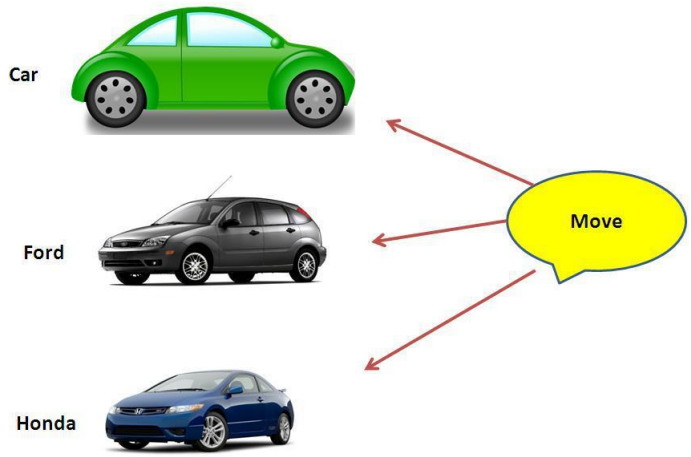
- Identity
  - It means that data is quantized into discrete, distinguishable entities called objects
  - Ex. White queen in chess
  - Object can be concrete, such as a file in a file system, or conceptual, such as a scheduling policy in a multiprogramming operating system
  - Each object has its own inherent identity
  - Two objects are different even if the attribute values are same

Overview of object oriented concepts

- Classification
  - Objects with same data structure and behavior are grouped together into a class
  - A class is an abstraction that describes a properties important to an application and ignores the rest
  - Any choice of classes is arbitrary and depends on the application
  - Each class describes infinite set of individual objects
  - **Object is said to be instance of its class**
- Inheritance
  - It is the sharing of attributes and operations among classes based on a hierarchical relationship
  - A super class has general information that subclass refine and elaborate
  - Each subclass inherits all features of its super class and adds its own unique features
  - Subclasses need not to repeat the features of the super class
  - Ex. Scrolling window and Fixed window are subclasses of Window class

Overview of object oriented concepts

- Polymorphism
  - The same operation behave differently for different classes
  - Ex. Move operation for a pawn and the queen in a chess game
  - An operation is a procedure or transformation that an object performs
  - An implementation of the operation by a specific class is called a method



Advantage of OOD,

- ***Faster Development***
- ***Reuse of Previous work***
- ***Increased Quality***
- ***Modular Architecture***
- ***Client/Server Applications***
- ***Better Mapping to the Problem Domain***

Unified modelling language(UML)

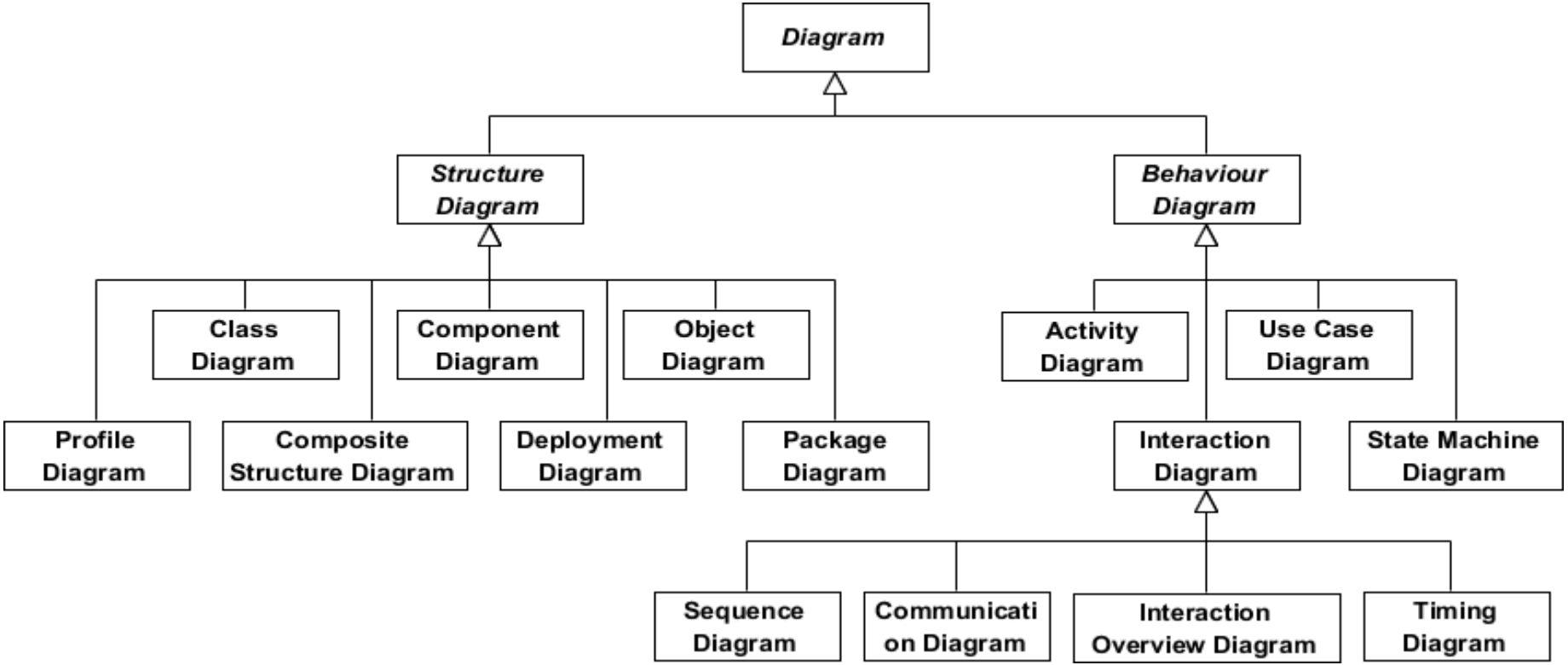
What is UML?

- UML stands for “Unified Modeling Language”
- It is a industry-standard graphical language for specifying, visualizing, constructing, and documenting the artifacts of software systems
- The UML uses mostly graphical notations to express the OO analysis and design of software projects.
- Simplifies the complex process of software design

Why UML for Modeling

- Use graphical notation to communicate more clearly than natural language (imprecise) and code(too detailed).
- Help acquire an overall view of a system.
- UML is not dependent on any one language or technology.

UML diagrams



UML diagrams

- Structure diagrams show the static structure of the system and its parts on different abstraction and implementation levels and how they are related to each other. The elements in a structure diagram represent the meaningful concepts of a system, and may include abstract, real world and implementation concepts, there are seven types of structure diagram as follows:
  - Class Diagram
  - Component Diagram
  - Deployment Diagram
  - Object Diagram
  - Package Diagram
  - Composite Structure Diagram
  - Profile Diagram

- Behavior diagrams show the dynamic behavior of the objects in a system, which can be described as a series of changes to the system over time, there are seven types of behavior diagrams as follows:
  - Use Case Diagram
  - Activity Diagram
  - State Machine Diagram
  - Sequence Diagram
  - Communication Diagram
  - Interaction Overview Diagram
  - Timing Diagram

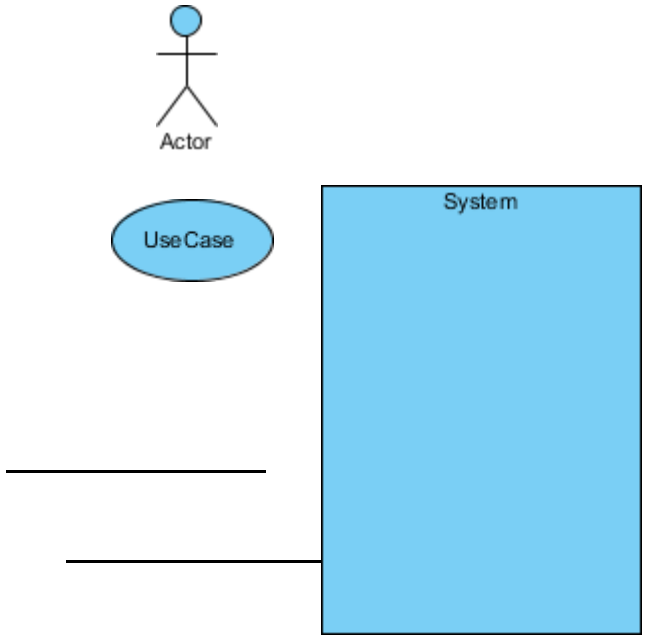
Use Case Diagram

- A use case diagram is a dynamic or behavior diagram in UML. Use case diagrams model the functionality of a system using actors and use cases. Use cases are a set of actions, services, and functions that the system needs to perform
- Purposes of use case diagrams
  - Used to gather the requirements of a system.
  - Used to get an outside view of a system.
  - Identify the external and internal factors influencing the system.
  - Show the interaction among the requirements are actors.

- Use case diagrams are used to visualize, specify, construct, and document the (intended) behavior of the system, during requirements capture and analysis.
- Provide a way for developers, domain experts and end-users to Communicate.
- Serve as basis for testing.
- Use case diagrams contain use cases, actors, and their relationships.

Use Case Diagram

- **Actor:** A role that a user plays with respect to the system, including human users and other systems. e.g., inanimate physical objects (e.g. robot); an external system that needs some information from the current system.
- **Use case:** A set of scenarios that describing an interaction between a user and a system, including alternatives.
- **System boundary**: rectangle diagram representing the boundary between the actors and the system.
- **Association:** Communication between an actor and a use case; Represented by a solid line.

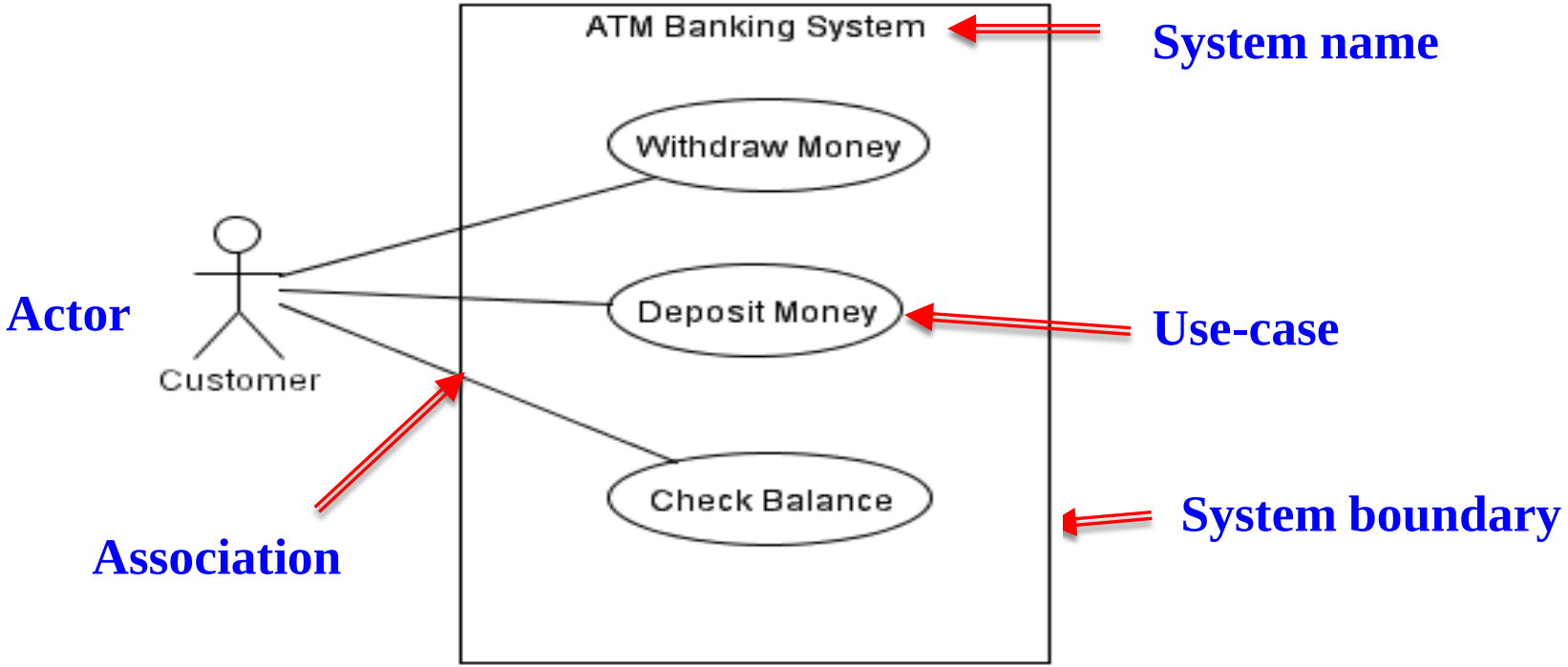


Use Case Diagram

- A use case describes a sequence of actions a system performs to yield an **observable result** or **value** to a particular actor
- Naming convention = verb + noun (or) verb + noun-phrase,
  - e.g. withdraw cash
- A good use case should:
  - Describe a sequence of transactions performed by a system that produces a measurable result (goal) for a particular actor
  - Describe the behavior expected of a system from **a user's perspective**
  - Enable the system analyst to understand and model a system from a **high-level business viewpoint**
  - Represent the interfaces that a system makes visible to the external entities and the interrelationships between the actors and the system

- Identifying use cases for a system
- What are the tasks of each actor?
- Will any actor create, store, change, remove, or read the information?
- Will any actor need to inform the system about the sudden, external changes?
- Does any actor need to informed about certain occurrences in the system?
- What use cases will support and maintain the system?
- Can all functional requirements be performed by the use cases?

Example


ATM Banking System
Withdraw Money
Deposit Money
Check Balance
Customer
Actor
Association
System name
Use-case
System boundary

Use Case Relationships

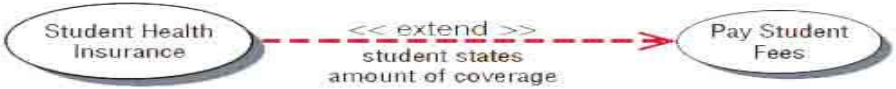
- **INCLUDES.** The includes relationship (also called uses relationship) describes the situation in which a use case contains behavior that is common to more than one use case. In other words, the common use case is included in the other use cases. A dotted arrow that points to the common use case indicates the includes relationship. An example would be a use case Pay Student Fees that is included in Enroll in Course and Arrange Housing, because in both cases students must pay their fees. This may be used by several use cases. The arrow points toward the common use case.
- **EXTENDS.** The extends relationship describes the situation in which one use case possesses the behavior that allows the new use case to handle a variation or exception from the basic use case. For example, the extended use case Student Health Insurance extends the basic use case Pay Student Fees. The arrow goes from the extended to the basic use case.
- **GENERALIZES.** The generalizes relationship implies that one thing is more typical than the other thing. This relationship may exist between two actors or two use cases. For example, a Part-Time Student generalizes a Student. Similarly, some of the university employees are professors. The arrow points to the general thing.

Example

Communicates Relationship

Includes Relationship

Generalizes Relationship



Extends Relationship

Class Diagram

- The **UML** Class diagram is a graphical notation used to construct and visualize object oriented systems. A class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's:
- classes,
- their attributes,
- operations (or methods),
- and the relationships among objects.

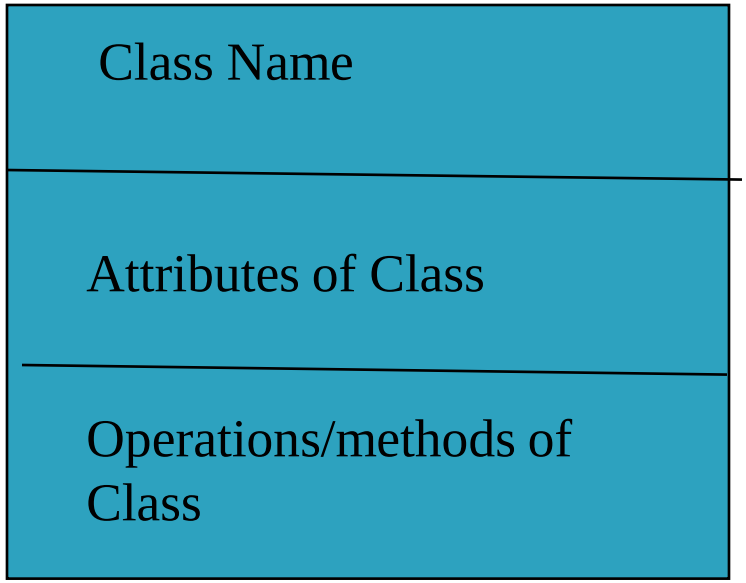
- Class diagram is a static diagram. It represents the static view of an application.

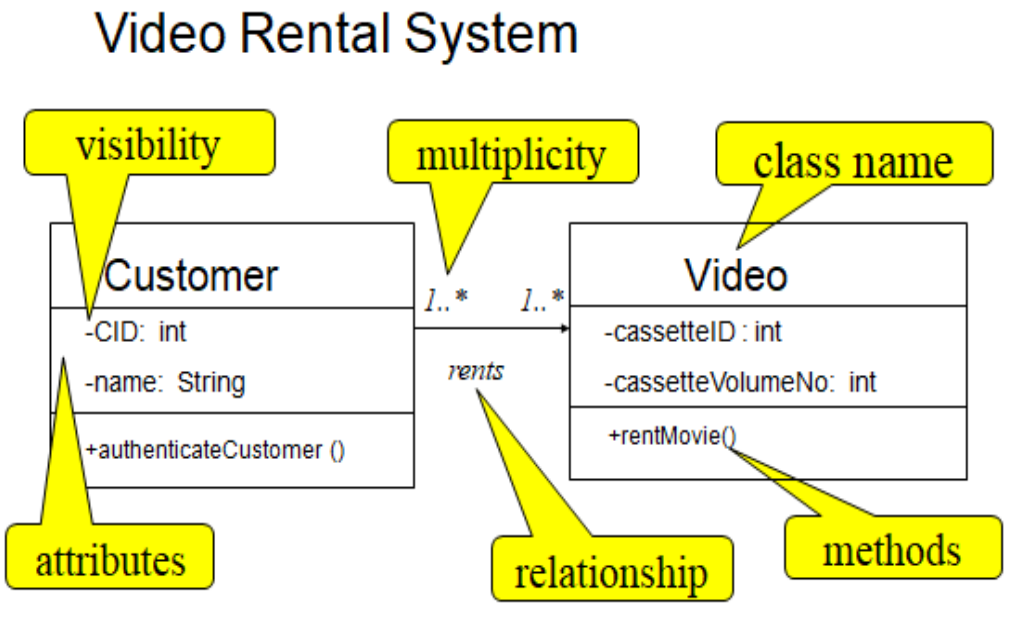
Purpose of Class Diagrams

- Analysis and design of the static view of an application.
- Describe responsibilities of a system.
- Base for component and deployment diagrams.
- Forward and reverse engineering.

Class Diagram

- UML Representation of Class





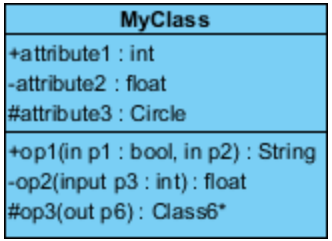
Class Diagram

Visibility of Attributes and Operations

In object-oriented design, there is a notation of visibility for attributes and operations. UML identifies four types of visibility: **public**, **protected**, **private**, and **package**.

- The +, -, # and ~ symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.
- + denotes public attributes or operations
- - denotes private attributes or operations
- # denotes protected attributes or operations
- ~ denotes package attributes or operations

- Class Visibility Example

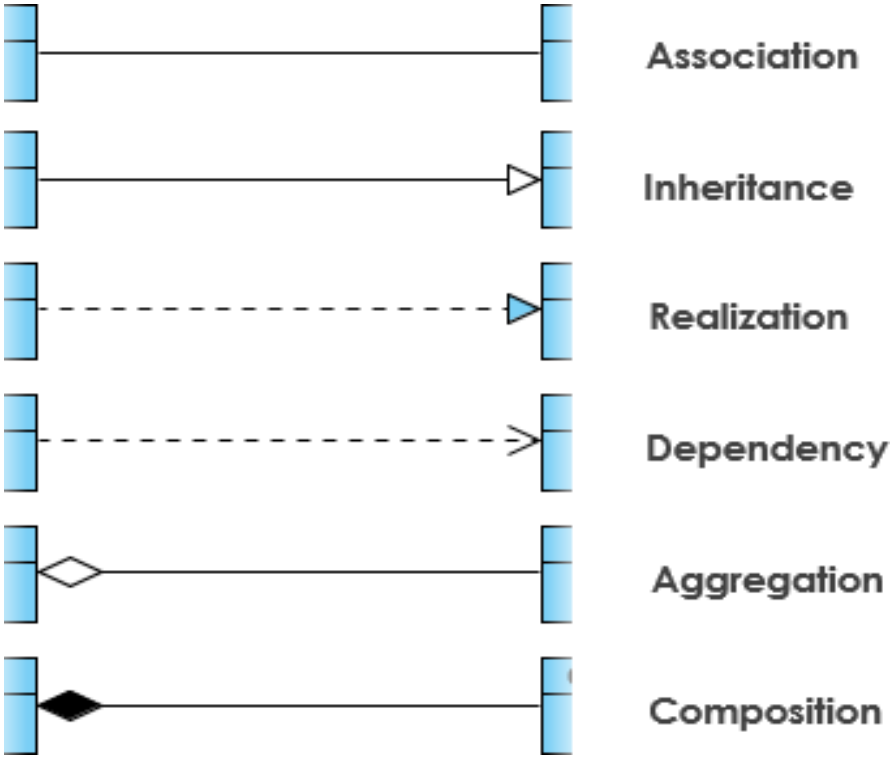


Access Right	public (+)	private (-)	protected (#)	Package (~)
Members of the same class	yes	yes	yes	yes
Members of derived classes	yes	no	yes	yes
Members of any other class	yes	no	no	in same package

Class Diagram

Relationships among Classes

- Represents a connection between multiple classes or a class and itself
- basic categories:
  - association relationships
  - generalization relationships
  - aggregation relationships
  - Composition relationships



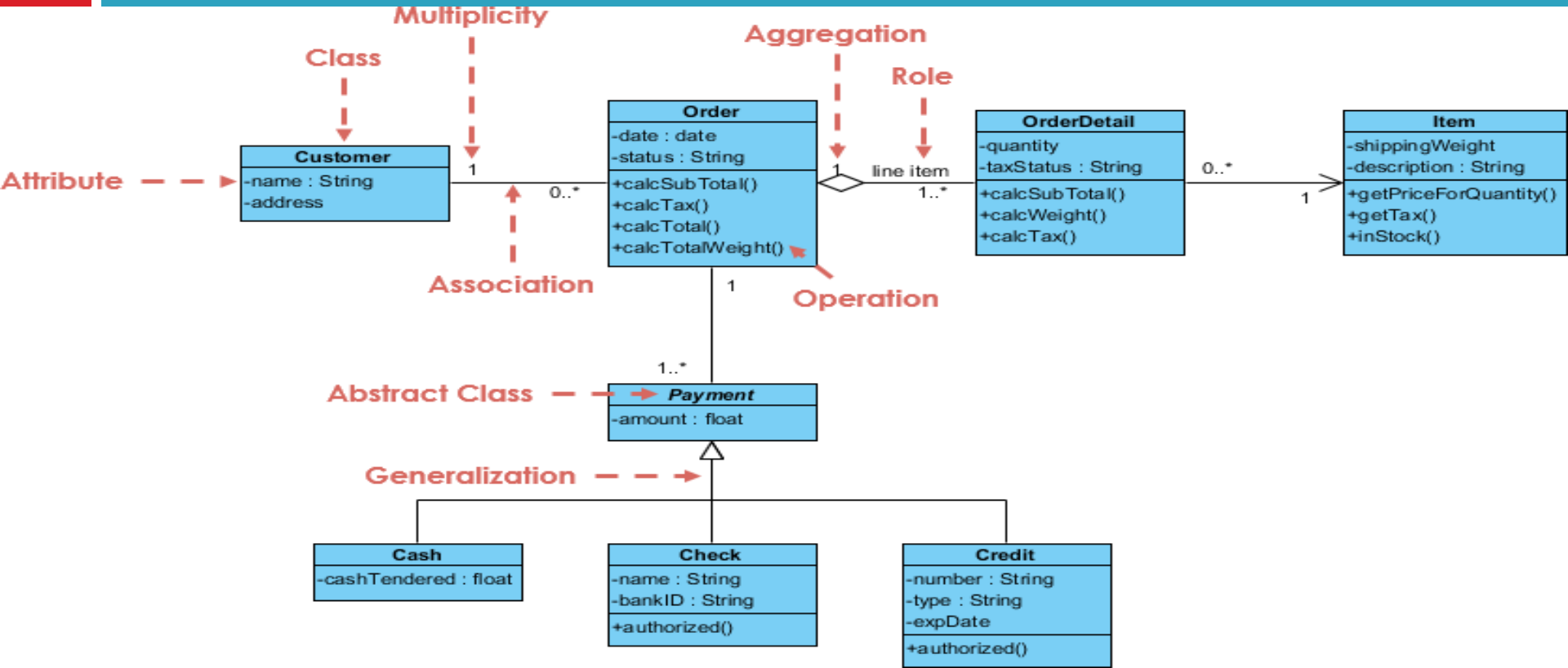
Class Diagram

Multiplicity

How many objects of each class take part in the relationships and multiplicity can be expressed as:

- Exactly one - 1
- Zero or one - 0..1
- Many - 0..* or *
- One or more - 1..*
- Exact Number - e.g. 3..4 or 6
- Or a complex relationship - e.g. 0..1, 3..4, 6.* would mean any number of objects other than 2 or 5

Example


Multiplicity
Class
Aggregation
Role
Attribute
Customer
-name : String
-address
1
0..*
Order
-date : date
-status : String
+calcSubTotal()
+calcTax()
+calcTotal()
+calcTotalWeight()
1
line item
1..*
OrderDetail
-quantity
-taxStatus : String
+calcSubTotal()
+calcWeight()
+calcTax()
0..*
1
Item
-shippingWeight
-description : String
+getPriceForQuantity()
+getTax()
+inStock()
Association
Operation
1
1..*
Abstract Class
Payment
-amount : float
Generalization
Cash
-cashTendered : float
Check
-name : String
-bankID : String
+authorized()
Credit
-number : String
-type : String
-expDate
+authorized()

Sequence Diagrams

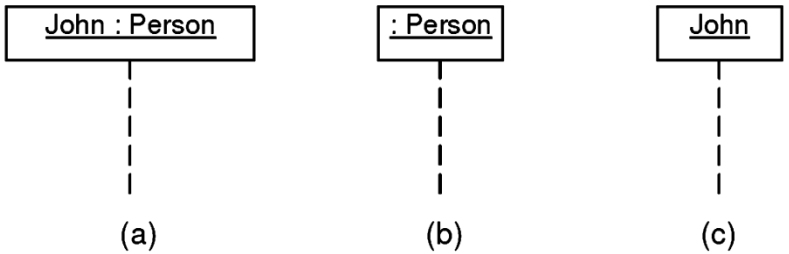
- A sequence diagram simply depicts interaction between objects in a sequential order i.e. the order in which these interactions take place.
- We can also use the terms event diagrams or event scenarios to refer to a sequence diagram. Sequence diagrams describe how and in what order the objects in a system function.
- These diagrams are widely used by businessmen and software developers to document and understand requirements for new and existing systems.
- Sequence diagrams are used to capture the order of messages flowing from one object to another.

The purpose of interaction diagram is –

- To capture the dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

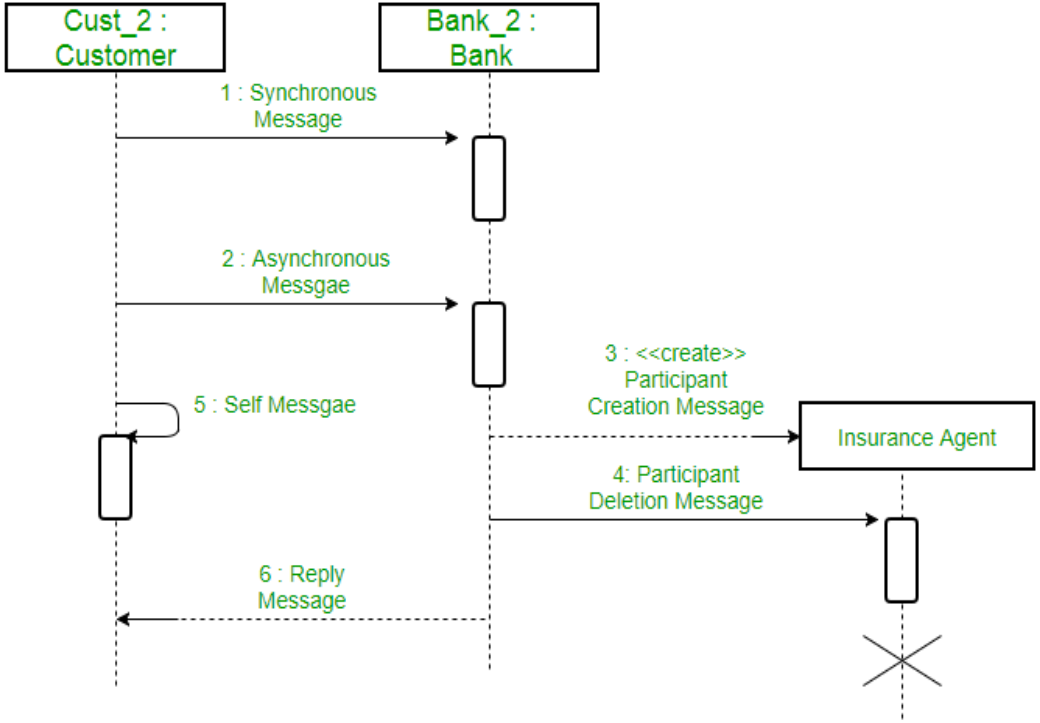
Sequence Diagram Notations

- **Lifelines –** A lifeline is a named element which depicts an individual participant in a sequence diagram. So basically each instance in a sequence diagram is represented by a lifeline. Lifeline elements are located at the top in a sequence diagram.



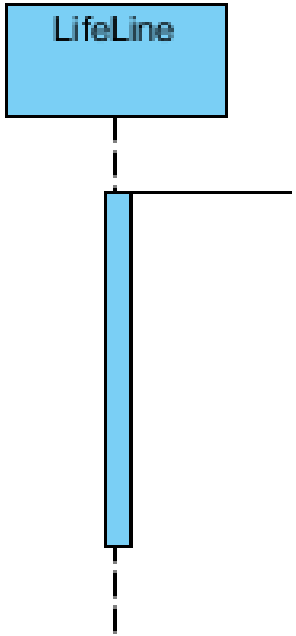
Sequence Diagram Notations

- **Messages–**
  Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline. We represent messages using arrows. Lifelines and messages form the core of a sequence diagram.



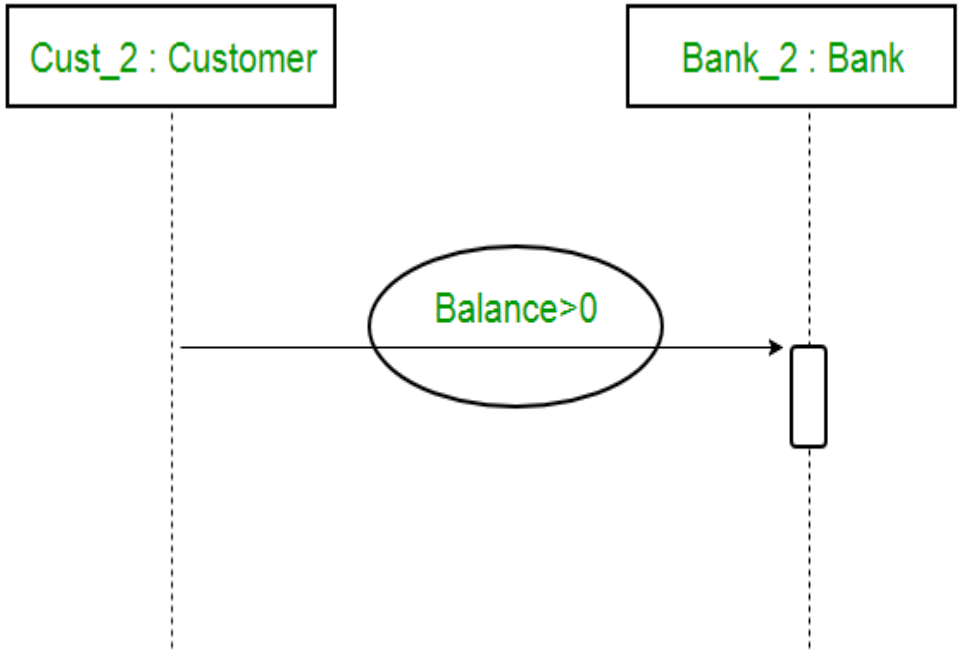
Sequence Diagram Notations

- **Activations**
  - A thin rectangle on a lifeline) represents the period during which an element is performing an operation.
  - The top and the bottom of the of the rectangle are aligned with the initiation and the completion time respectively



Sequence Diagram Notations

- **Guards –** To model conditions we use guards in UML. They are used when we need to restrict the flow of messages on the pretext of a condition being met. Guards play an important role in letting software developers know the constraints attached to a system or a particular process.

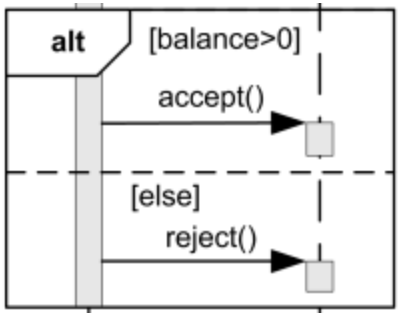


Sequence Diagram Notations

Combined Fragment

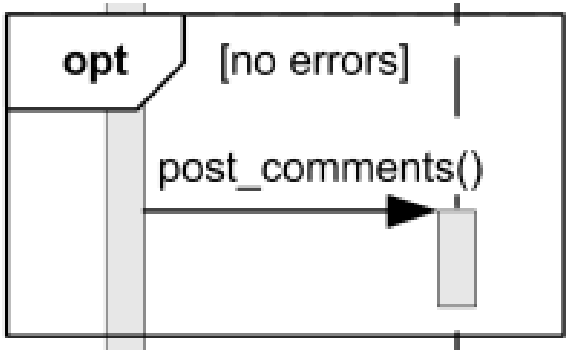
- **Alternatives**

  The interaction operator **alt** means that the combined fragment represents a **choice** or alternatives of behavior.



- **Option**

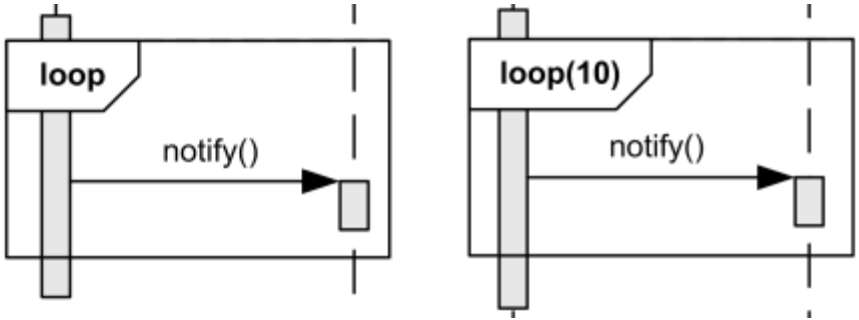
  The interaction operator **opt** means that the combined fragment represents a **choice** of behavior where either the (sole) operand happens or nothing happens.

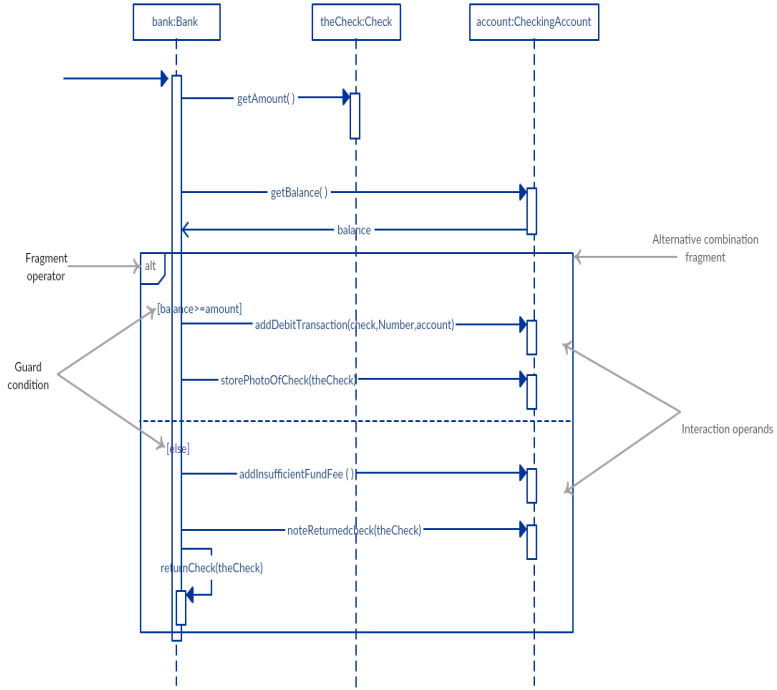


Sequence Diagram Notations

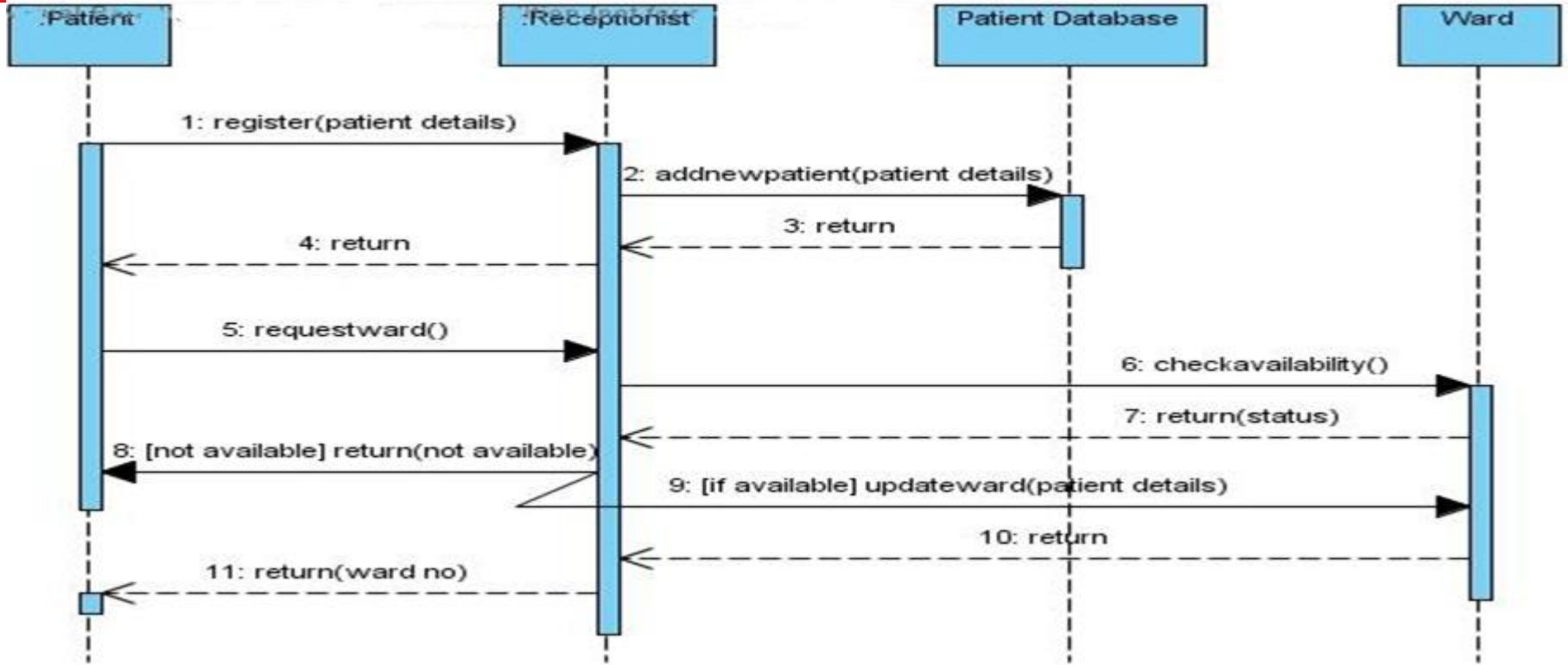
- **Loop**

  The interaction operator **loop** means that the combined fragment represents a loop. The loop operand will be repeated a number of times.





Example :Sequence Diagram for Patient Admit / Registration



Collaboration Diagram

- The collaboration diagram is used to show the relationship between the objects in a system. Both the sequence and the collaboration diagrams represent the same information but differently.
- An object consists of several features. Multiple objects present in the system are connected to each other. The collaboration diagram, which is also known as a communication diagram, is used to portray the object's architecture in the system.

Notations of a Collaboration Diagram

- **Objects:** The representation of an object is done by an object symbol with its name and class underlined, separated by a colon.

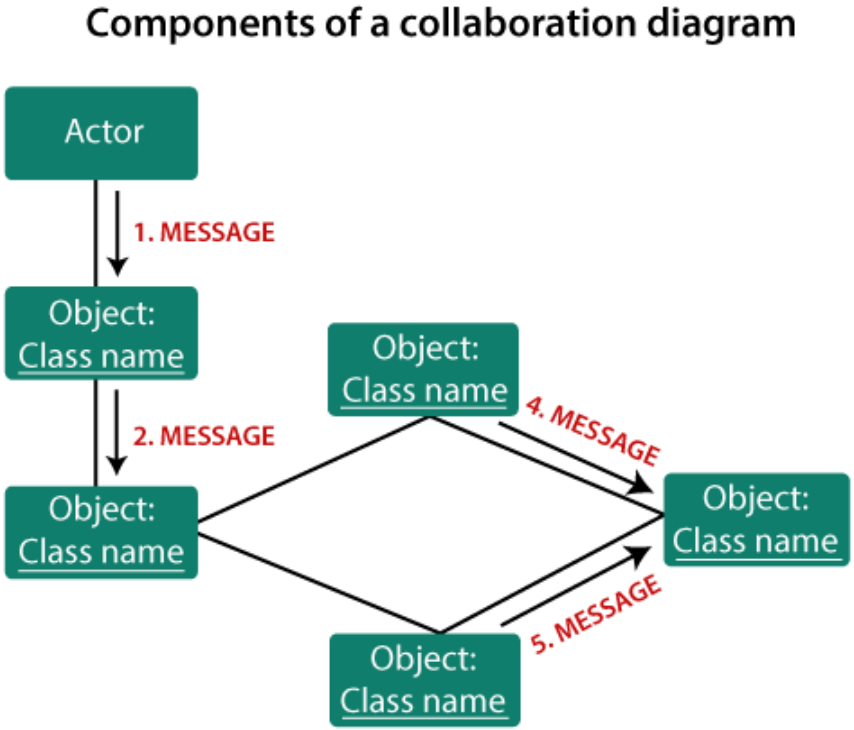
  In the collaboration diagram, objects are utilized in the following ways:

  - The object is represented by specifying their name and class.
  - It is not mandatory for every class to appear.
  - A class may constitute more than one object.
  - In the collaboration diagram, firstly, the object is created, and then its class is specified.
  - To differentiate one object from another object, it is necessary to name them.

Collaboration Diagram

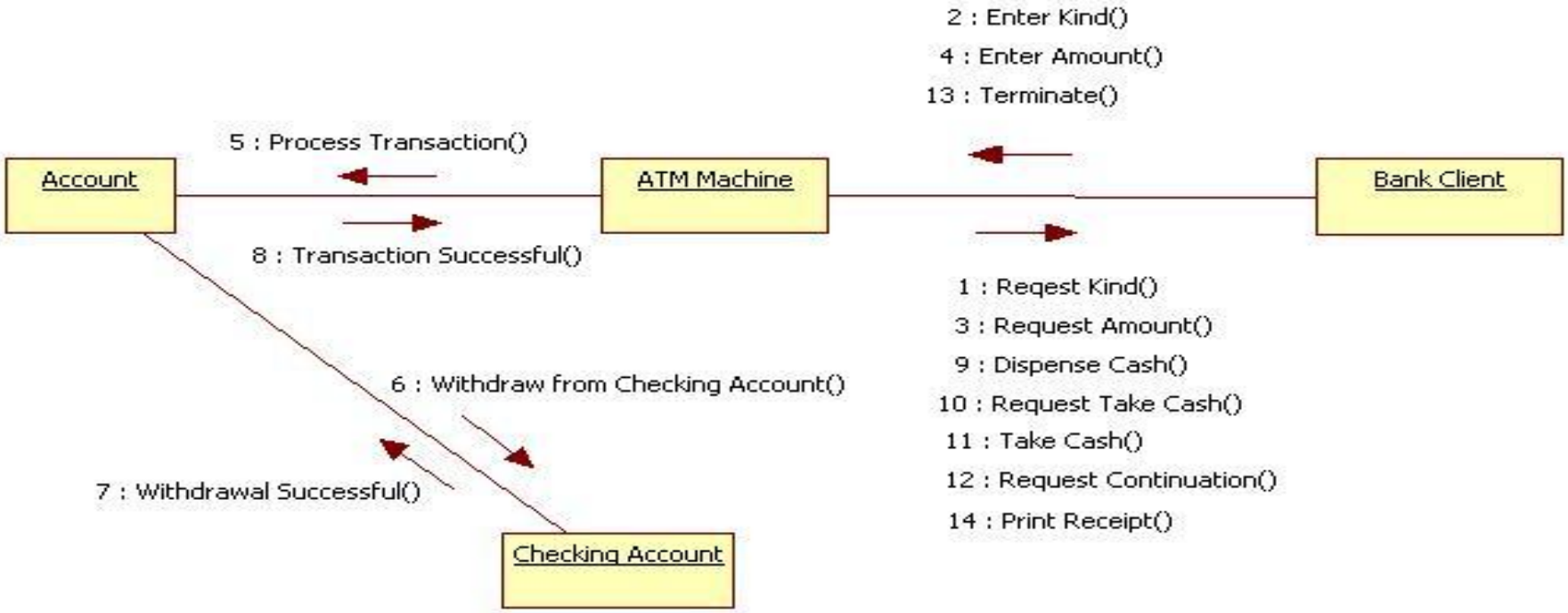
- **Actors:** In the collaboration diagram, the actor plays the main role as it invokes the interaction. Each actor has its respective role and name. In this, one actor initiates the use case.
- **Links:** The link is an instance of association, which associates the objects and actors. It portrays a relationship between the objects through which the messages are sent. It is represented by a solid line. The link helps an object to connect with or navigate to another object, such that the message flows are attached to links.
- **Messages:** It is a communication between objects which carries information and includes a sequence number, so that the activity may take place. It is represented by a labeled arrow, which is placed near a link. The messages are sent from the sender to the receiver, and the direction must be navigable in that particular direction. The receiver must understand the message.

Collaboration Diagram



- **Purpose of Interaction Diagrams**
- To capture the dynamic behaviour of a system.
- To describe the message flow in the system.
- To describe the structural organization of the objects.
- To describe the interaction among objects.

Example : ATM


2 : Enter Kind()
4 : Enter Amount()
13 : Terminate()
5 : Process Transaction()
Account
ATM Machine
Bank Client
8 : Transaction Successful()
1 : Reqest Kind()
3 : Request Amount()
9 : Dispense Cash()
10 : Request Take Cash()
11 : Take Cash()
12 : Request Continuation()
14 : Print Receipt()
6 : Withdraw from Checking Account()
7 : Withdrawal Successful()
Checking Account

Activity Diagram

- Activity diagrams and use cases are logical model which describe the business domain's activities without suggesting how they are conduct.
- Shows the sequence of steps that make up a complex process.
- Shows flow of control, similarly sequence diagram but focus on operations.
- A diagram that emphasizes the flow of control from activity to activity in an object.
- Similar to the traditional program flowchart.
- Used to provide detail for complex algorithms.
- Primary activities and the relationships among the activities in a process.

Activity Diagram

Purpose

- To model a task (for example in business modelling)
- To describe a function of a system represented by a use case
- To describe the logic of an operation
- To model the activities that make up the life cycle in the Unified Process
- 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.

- **Activity Diagram Notations**

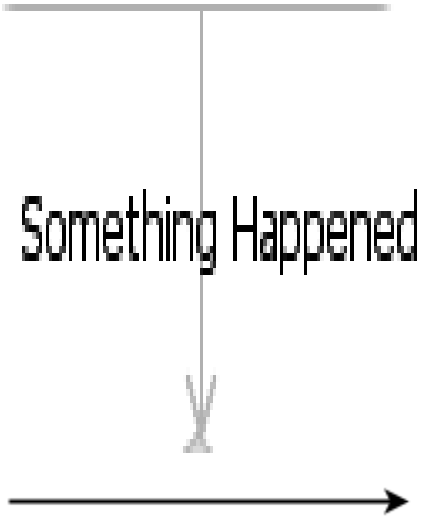
Initial Node

- This represents the start of the flow of an activity diagram.
- An activity diagram contains a single start node.
- The name of the initial node is entered on the node. It takes the form of an adjective.

Activity Diagram Notations

Control Flow

- A control flow connects any combination of:
  - activities
  - branches
  - merges
  - forks
  - joins



- A control flow has direction, which is indicated by the arrow head – you may only traverse the control flow in the direction of the arrow.
- A control flow may not enter an initial state.
- A control flow may not exit a final node.
- A control flow is the representation of an occurrence of an event.

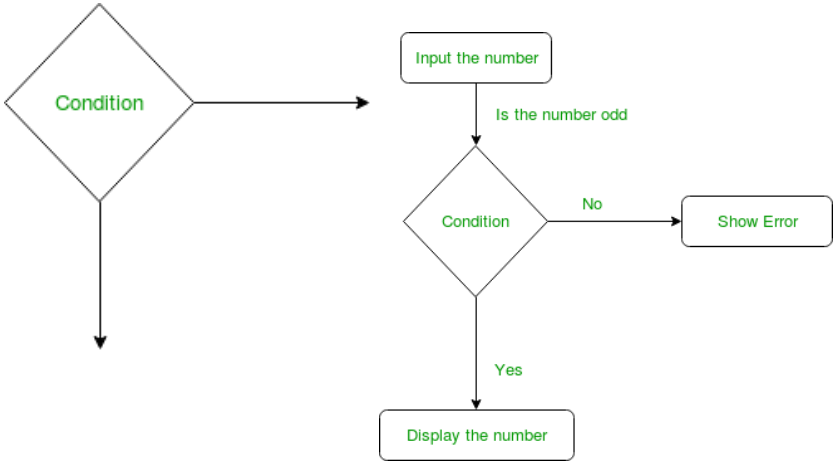
Activity Diagram Notations

Activity or Action

An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.

Decision node and Branching

When we need to make a decision before deciding the flow of control, we use the decision node.

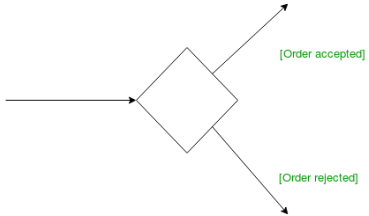


Activity Diagram Notations

Guards

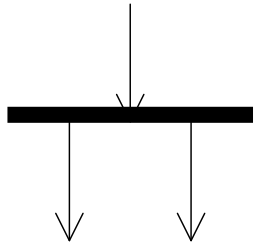
A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets

The statement must be true for the control to shift along a particular direction. Guards help us know the constraints and conditions which determine the flow of a process.



Fork

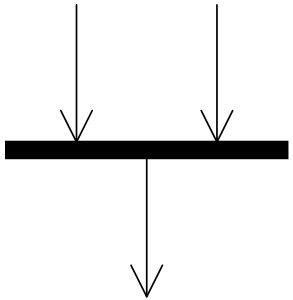
- The fork may be represented by vertical or horizontal bars.
- The fork represents that the flow through the diagram has split into 2 paths that are running in parallel (multitasking).
- The fork has a single control flow on entry and several control flows exiting.



Activity Diagram Notations

Join

- For every fork there should be a join (if not your activity diagram is broken).
- The join may be represented by vertical or horizontal bars.
- A join simply shows that when the parallel activities have finished that they then come back to join a single flow again.
- The join has several control flows entering and a single control flow on exit.
- The exiting control flow cannot be executed until every incoming control flow has completed.
- There is no need to label the fork or join.

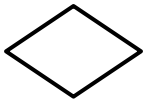


Final State

The state which the system reaches when a particular process or activity ends is known as a Final State or End State. We use a filled circle within a circle notation to represent the final state in a state machine diagram

Activity Diagram Notations

START POINT



DECISION POINT

END POINT

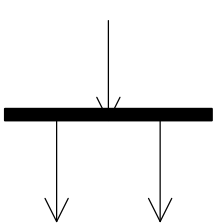
[Condition]

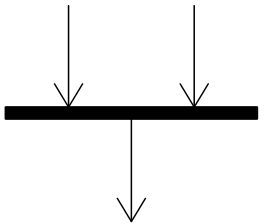
GUARD

STEP

PARALLEL STEPS

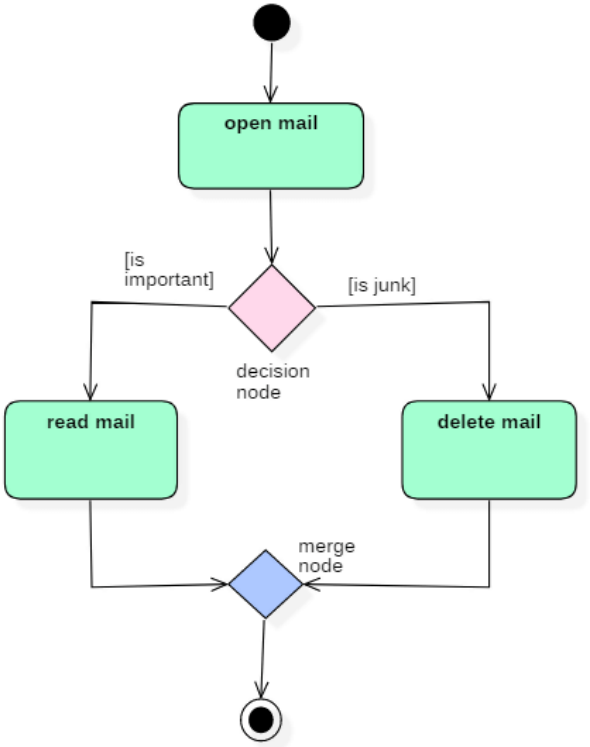
TRANSITION

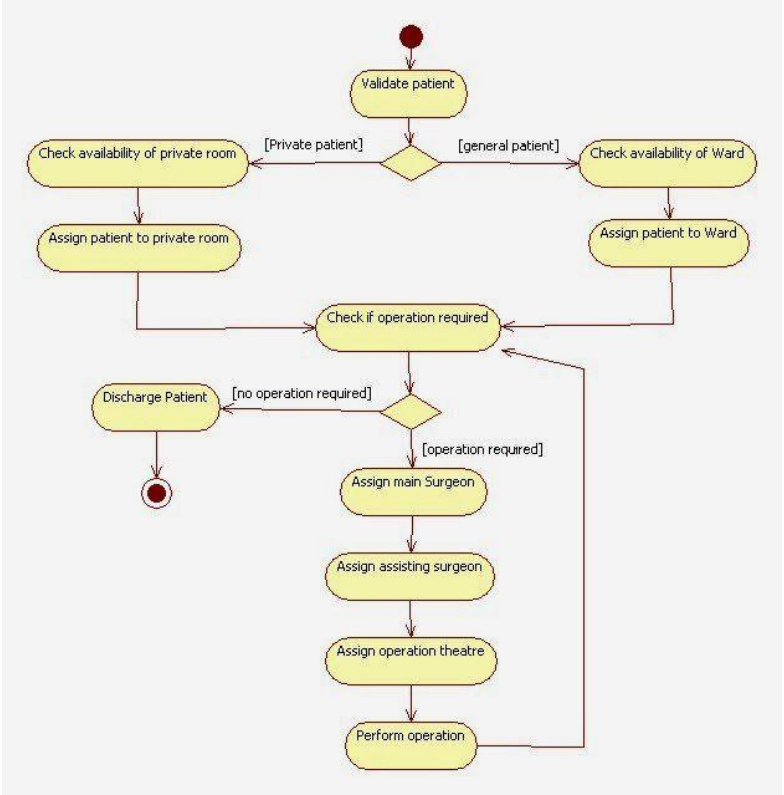




FORK AND JOIN

Example of Activity Diagram





State Diagram/State Machine/State chart

- State model describes the sequences of operations that occur in response to external stimuli.
- The state model consists of multiple state diagrams, one for each class with temporal behavior that is important to an application.
- The state diagram is a standard computer science concept that relates events and states.
- Events represent external stimuli and states represent values objects.

- The basic elements of state diagrams are
  - Initial state – We use a black filled circle represent the initial state of a System or a class.
  - Events – An event is an occurrence at a point in time
  - states – the state in which the object finds itself at any moment
  - transitions – take the object from one state to another
  - actions – take place as a result of a transition
  - Final state – We use a filled circle within a circle notation to represent the final state in a state machine diagram.

State chart Diagram

- **Events**
- An event is an occurrence at a point in time such as –

  User presses left button

  Indigo flight departs from Mumbai

- An event happens instantaneously with regard to time scale of an application.

- **Types of Events**
  - Signal event
  - Time event
  - Change event

State chart Diagram

Signal Event

- A signal is an explicit one-way transmission of information from one object to another.
- An object sending a signal to another object may expect a reply, but the reply is a separate signal under the control of the second object, which may or may not choose to send it.

Time Event

- Time event is an event caused by the occurrence of an absolute time or the elapse of a time interval.
- UML notation for an absolute time is the keyword when followed by a parenthesized expression involving time.
- when (date = Aug 1 , 2015)

Change Event

A change event occurs whenever a specified condition is met

- Event name is specified as keyword *when*
- Parameter list is a *Boolean expression*

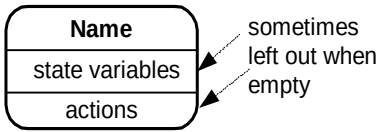
when (battery power < lower limit)

when (tire pressure < minimum pressure)

State chart Diagram

States

- State is a condition or situation during the life of an object within which it performs some activity, or waits for some events
- The objects in a class have a finite number of possible states.
- Each object can be in one state at a time.
- A state specifies the response of an object to input events.
- At any given point in time, the system is in one state.
- It will remain in this state until an event occurs that causes it to change state.

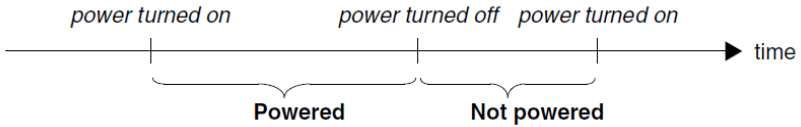


- A state is when a system is:

Doing something – e.g., heating oven, mixing ingredients, accelerating engine,

Waiting for something to happen – Waiting for user to enter password, waiting for sensor reading

Event vs. state



Events represent points in time; states represent intervals of time.

Various characterizations of a state. A state specifies the response of an object to input events.

State: *AlarmRinging*

Description: alarm on watch is ringing to indicate target time

Event sequence that produces the state:

setAlarm (*targetTime*)
any sequence not including *clearAlarm*
when (*currentTime* = *targetTime*)

Condition that characterizes the state:

alarm = on, alarm set to *targetTime*, *targetTime* ≤ *currentTime* ≤ *targetTime* + 20 seconds, and no button has been pushed since *targetTime*

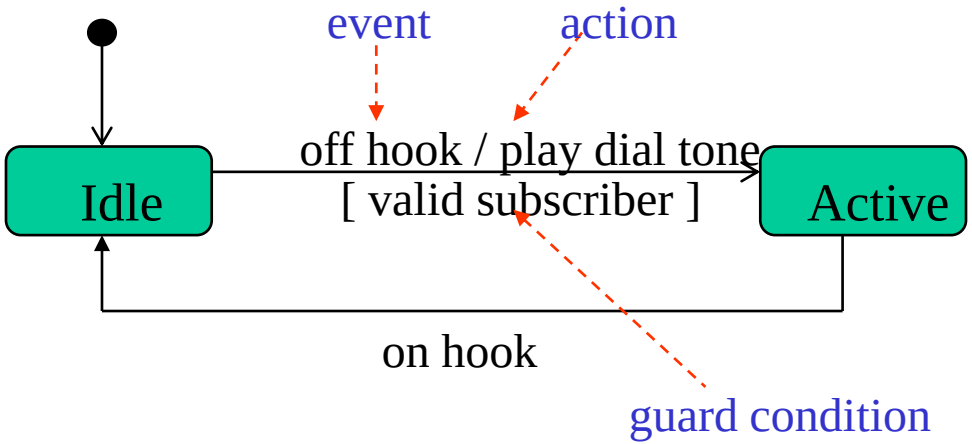
Events accepted in the state:

event	response	next state
when (*currentTime* = *targetTime* + 20)	*resetAlarm*	*normal*
buttonPushed (any button)	*resetAlarm*	*normal*

Transitions

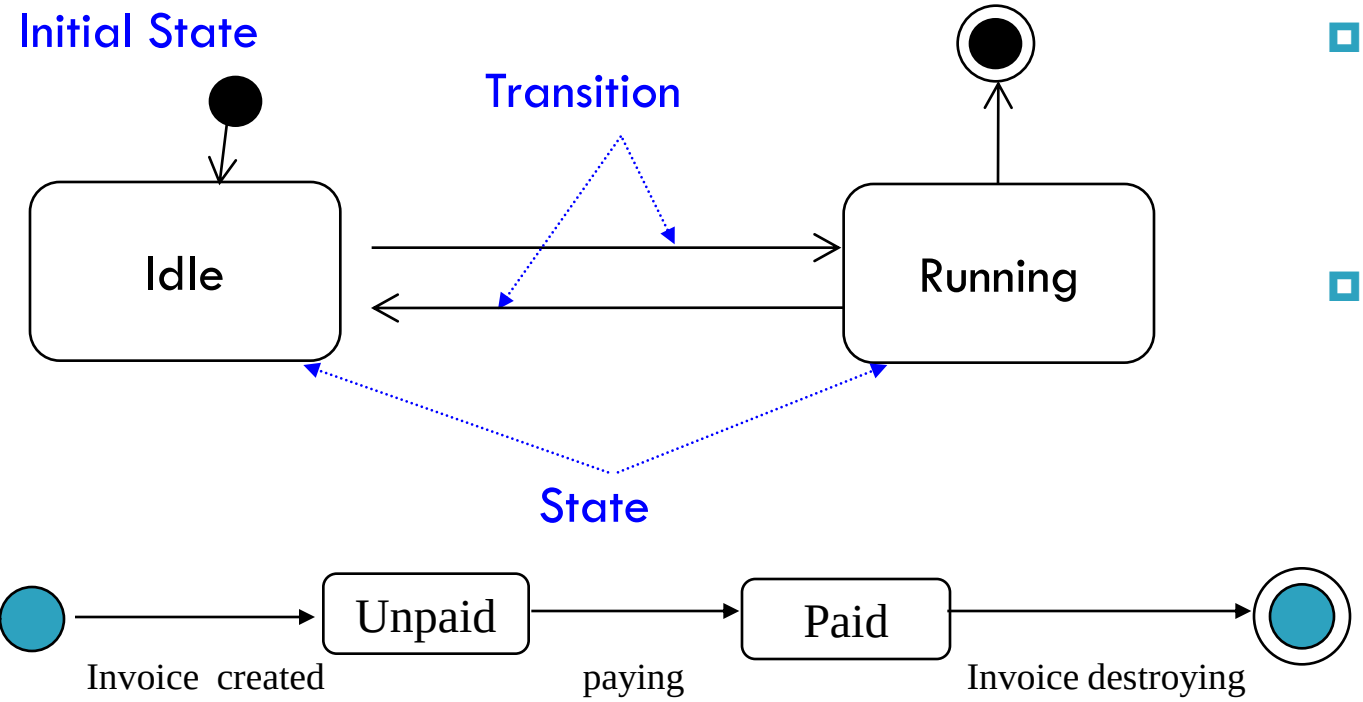
- A transition is a relationship between two states indicating that an object in the first state will enter the second state.
- A transition is an instantaneous change from one state to another.
- The transition is said to fire upon the change from the source state to target state.
- A guard condition must be true in order for a transition to occur.
- A guard condition is checked only once, at the time the event occurs, and the transition fires if the condition is true.

- A directed relationship between two states.
  - Source state - current state before transition fires.
  - Event trigger - external stimulus that has the potential to cause a transition to fire.
  - Guard condition - a condition that must be satisfied before a transition can fire.
  - Target state - new state after transition fires.



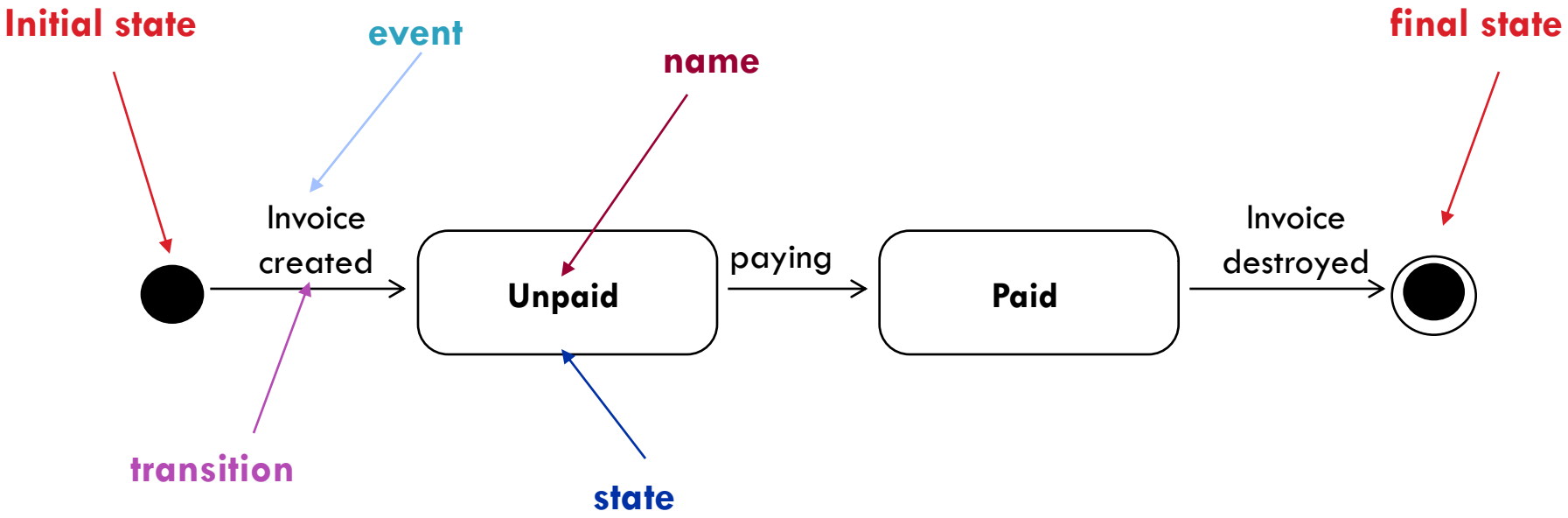
- A transition is drawn with an arrow, possibly labeled with
  - event causing the transaction
  - guard condition
  - Action to perform

AnEvent [guard] / SomeAction

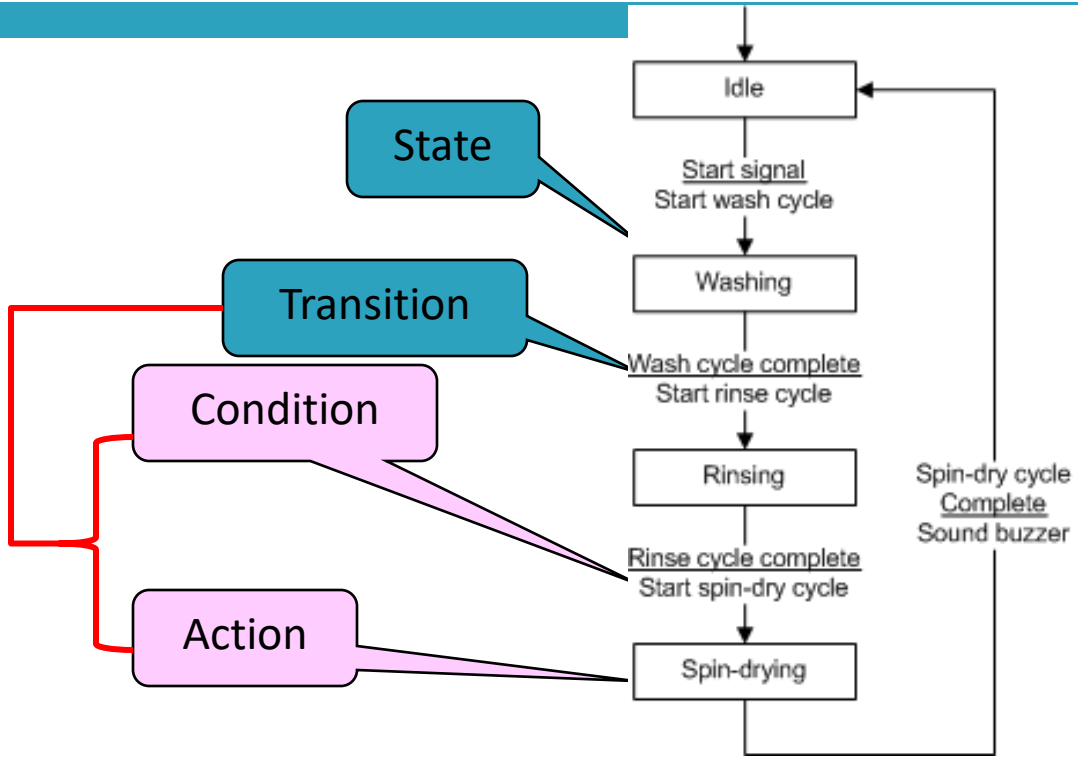


- **Actions**
  - is an executable atomic computation
  - includes operation calls, the creation or destruction of another object, or the sending of a signal to an object

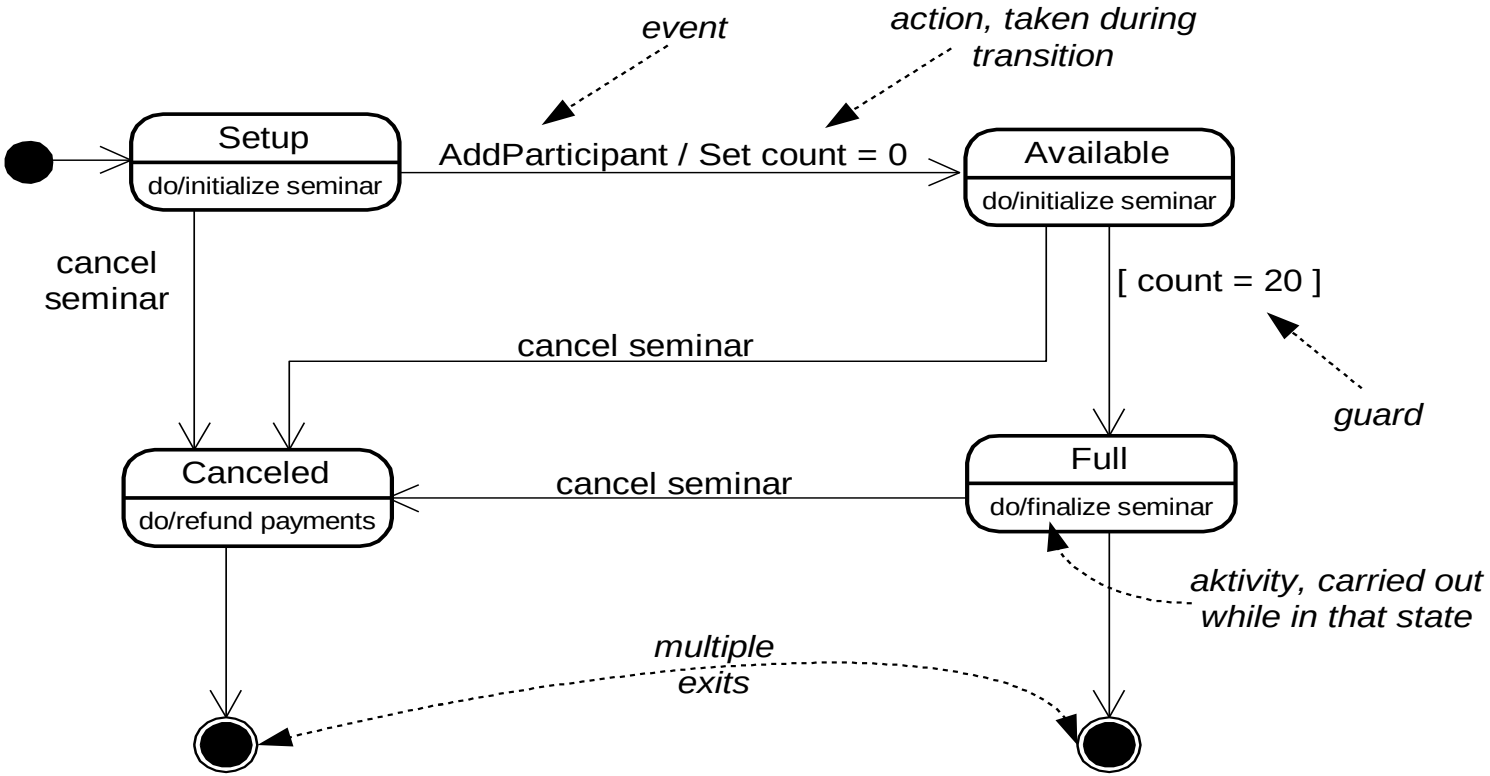
State Diagrams notation



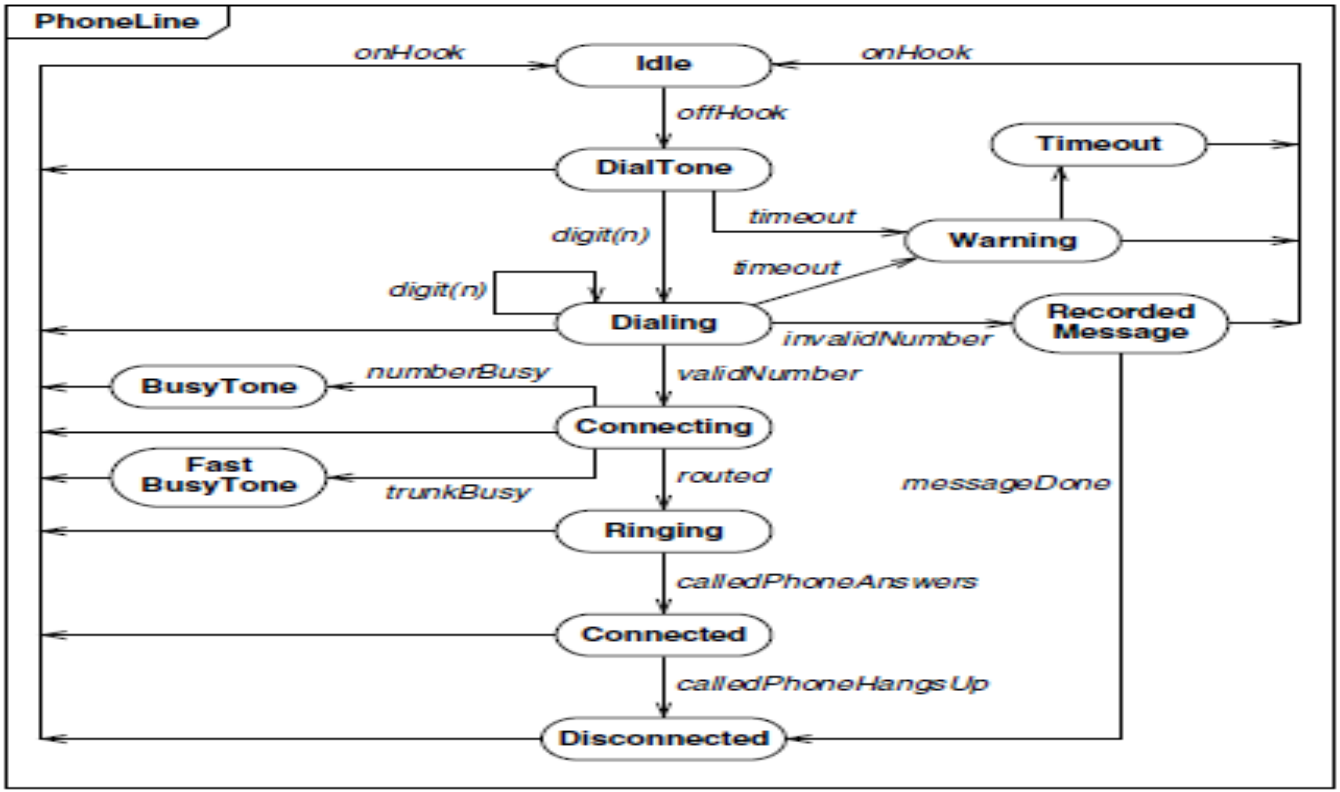
Example: Here's a **simple** example SD for a washing machine



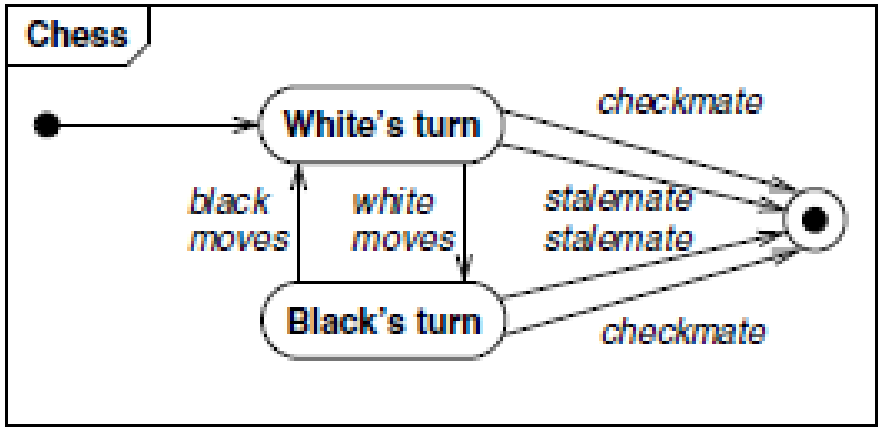
Seminar Registration

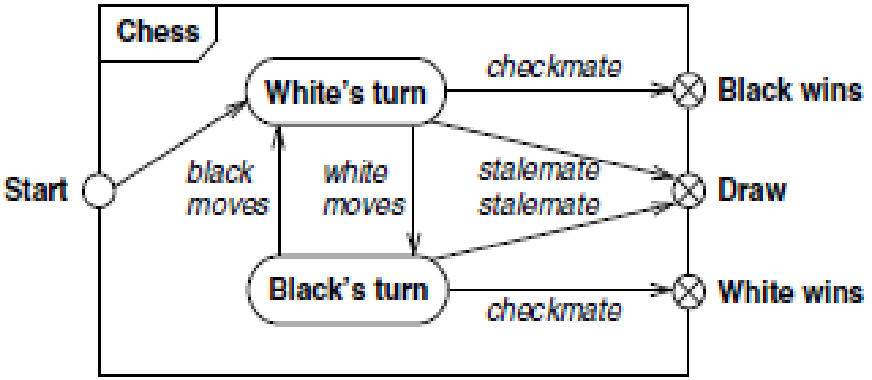


State diagram for a telephone line

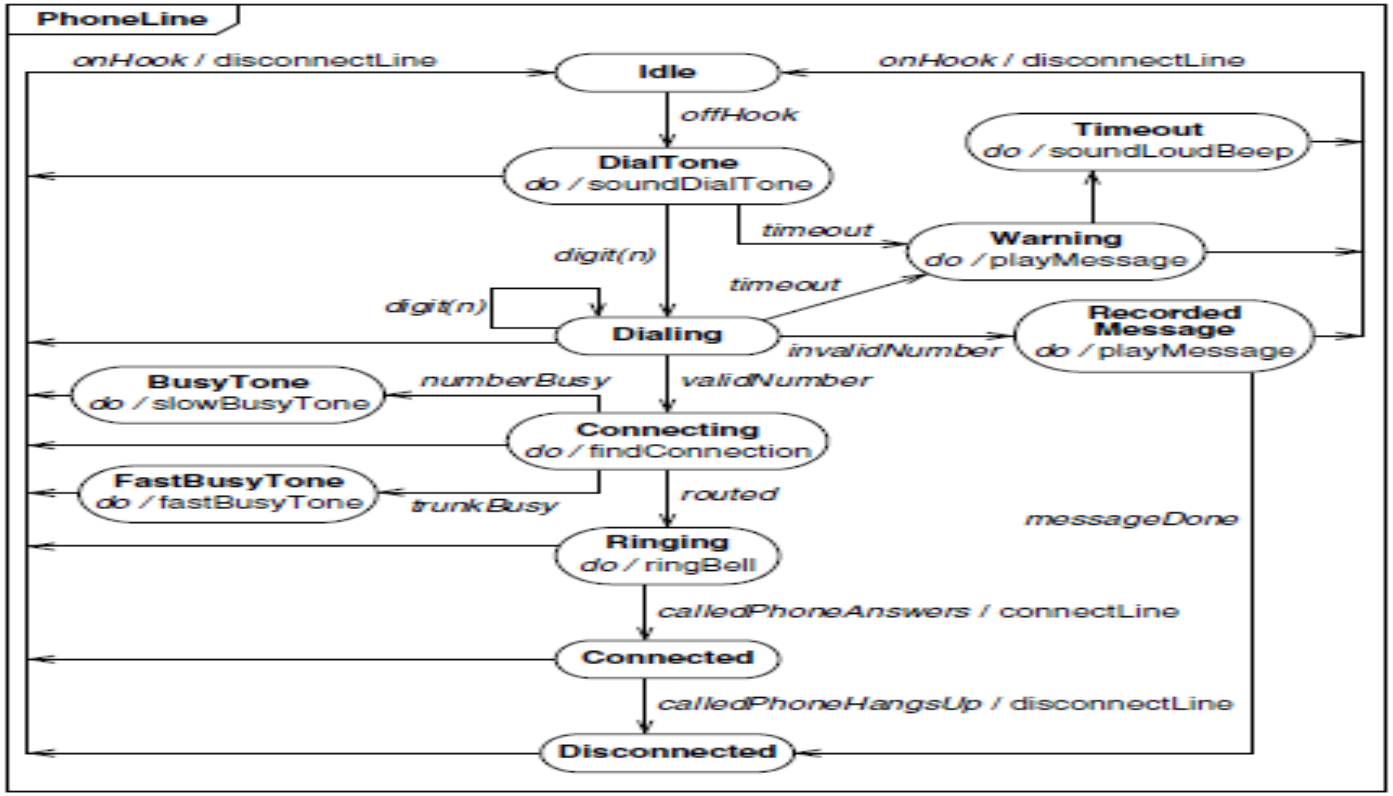


State diagram for chess game



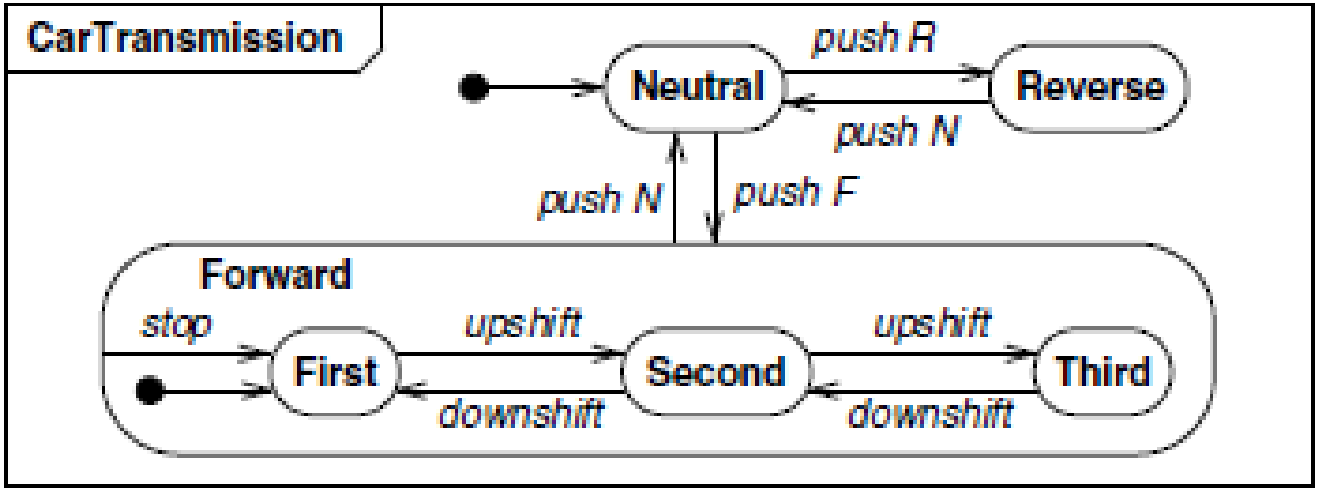


State diagram for phone line with activities



Nested State Diagrams

- You can structure states more deeply than just replacing a state with a submachine.




THANK YOU!
ANY QUESTIONS?