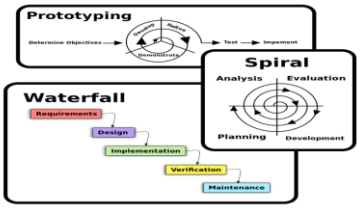
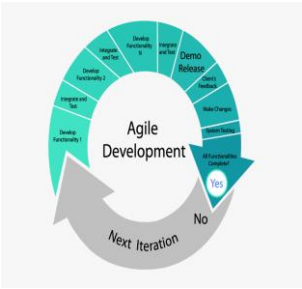
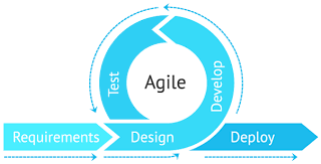


Software Engineering Understanding Requirements



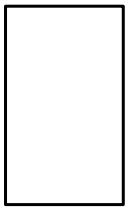


Use Case Diagram

- A use case diagram is a representation of a user's interaction with the system.
- This interaction shows the relationship between the user and the different use cases in which the user is involved.
- The purpose of the use case diagrams is simply to provide a high-level view of the system and convey the requirements in layman's terms for the stakeholders.

Components of Use Case Diagram

System boundary



- Represent the scope of the system
- Use cases of the system are placed inside the system boundary
- Actors who interact with the system are placed outside the system

Actor

- An actor is an entity that interacts directly with the system but that is not part of system
- Actor may be people, computer hardware, other systems, etc.

3

Components of Use Case Diagram

Use case

- A use case represents a user goal / piece of functionality that can be achieved by accessing the system or software application.

Association

- An actor and use case can be associated to indicate that the actor participates in that use case

Generalization

- A generalization relationship is used to represent the inheritance relationship between model elements of the same type

4

Components of Use Case Diagram

Include <<include>>

- An include relationship is a relationship in which one use case includes the functionality of another use case
- The include relationship supports the reuse of functionality in a use-case model.

Extends <<Extends>>

- The extend relationship specifies that the incorporation of the extension use case is dependent on what happens when the base use case executes.

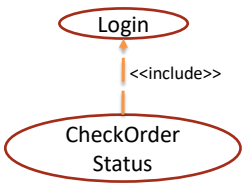
Constraint {condition}

- Show condition exists between actors and activity

5

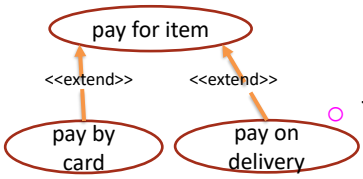
Example of Include and Extends

Include <<include>>



- In e-commerce application that provides customers with the option of checking the status of their orders. For checking the status of their order user should be login.
- This behavior is modeled with a base use case called CheckOrderStatus that has an inclusion use case called LogIn.

Extends <<Extends>>



- In e-commerce site, When paying for an item, you may choose to pay on delivery, pay using PayPal, or pay by card.
- These are all alternatives to the "pay for item" use case. I may choose any of these options depending on my preference.

6

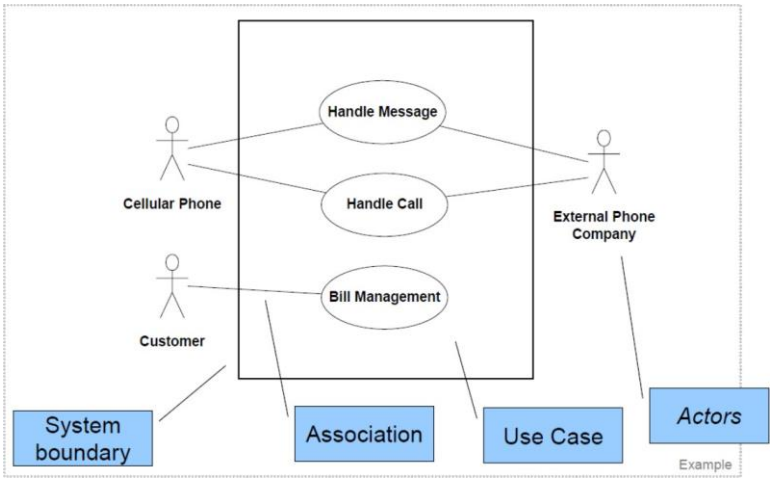
Guideline for constructing use case diagram

- Determine the system boundary.
- Ensure that actors are focused, each actor should have a single, coherent purpose. If a real world object contains multiple purpose, capture them with separate actors
- Each use case must provide value of users
- Relate use cases and actors

7

Example Use-Case Diagram

- A standard form of use case diagram is defined in the Unified Modeling Language



8

Library Management System(LMS) formal Requirement

- A Library Management System is a software built to handle the primary housekeeping functions of a library.
- In library management systems to **manage asset collections** as well as relationships with their members.
- Library management systems help libraries keep track of the books and their checkouts, as well as members' subscriptions and profiles.
- Library management systems also involve maintaining the database for entering new articles and recording articles that have been borrowed with their respective due dates.

9

Identify the Functionality & Stakeholders for LMS

Functionality

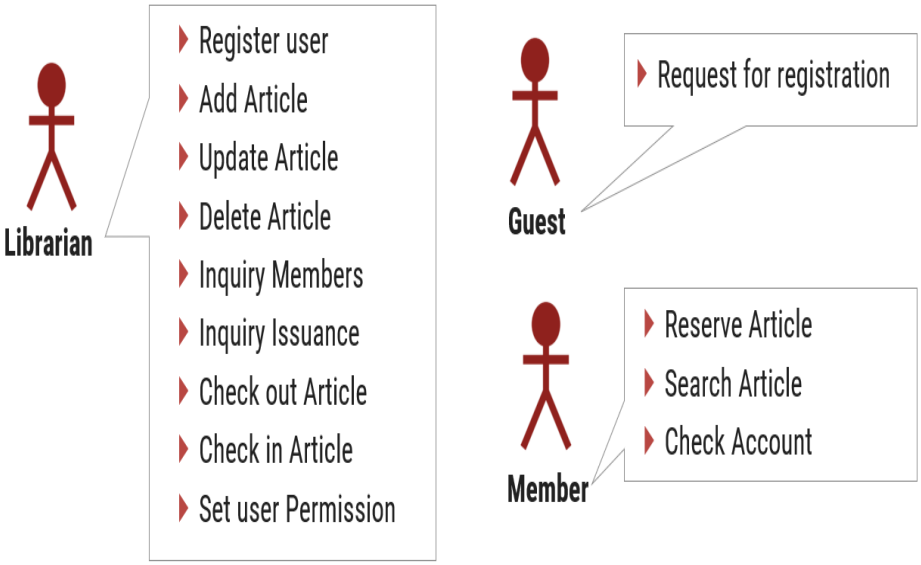
- Register User
- Add Article
- Update Article
- Delete Article
- Inquiry Members
- Inquiry Issuance
- Check out Article
- Check in Article
- Reserve Article
- Set user Permission
- Search Article
- Check Account
- Prepare Library Database

Stakeholders

- Librarian
- Member
- Guest

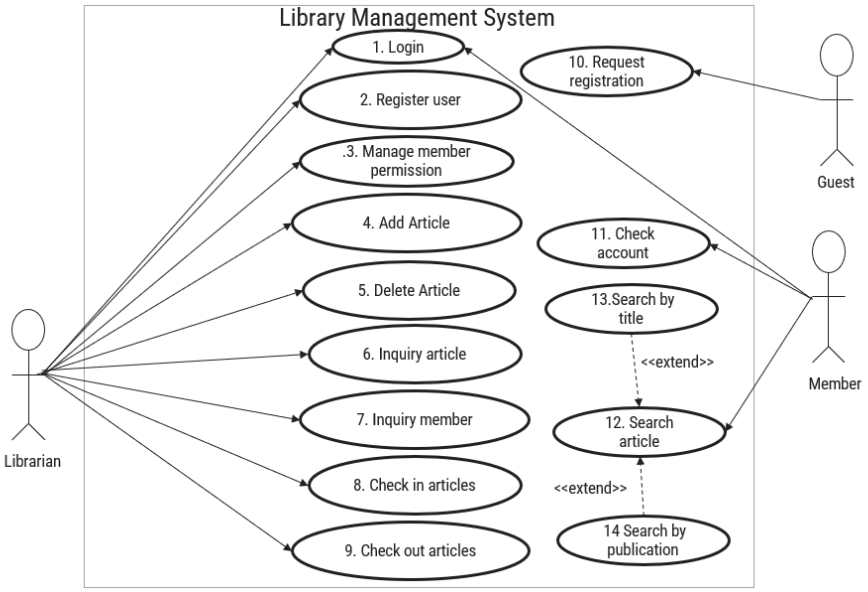
10

Relationship Between Functionality & Stakeholders



11

Use Case Diagram Library Management



12

The Elevator Problem Case Study

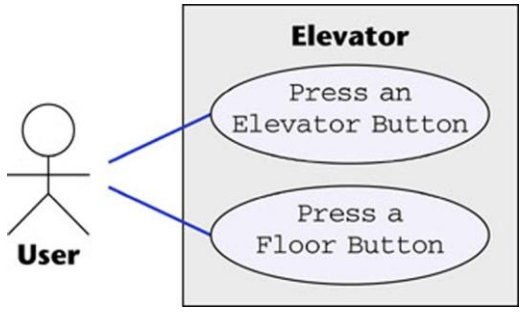
- A product is to be installed to control n elevators in a building with m floors. The problem concerns the logic required to move elevators between floors according to the following constraints:

1. Each elevator has a set of m buttons, one for each floor. These illuminate when pressed and cause the elevator to visit the corresponding floor. The illumination is cancelled when the corresponding floor is visited by the elevator
2. Each floor, except the first and the top floor, has two buttons, one to request an up-elevator, one to request a down-elevator. These buttons illuminate when pressed. The illumination is cancelled when an elevator visits the floor, then moves in the desired direction
3. If an elevator has no requests, it remains at its current floor with its doors closed

13

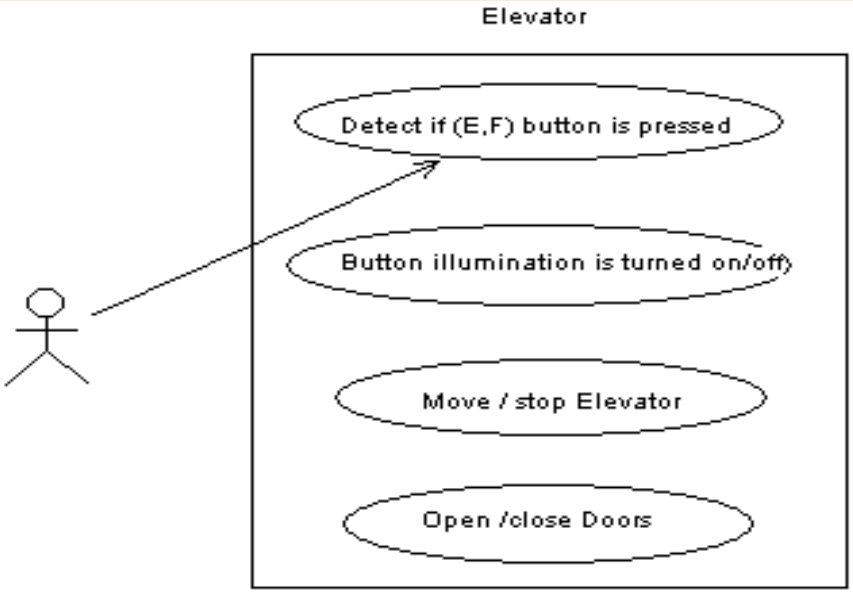
Use Cases

- For the elevator problem, there are only two possible use cases
  - Press an Elevator Button, and
  - Press a Floor Button



14

Use Cases



15

Scenarios

- A use case provides a generic description of the overall functionality. A scenario is an instance of a use case
- Sufficient scenarios need to be studied to get a comprehensive insight into the target product being modelled
- Each use case is one or more scenarios.
- Add Subject Use Case :
  - Scenario 1 : Subject gets added successfully.
  - Scenario 2 : Adding the subject fails since the subject is already in the database.
- Enroll Subject Use Case:
  - Scenario 1 : Student is enrolled for the subject.
  - Scenario 2 : Enrollment fails since the student is already enrolled in the subject.
- Each scenario has a sequence of steps.

16

Scenarios

- Each scenario has a sequence of steps.
- Scenario 1 : Student is enrolled for the subject.
  - Student chooses the “enroll subject” action.
  - Check the student has enrolled in less than 10 subjects.
  - Check if the subject is valid.
  - Assign the subject to the student.
- Scenario 2 : Enrolling fails since the student is already enrolled in 10 subjects.
  - Student chooses the “enroll subject” action.
  - Check the student has enrolled in less than 10 subjects.
  - Return an error message to the student.

17

Normal Scenario: Elevator Problem

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 7.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The Up floor button is turned off.
5. The elevator doors open.
6. The timer starts.
   User A enters the elevator.
7. User A presses the elevator button for floor 7.
8. The elevator button for floor 7 is turned on.
9. The elevator doors close after a timeout.
10. The elevator travels to floor 7.
11. The elevator button for floor 7 is turned off.
12. The elevator doors open to allow User A to exit from the elevator.
13. The timer starts.
    User A exits from the elevator.
14. The elevator doors close after a timeout.
15. The elevator proceeds to floor 9 with User B.

18

Exception Scenario: Elevator Problem

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 1.
2. The Up floor button is turned on.
3. An elevator arrives at floor 3. It contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
4. The Up floor button is turned off.
5. The elevator doors open.
6. The timer starts.
   User A enters the elevator.
7. User A presses the elevator button for floor 1.
8. The elevator button for floor 1 is turned on.
9. The elevator doors close after a timeout.
10. The elevator travels to floor 9.
11. The elevator button for floor 9 is turned off.
12. The elevator doors open to allow User B to exit from the elevator.
13. The timer starts.
    User B exits from the elevator.
14. The elevator doors close after a timeout.
15. The elevator proceeds to floor 1 with User A.

Use cases & Usage Scenarios

- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an "actor"
- An actor is a person or device that interacts with the software
- Each scenario answers the following questions
  - Who is the primary actor, the secondary actor (s)?
  - What are the actor's goals?
  - What preconditions should exist before the story begins?
  - What main tasks or functions are performed by the actor?
  - What extensions might be considered as the story is described?
  - What variations in the actor's interaction are possible?
  - What information does the actor desire from the system?

20

Use cases & Usage Scenarios

- What system information will the actor acquire, produce, or change?
- Will the actor have to inform the system about changes in the external environment?
- Does the actor wish to be informed about unexpected changes?

- Scenarios are created by user researchers to help communicate with the design team.
- User stories are created by project/product managers to define the requirements prior to a sprint in agile development.
- Scenarios are stories that capture the goals, motivations, and tasks of a persona in a given system.

21

Building the requirement model

- The intent of the analysis model is to provide a description of the required informational, functional, and behavioral domains for a computer-based system.
- The model changes dynamically as you learn more about the system to be built, and other stakeholders understand more about what they really require.
- For that reason, the analysis model is a snapshot of requirements at any given time. You should expect it to change.
- As the requirements model evolves, certain elements will become relatively stable, providing a solid foundation for the design tasks that follow.
- However, other elements of the model may be more volatile, indicating that stakeholders do not yet fully understand requirements for the system.

22

Activity Diagram

- An activity diagram visually presents a series of operation or flow of control in a system similar to algorithm or a flowchart.
- An activity diagram is like a traditional flowchart in that it show the flow of control from step to step.
- An activity diagram can show both sequential and concurrent flow of control.
- Activity diagram mainly focus on the sequence of operation rather than on objects.
- Activity diagram represent the dynamic behavior of the system or part of the system.
- An activity diagram shows 'How' system works.
- Activity diagram are most useful during early stages of designing algorithms and workflows.

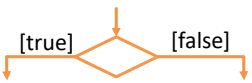
23

Elements of Activity Diagram

Activity

- The main element of an activity diagram is the activity itself.
- An activity is a function/operation performed by the system.
- The elongated ovals show activities.
- An unlabeled arrow from one activity to another activity, that indicates that the first activity must complete before the second activity begin.

Branches



- If there is more than one successor to an activity, each arrow may be labeled with a condition in square brackets. For e.g. *[failure]*
- As a notational convenience, a diamond shows a branch into multiple successors.
- The diamond has one incoming arrows and two or more outgoing arrows. Each with condition.

24

Elements of Activity Diagram

Initiation

- A solid circle with an outgoing arrow shows the starting point of an activity diagram.
- When an activity diagram is activated, control starts at the solid circle and proceeds via the outgoing arrow toward the first activities.

Termination

- A bull's eye – a solid circle surrounded by a hollow circle shows the termination point.
- The symbol only has incoming arrows.
- When control reaches a bull's eye, the overall activity is complete and execution of the activity diagram ends.

25

Elements of Activity Diagram

Concurrent Activities

Merge 

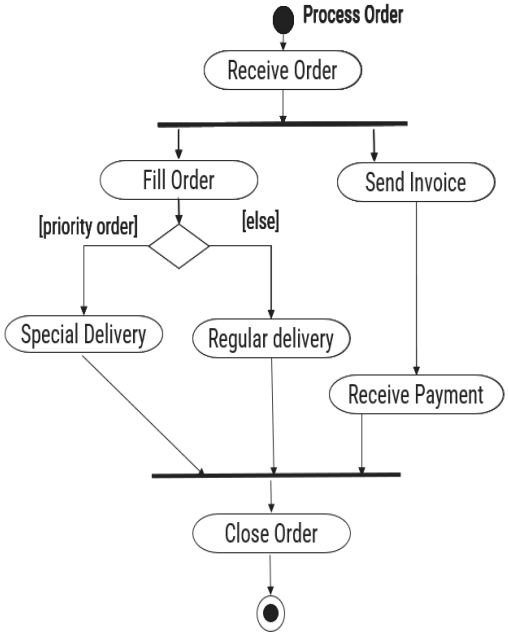
Fork 

- System can perform more than one activity at a time.
- For e.g. one activity may be followed by another activity, then split into several concurrent activities (a fork of control), and finally be combined into a single activity (a merge of control).
- A fork or merge is shown by a synchronization bar –a heavy line with one or more input arrows and one or more output arrows.

26

Example of Fork & Join

- An example of business flow activity of order processing, based on the Example order is input parameter of the activity.
- After order is accepted and all required information is filled in, payment is accepted and order is shipped.
- **Note, that this business flow allows order shipment before invoice is sent or payment is confirmed.**



Guideline for Activity Diagram

- Activity diagram elaborate the details of computation, thus documenting the steps needed to implement an operation or a business process.
- Activity diagram can help developers to understand complex computations by graphically displaying the progression through intermediate execution steps.
- Here is some advice for activity diagram.

Don't misuse activity

- Activity diagrams are intended to elaborate use case and sequence models so that a developer can study algorithms and workflow.
- Activity diagrams supplement the object-oriented focus of UML models and should not be used as an excuse to develop software via flowchart.

28

Guideline for Activity Diagram

Level diagrams

- Activities on a diagram should be at a consistent level of details.
- Place additional details for an activity in a separate diagram.

Be careful with branches and conditions

- If there are conditions, at last one must be satisfied when an activity completes, consider using an *[else]* condition.
- It is possible for multiple conditions to be satisfied otherwise this is an error condition.

Be careful with concurrent activities

- Means that the activities can complete in any order and still yield an acceptable result.
- Before a merge can happen, all inputs must first complete

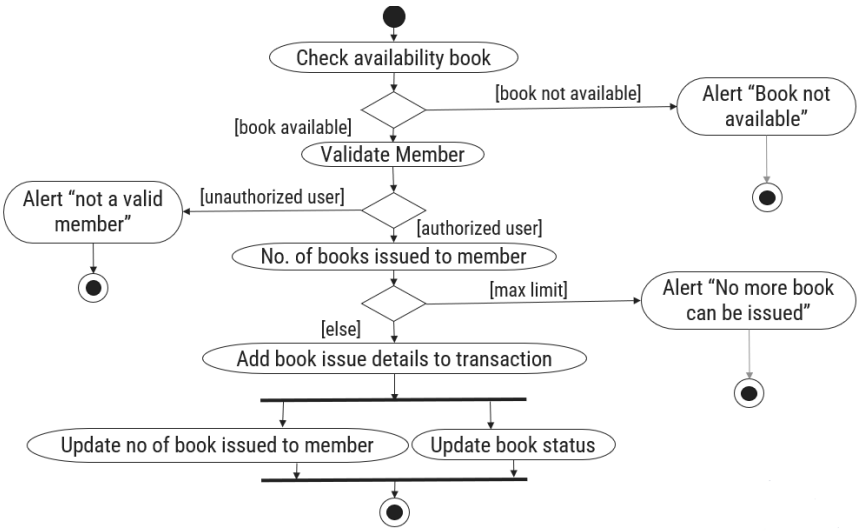
29

How to Draw an Activity Diagram

- **Step 1**: Identify the various activities and actions your business process or system
- **Step 2:** Find a flow among the activities
- For e.g. in library management system, book issue is a one business process or a function. Show we prepare a activity diagram for Book issue.
- Various activity in book issue process like…
  - Check availability of book
  - Validate the member
  - Check No. of books issued by member
  - Add book issue details to transaction
  - Update no of book issued by member
  - Update book status.

30

Activity Diagram for Book Issue



31

Swimlane Diagram

- In a business model, it is often useful to know which human department is responsible for an activity.
- When design of the system is complete, the activity will be assigned to a person/department, but at a high level it is sufficient to partition the activities among departments.
- You can show such a partitioning with an activity diagram by dividing in to columns and lines.
- Each column is called swim-lane by analogy to a swimming pool.
- Placing an activity within a particular swim-lane indicates that is performed by a person/ department.
- Lines across swim-lane boundaries indicate interaction among different person/department.

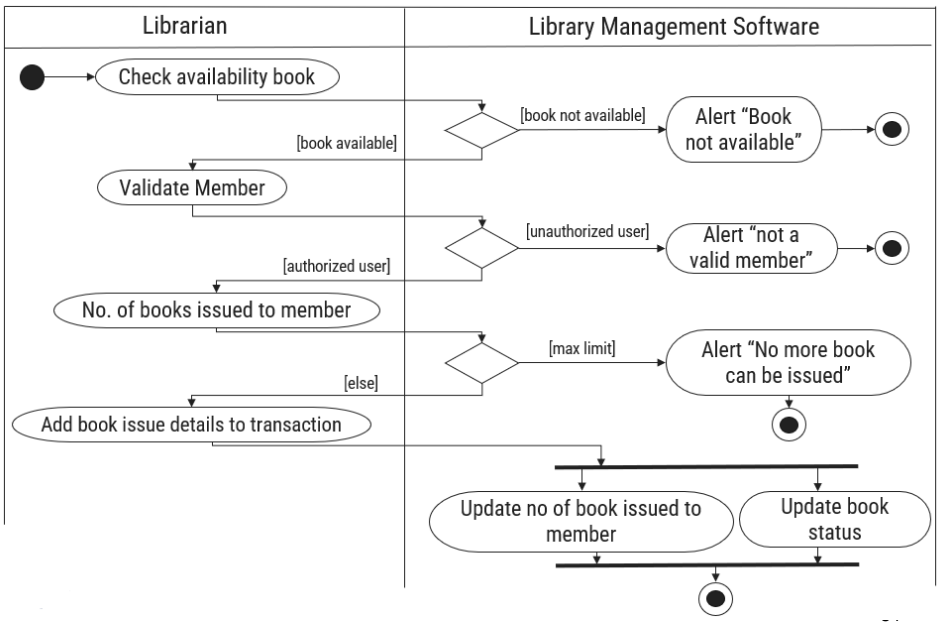
32

How to Draw a Swimlane Diagram

- **Step 1:** Identify the various activities and actions your business process or system
- **Step 2:** Figure out which person/departments are responsible for the competition of activity.
- **Step 3:** Figure out in which order the actions are processed.
- **Step 4:** Figured out who is responsible for each action and assign them a swimlane and group each action they are responsible for under them

33

How to Draw a Swimlane Diagram

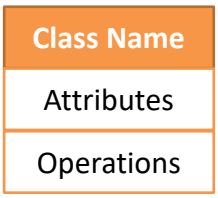


Elements of the Requirements Model

- Class-based elements: Each usage scenario implies a set of objects that are manipulated as an actor interacts with the system.
- These objects are categorized into classes—a collection of things that have similar attributes and common behaviors.
- For example, a UML class diagram can be used to depict
- The **purpose** of class modeling is to describe **objects in systems** and **different types of relationships between them.**
- Class diagrams represent an overview of the system like classes, attributes, operations, and relationships.

35

Elements of the Class Diagram

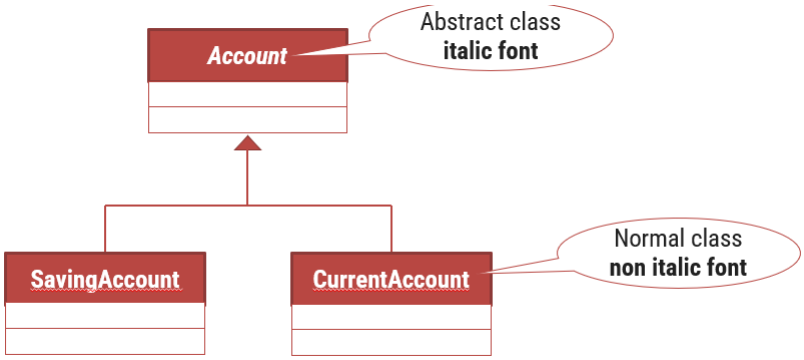


- The name of the class appears in the upper section.
- Class name should be meaningful.
- Class name should always be aligned center of the upper section.
- Class name should start with capital letters, and intermediate letter is a capital.
- Class name should be always bold format.
- For e.g.:

36

Elements of the Class Diagram

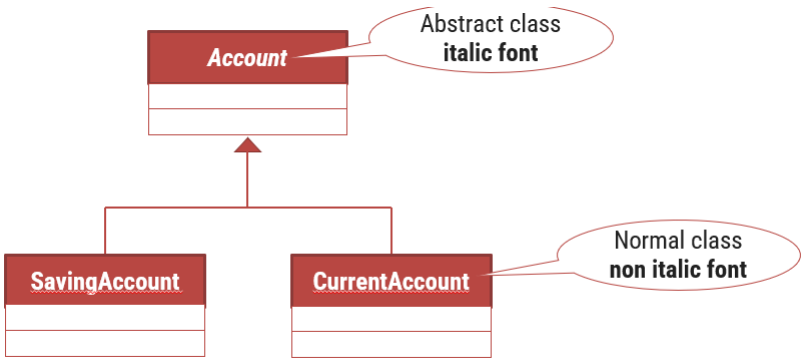
- Abstract class name should be written in italic format.



37

Elements of the Class Diagram

- Abstract class name should be written in italic format.



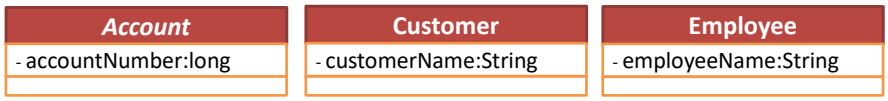
38

Elements of the Class Diagram

An attribute is a named **property of a class** that describes a value held by each object of the class.

The UML notation lists attributes in the second compartment of the class box.

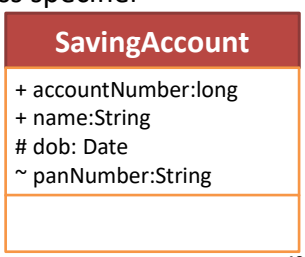
- The attribute name should be in the regular face, left align in the box & use the lowercase letters for the first character.
- The data type for the attribute should be written after the colon.
- **Accessibility** of attribute must be defined using a member access modifier.
- Syntax : accessModifier attributeName:dataType=defaultValue
- For e.g. *in this example '–' represents private access modifier*



39

Elements of the Class Diagram (Access Modifiers)

- Public (+): Member accessible by all classes, whether these classes are in the same package or in another package.
- Private (-): Member cannot be accessed outside the enclosing/declaring class.
- Protected (#): Member can be accessed only by subclasses and within a class.
- Package (~): Member can be accessible by all classes, within the package. Outside package member not accessible.
- In example you can see how to use access specifier



40

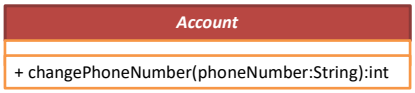
Elements of the Class Diagram

The operation is a function or procedure that may be applied objects in a class.

The UML notation is to list operations in the third compartment of the class box.

- The operation name in the regular face, left align the name in the box, and use a lowercase letter for the first character.
- Optional detail, such as an argument list and result type, may follow each operation name.
- The return type of method should be written after colon.
- **Accessibility** of operation must be defined using a member access modifier.

Syntax : accessModifier methodName(argumentList):returnType



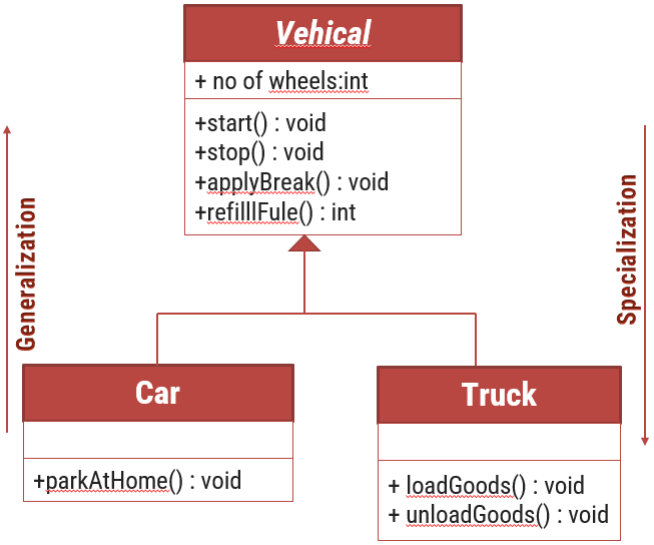
41

Generalization & Specialization

- Generalization is the **process of extracting shared characteristics** from two or more classes and combining them into a generalized superclass
- Shared characteristics can be attributes or methods.
- Represents an "is-a" relationship
- For example, a car is a vehicle and a truck is a vehicle. In this case, vehicle is the general thing, whereas car and truck are the more specific things.
- Specialization is the **reverse process of Generalization** means creating new sub-classes from an existing class.

42

Generalization & Specialization



43

Link and Association Concepts

- Link and associations are the means for **establishing relationships among objects and classes**.
- A link is a physical or conceptual connection among objects.
- An association is a description of a group of links with common structure and common semantic & it is optional.
- **Aggregation** and **Composition** are the two forms of association. It is a subset of association.
- Means they are **specific cases of association**. In both aggregation and composition object of one class "owns" object of another class, but there is a minor difference.

44

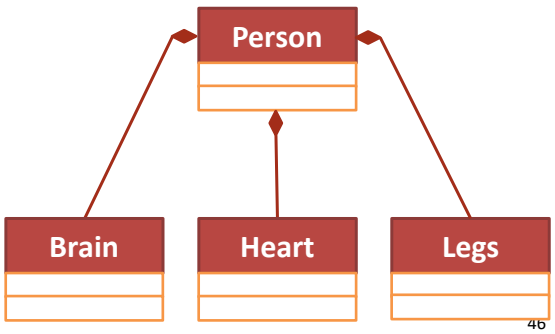
Aggregation

- It represents 'has a' relationship.
- Aggregation implies a relationship where the **child is independent** of its **parent**.
- For e.g.: Here we are considering a car and a wheel example.
  - A car cannot move without a wheel.
  - But the wheel can be independently used with the bike, scooter, cycle, or any other vehicle.
  - The wheel object can exist without the car object, which proves to be an aggregation relationship.

45

Composition

- It represents the **dependency between** a **parent** and its **children**, which means if the parent is discarded then its children will also discard.
- It represents 'part-of' relationship.
- In composition, both the entities are dependent on each other.
- For e.g.: Person class with Brain class, Heart class, and Legs class.
- If the person is destroyed, the brain, heart, and legs will also get discarded.



46

Multiplicity

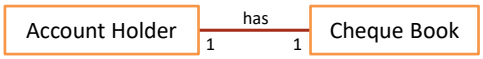
- **Multiplicity** is the specification of the number of instances of one class that may be related to the instance of another class.
- Multiplicity constrains the number of a related object.
- You can use multiple associations between objects.
- Some typical type of multiplicity:

Multiplicity	Option	Cardinality
0..1		No instances or one instance
1..1	1	Exactly one instance
0..*	*	Zero or more instances
1..*		At least one instance
5..5	5	Exactly 5 instances
m..n		At least m but no more than n instances

47

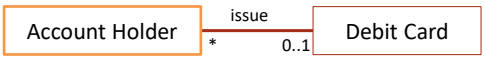
Example Of Multiplicity

One to One Association



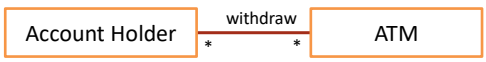
One account holder has one cheque book

Many to Zero or One Association



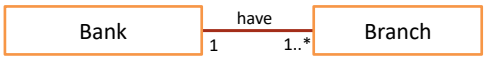
An account holder can issue at most one debit card

Many to Many Association



Every account holder can withdraw money from all ATMs.

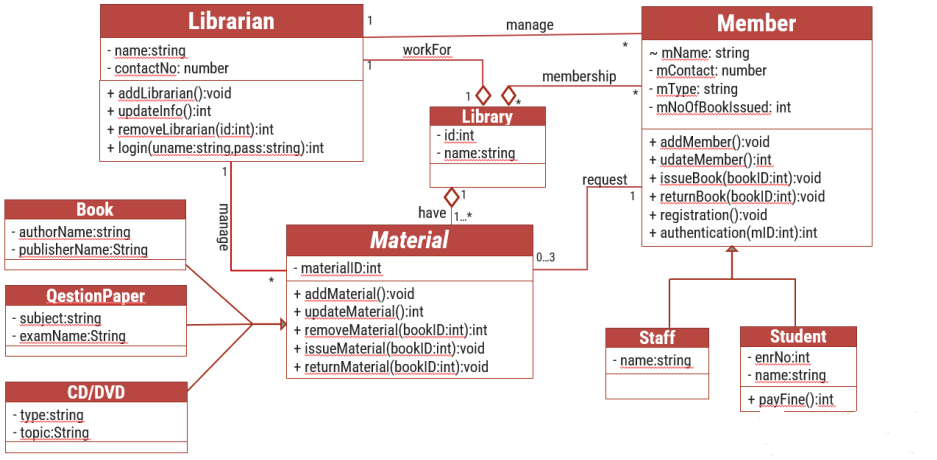
One to One or Many Association



The bank should have at least one branch.

48

Class Diagram Of Library Management System



49

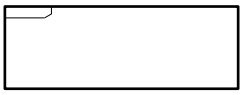
Elements of the Requirements Model

- Behavioral elements: The state diagram is one method for representing the behavior of a system by depicting its states and the events that cause the system to change state.
- A state is any externally observable mode of behavior. In addition, the state diagram indicates actions (e.g., process activation) taken as a consequence of a particular event.
- To illustrate the use of a state diagram.
- A state diagram is a graph whose nodes are states and whose directed arcs are transitions between states.
- A state diagram specifies the state sequences caused by event sequences.
- Events represent external stimuli. States represent value of objects.
- All objects in a class execute the state diagram for that class, which models their common behavior.

50

Components of state diagram

- The UML notation for a state diagram is a rectangle with object name in a small pentagonal tag in the upper left corner.



Initial State

- A solid circle with an outgoing arrow shows the initial state.

Final State

- A bull's eye – a solid circle surrounded by a hollow circle/encircled X shows the termination point.

State

- Drawn as a rounded box containing the name of the state.
- State names must be unique within the scope of the state diagram.

Components of state diagram

Transition/Event

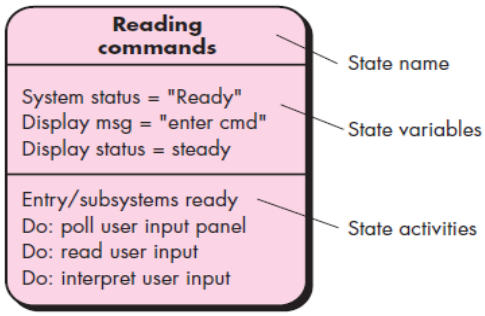
- Drawn as a line from the origin state to the target state.
- An arrowhead points to the target state.

Guard condition

- A guard condition is a Boolean expression that must be true in order for a transition to occur.
- A guarded transition fires when its event occurs.
- Optionally listed in square brackets after an event.

How to draw a state diagram

- **Step 1:** Identify the important objects.
- **Step 2:** Identify the possible states in which the object can exist.
- **Step 3:** Identify the initial state and the final terminating states.
- **Step 4:** Label the events which trigger these transitions.



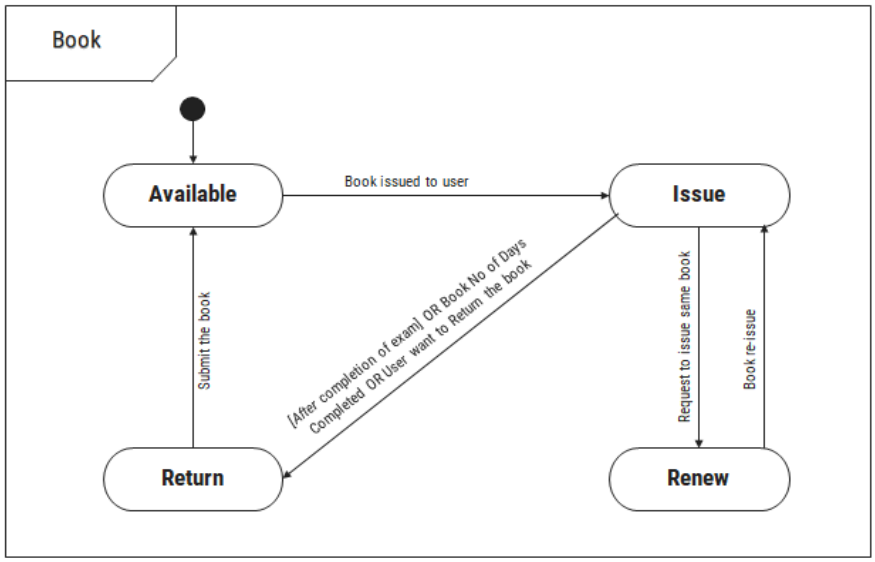
53

State diagram for library management system

- Identify the important objects
  - Book
  - CD/DVD
  - News Paper
  - Librarian
  - Member
- Identify the states of Book's Object
  - Available
  - Issue
  - Return
  - Renew
- Identify the events / transition
  - Book issued to user
  - Submit the book
  - Request to issue same book
  - Completion of exam / end of the Semester

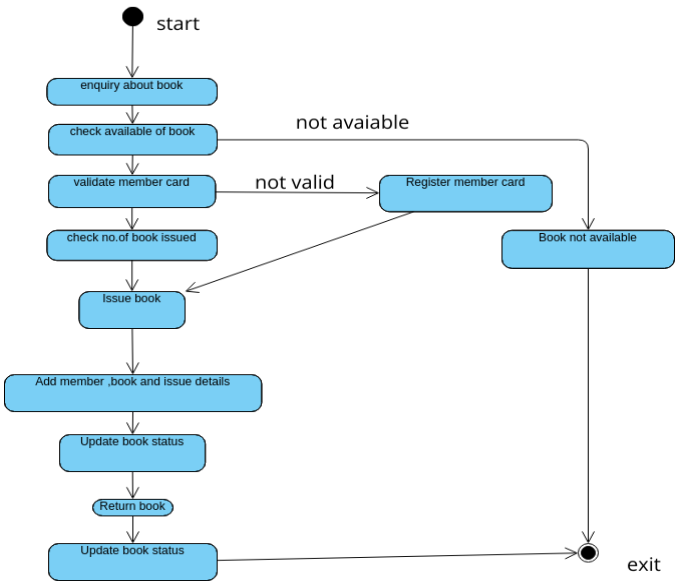
54

State diagram for book



55

State diagram for book



56

Negotiating Requirements

- In an ideal requirements engineering context, the inception, elicitation, and elaboration tasks determine customer requirements in sufficient detail to proceed to subsequent software engineering activities.
- Unfortunately, this rarely happens. In reality, you may have to enter into a negotiation with one or more stakeholders.
- In most cases, stakeholders are asked to balance functionality, performance, and other product or system characteristics against cost and time-to-market.
- The intent of this negotiation is to develop a project plan that meets stakeholder needs while at the same time reflecting the real-world constraints (e.g., time, people, budget) that have been placed on the software team.
- The best negotiations strive for a “win-win” result.

57

Negotiating Requirements

- In an ideal requirements engineering context, the inception, elicitation, and elaboration tasks determine customer requirements in sufficient detail to proceed to subsequent software engineering activities.
- Unfortunately, this rarely happens. In reality, you may have to enter into a negotiation with one or more stakeholders.
- In most cases, stakeholders are asked to balance functionality, performance, and other product or system characteristics against cost and time-to-market.
- The intent of this negotiation is to develop a project plan that meets stakeholder needs while at the same time reflecting the real-world constraints (e.g., time, people, budget) that have been placed on the software team.
- The best negotiations strive for a “win-win” result.

58

Validating Requirements

- As each element of the requirements model is created, it is examined for inconsistency, omissions, and ambiguity.
- The requirements represented by the model are prioritized by the stakeholders and grouped within requirements packages that will be implemented as software increments.
- A review of the requirements model addresses the following questions:
  - Is each requirement consistent with the overall objectives for the system/product?
  - Have all requirements been specified at the proper level of abstraction? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
  - Is the requirement really necessary or does it represent an add-on feature that may not be essential to the objective of the system?
  - Is each requirement bounded and unambiguous?

59

Validating Requirements

- Do any requirements conflict with other requirements?
- Does each requirement have attribution? That is, is a source (generally, a specific individual) noted for each requirement?
- Is each requirement achievable in the technical environment that will house the system or product?
- Is each requirement testable, once implemented?
- Does the requirements model properly reflect the information, function, and behavior of the system to be built?
- Has the requirements model been “partitioned” in a way that exposes progressively more detailed information about the system?
- Have requirements patterns been used to simplify the requirements model? Have all patterns been properly validated? Are all patterns consistent with customer requirements.

60


THANKS