Unit 3
Requirement Analysis and Specification

Analysis Models Part 3

- Activity & Swimlane Diagram

Activity Diagram

An activity diagram visually presents a series of operation or flow of control in a system similar to algorithm or a flowchart.

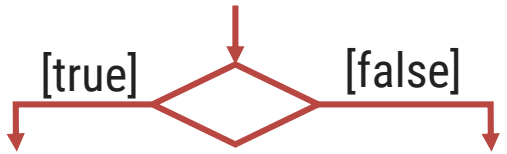
- An activity diagram is like a traditional flowchart in that it show the flow of control from step to step.
- An activity diagram can show both sequential and concurrent flow of control.
- Activity diagram mainly focus on the sequence of operation rather than on objects.
- Activity diagram represent 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.

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



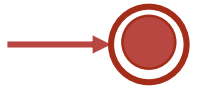
- If there is more than one successor to an activity, each arrow may be labeled with a condition in square brackets. For e.g. *[failure]*
- As a notational convenience, a diamond shows a branch into multiple successors.
- The diamond has one incoming arrows and two or more outgoing arrows. Each with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

Elements of Activity Diagram Cont.

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

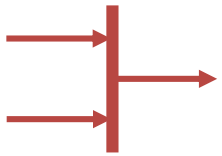


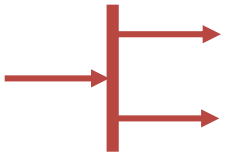
- A bull's eye – a solid circle surrounded by a hollow circle shows the termination point.
- The symbol only has incoming arrows.
- When control reaches a bull's eye, the overall activity is complete and execution of the activity diagram ends.

Elements of Activity Diagram Cont.

Concurrent Activities

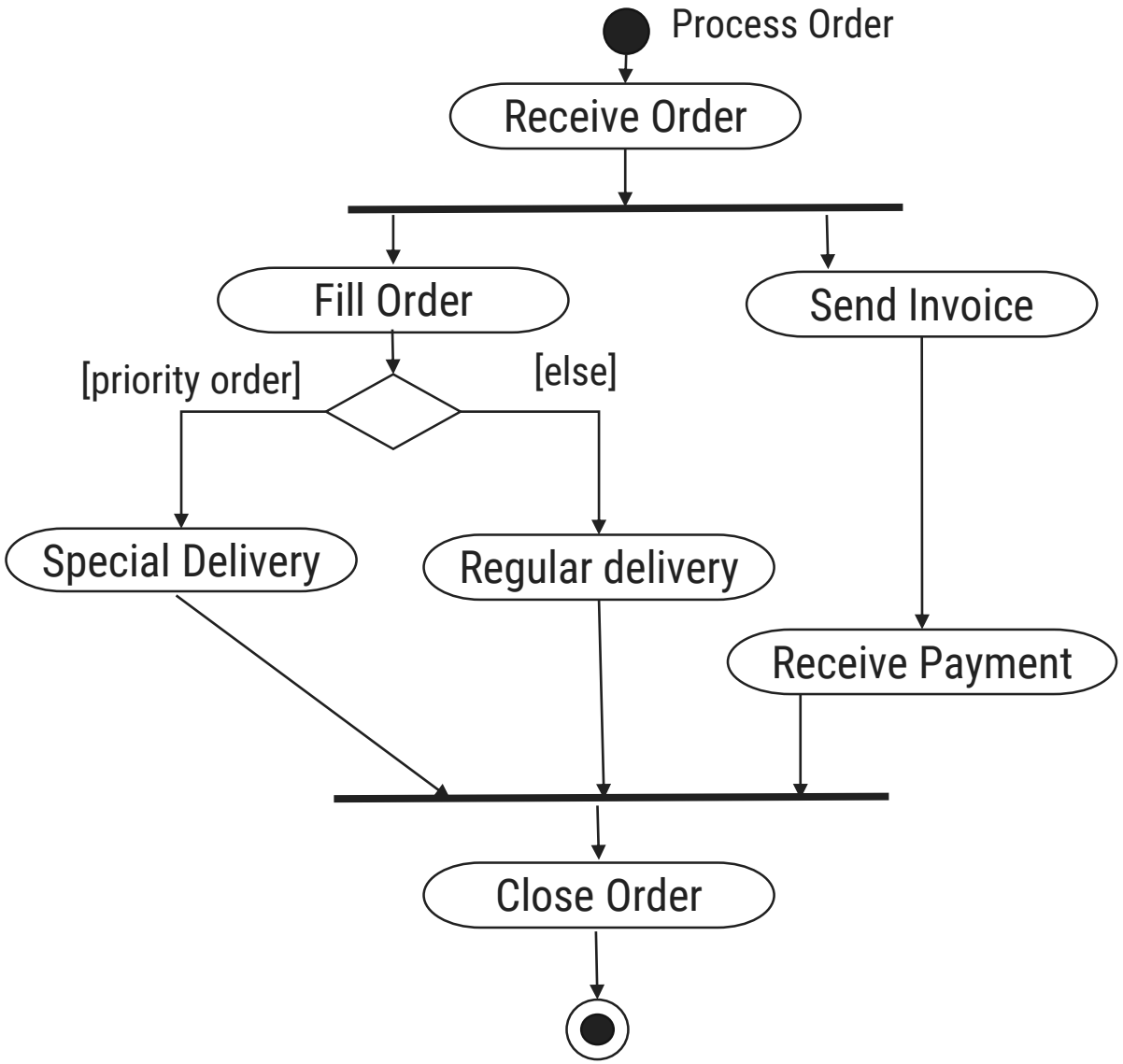
- System can perform more than one activity at a time.
- For e.g. one activity may be followed by another activity, then split into several concurrent activities (a fork of control), and finally be combined into a single activity (a merge of control).
- A fork or merge is shown by a synchronization bar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

Merge 

Fork 

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 diagram

- 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.

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.

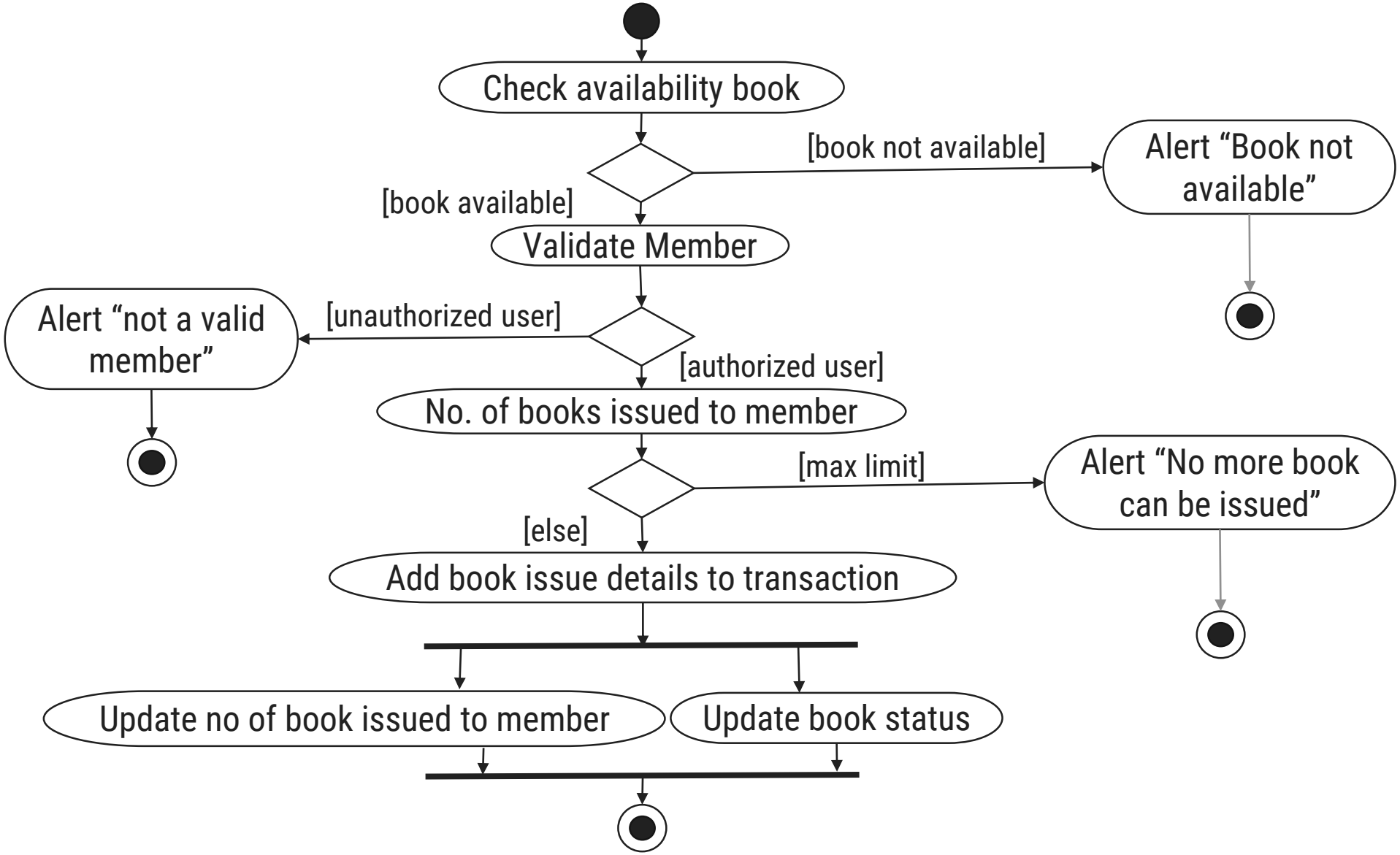
Be careful with concurrent activities

- 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

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.

Activity Diagram for Book Issue



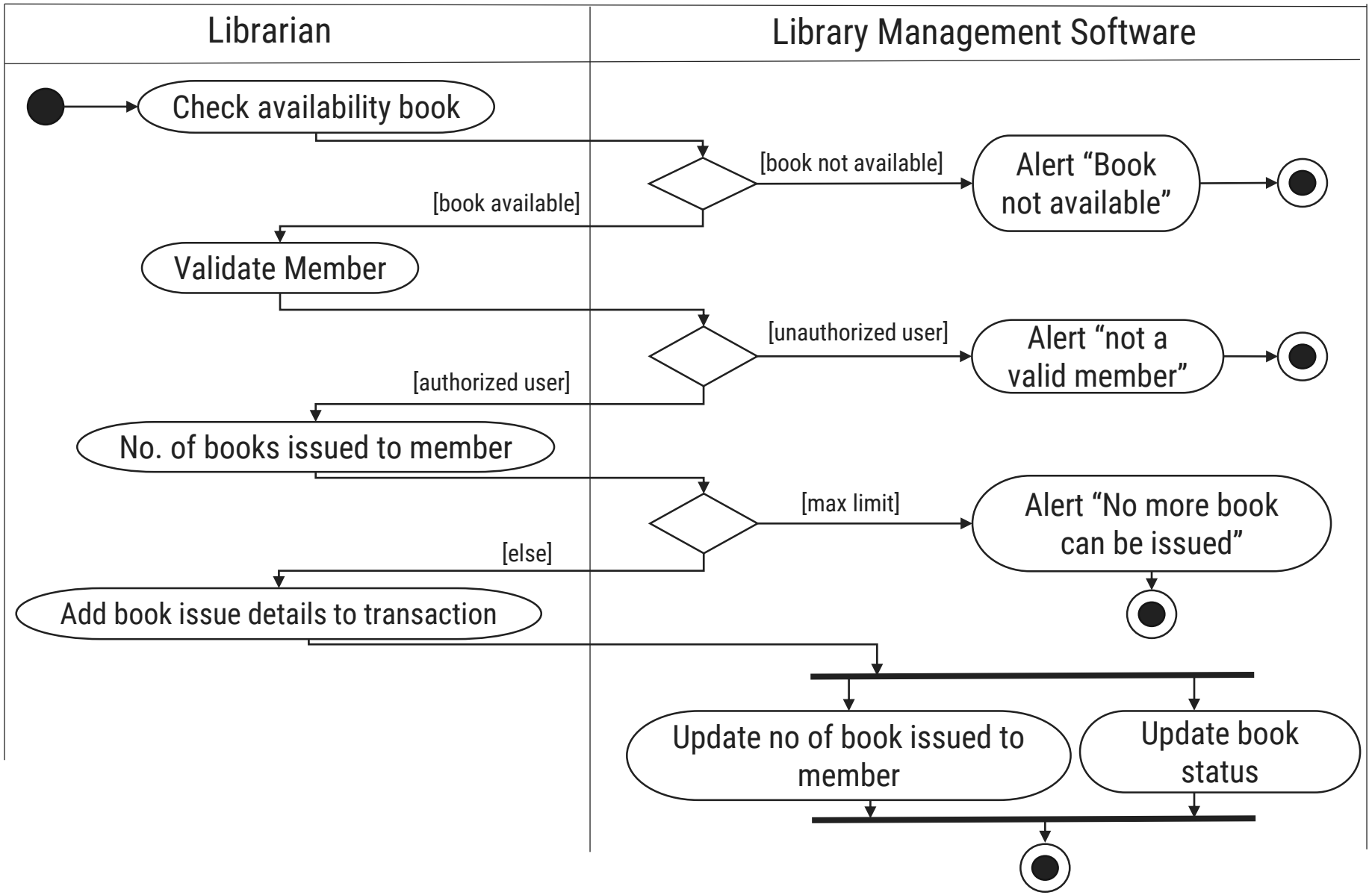
Swimlane Diagram

- 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.

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

Swimlane Diagram for Book Issue



Unit 3
Requirement analysis and Specification

Analysis Models Part 1

- Class Diagram

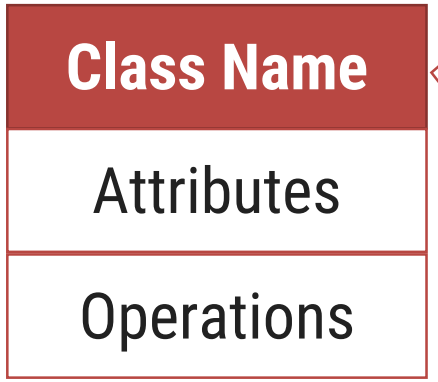
Class diagram

The **purpose** of class modeling is to describe **objects in systems** and **different types of relationships between them**.

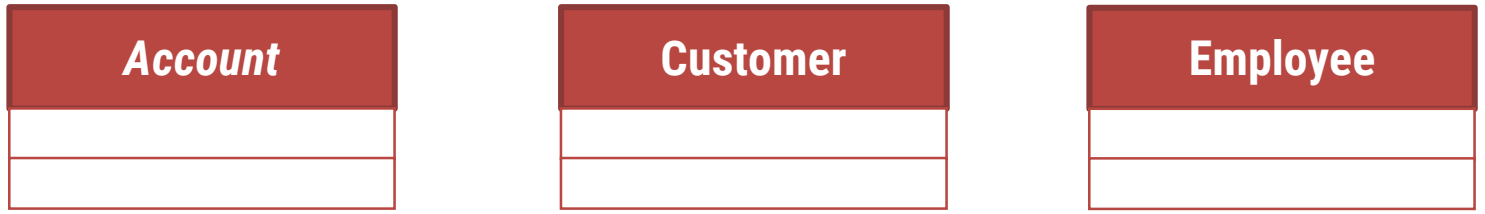
The class diagram is used to **construct** and **visualize** object-oriented systems.

- Class modeling is used to specify the structure of the objects, classes, or components that exist in the problem domain or system.
- Class diagram provides a graphic notation for modeling classes and their relationships.
- Class is a blueprint of an object.
- An object is a concept, abstraction, or thing with an identity that has meaning for an application.
- Class diagrams represent an overview of the system like classes, attributes, operations, and relationships.

Elements of Class Diagram (Class Name)



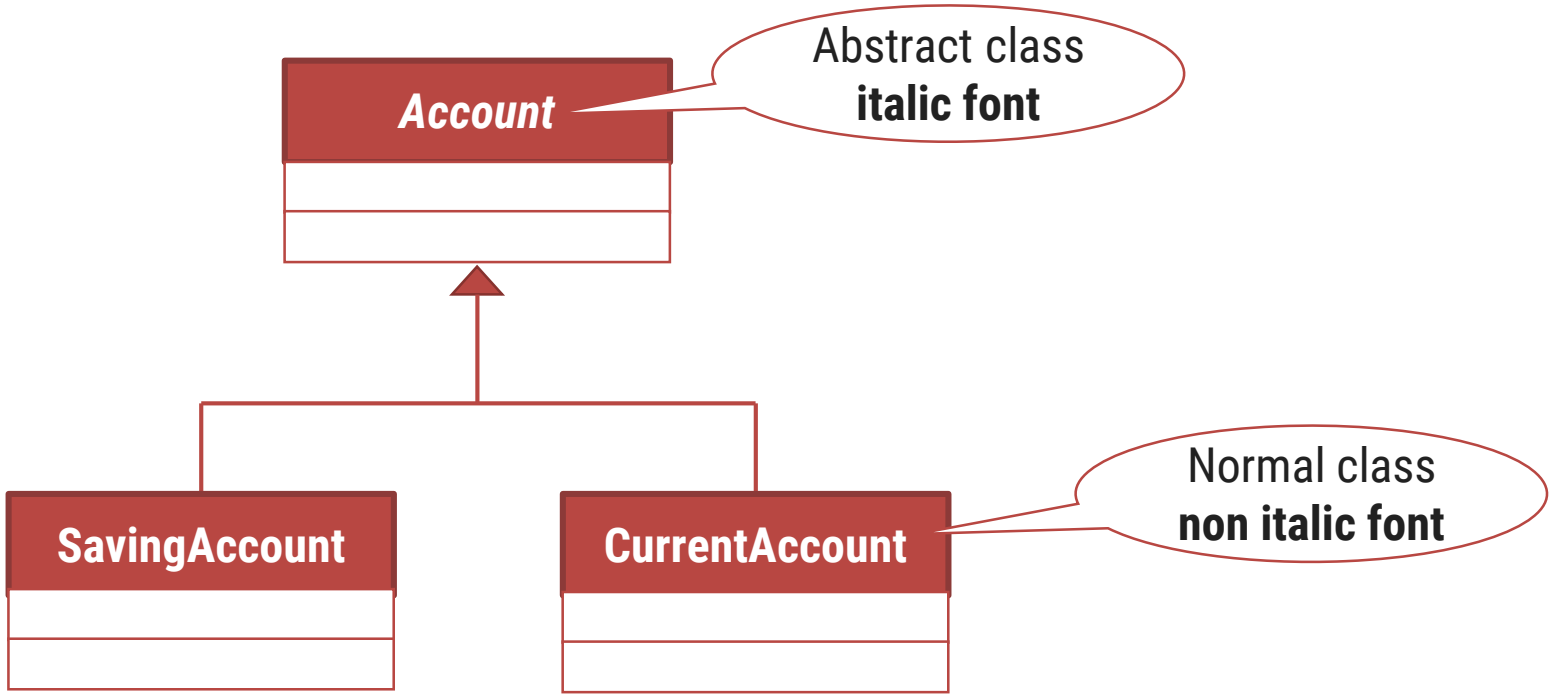
- The name of the class appears in the upper section.
- Class name should be meaningful.
- Class name should always be aligned center of the upper section.
- Class name should start with capital letters, and intermediate letter is a capital.
- Class name should be always bold format.
- For e.g.:



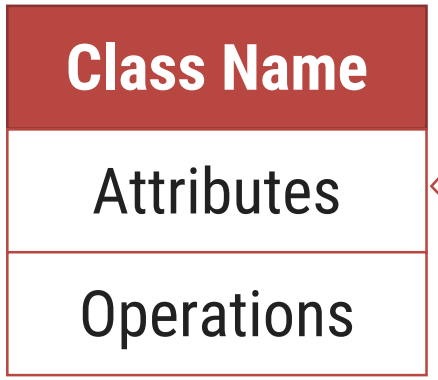
- Abstract class name should be written in italic format.

Elements of Class Diagram (Class Name) Cont.

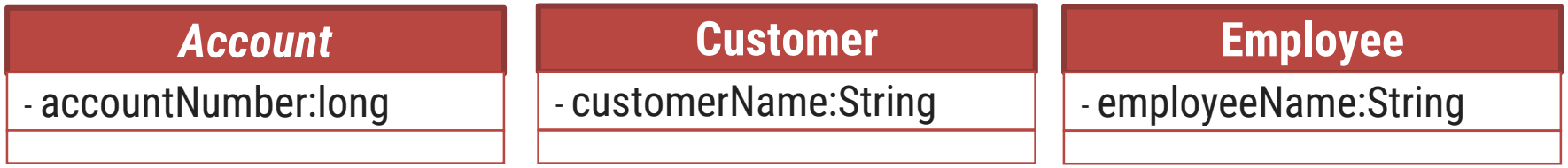
- For e.g. in the banking system, there are two types of accounts; one is a saving account and another is a current account.
- **Account is an abstract class** and saving account and the current account is a subclass of Account.
- The system can't directly access the Account class. It is accessible by only saving accounts and current accounts.



Elements of Class Diagram (Attributes)

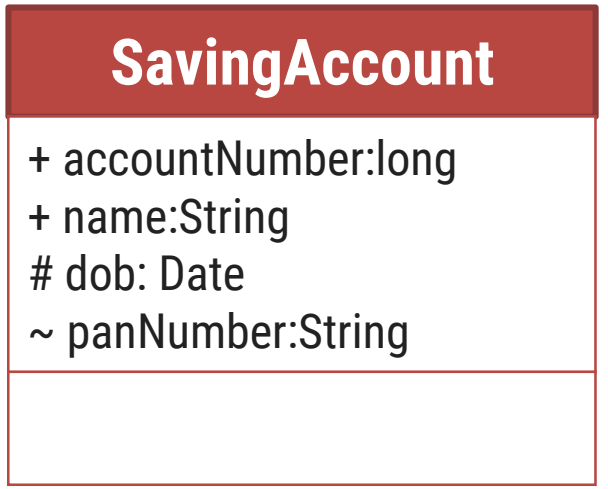


- An attribute is a named **property of a class** that describes a value held by each object of the class.
- The UML notation lists attributes in the second compartment of the class box.
- The attribute name should be in the regular face, left align in the box & use the lowercase letters for the first 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*

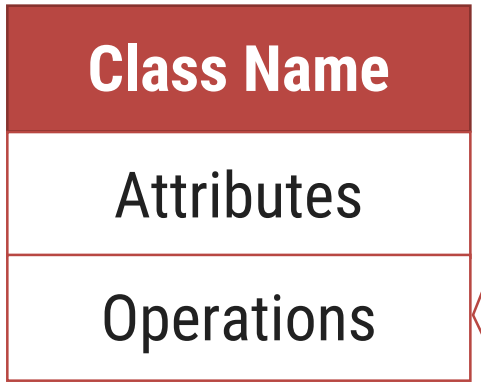


Elements of Class Diagram (Access Modifiers)

- 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.
- Static (underlined) : Member can be accessed using class name only.
- In example you can see how to use access specifier

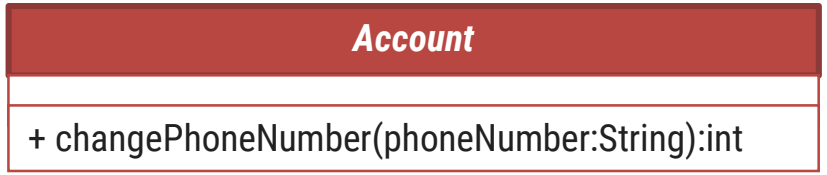


Elements of Class Diagram (Operation)



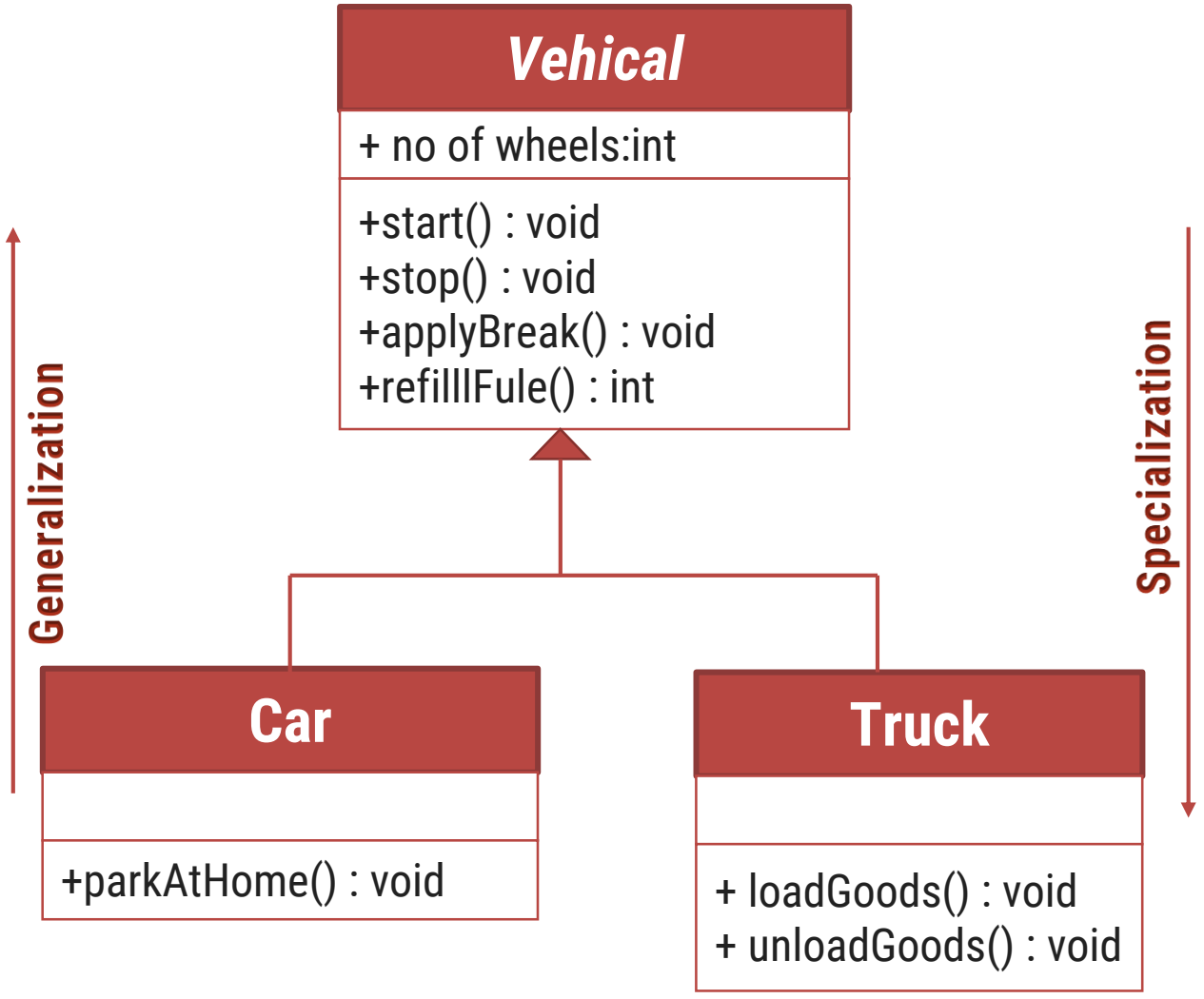
- The operation is a function or procedure that may be applied to objects in a class.
- The UML notation is to list operations in the third compartment of the class box.
- The operation name in the regular face, left align the name in the box, and use a lowercase letter for the first character.
- Optional 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

For e.g.: you can see change phone number is a method that accepts phone number as an argument and return the int value as a response.



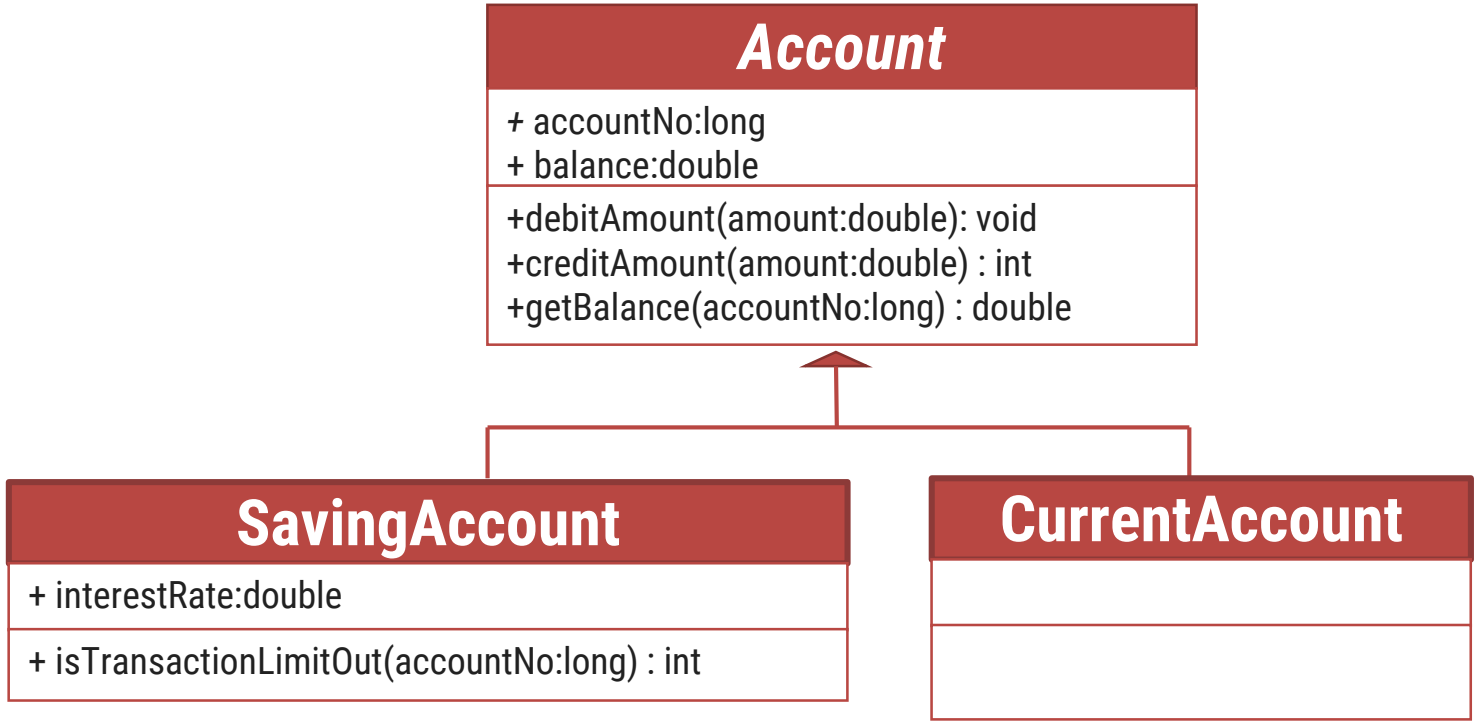
Generalization & Specialization

- 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.



Generalization & Specialization

- For example in a bank, any Customer opens an account.
- The account can be either a savings account or a current account. In saving account, customer earns fixed interest on the deposit. But this facility is not available in the current account.



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.

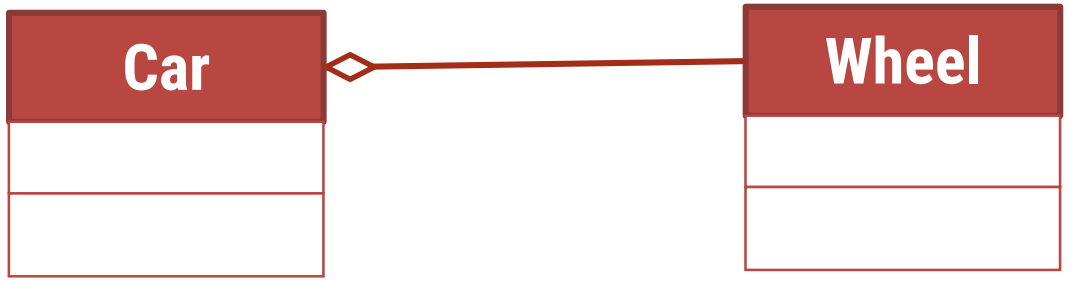
Aggregation

Aggregation is a **subset of association**. it is a collection of different things.
It is more specific than an association.

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.



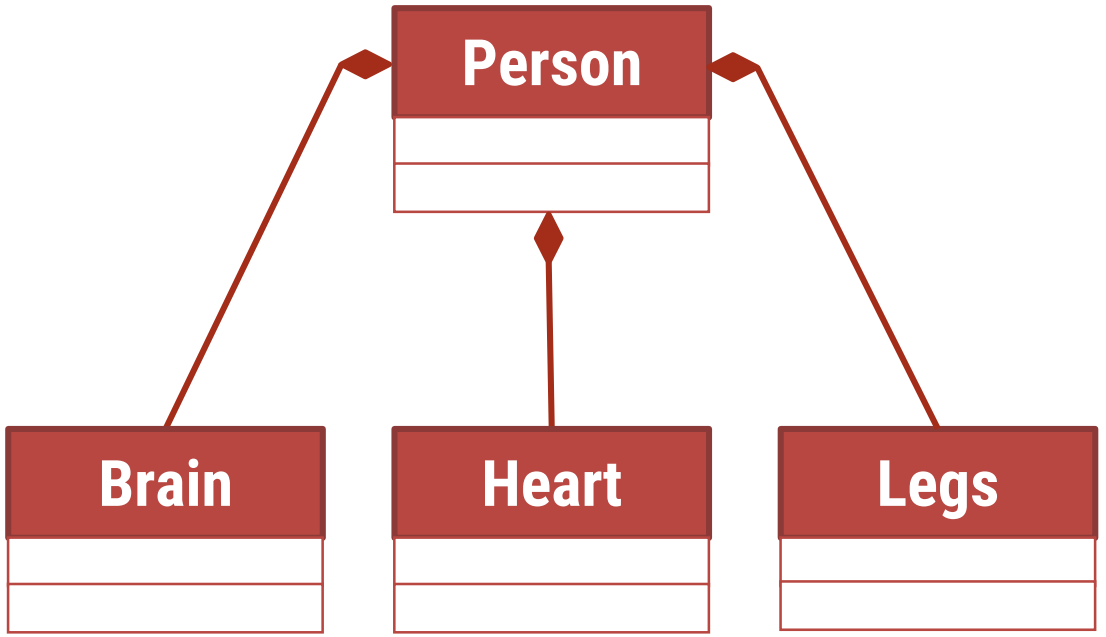
Composition

The composition is a part of the aggregation. 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.



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:

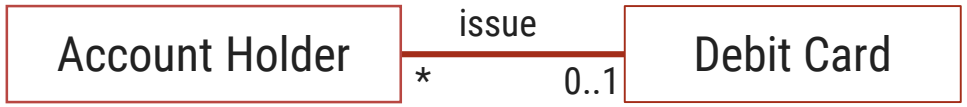
Multiplicity	Option	Cardinality
0..1		No instances or one instance
1..1	1	Exactly one instance
0..*	*	Zero or more instances
1..*		At least one instance
5..5	5	Exactly 5 instances
m..n		At least m but no more than n instances

Example Of Multiplicity

One to One Association

One account holder has one cheque book

Many to Zero or One Association

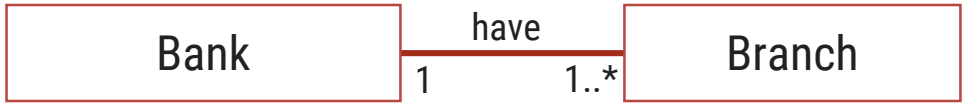


An account holder can issue at most one debit card.

Many to Many Association

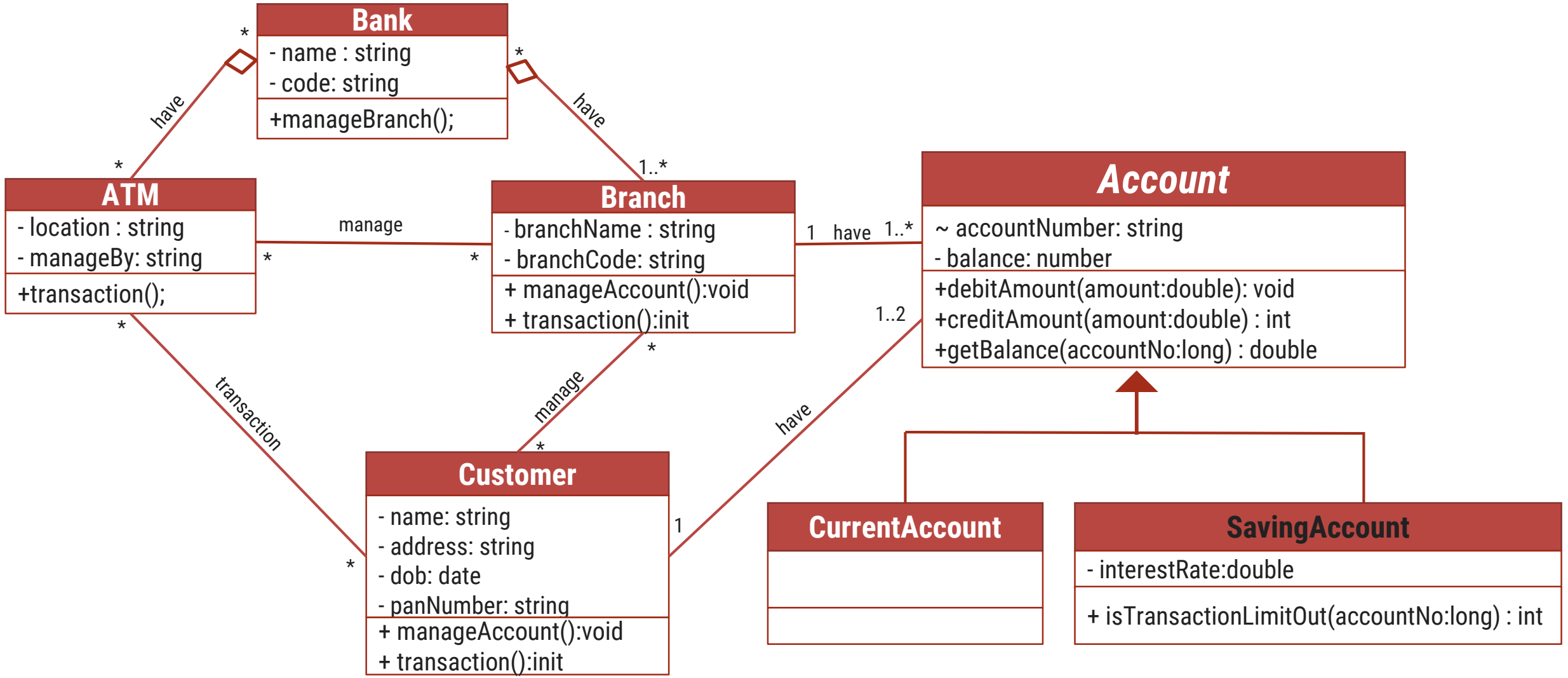
Every account holder can withdraw money from all ATMs.

One to One or Many Association

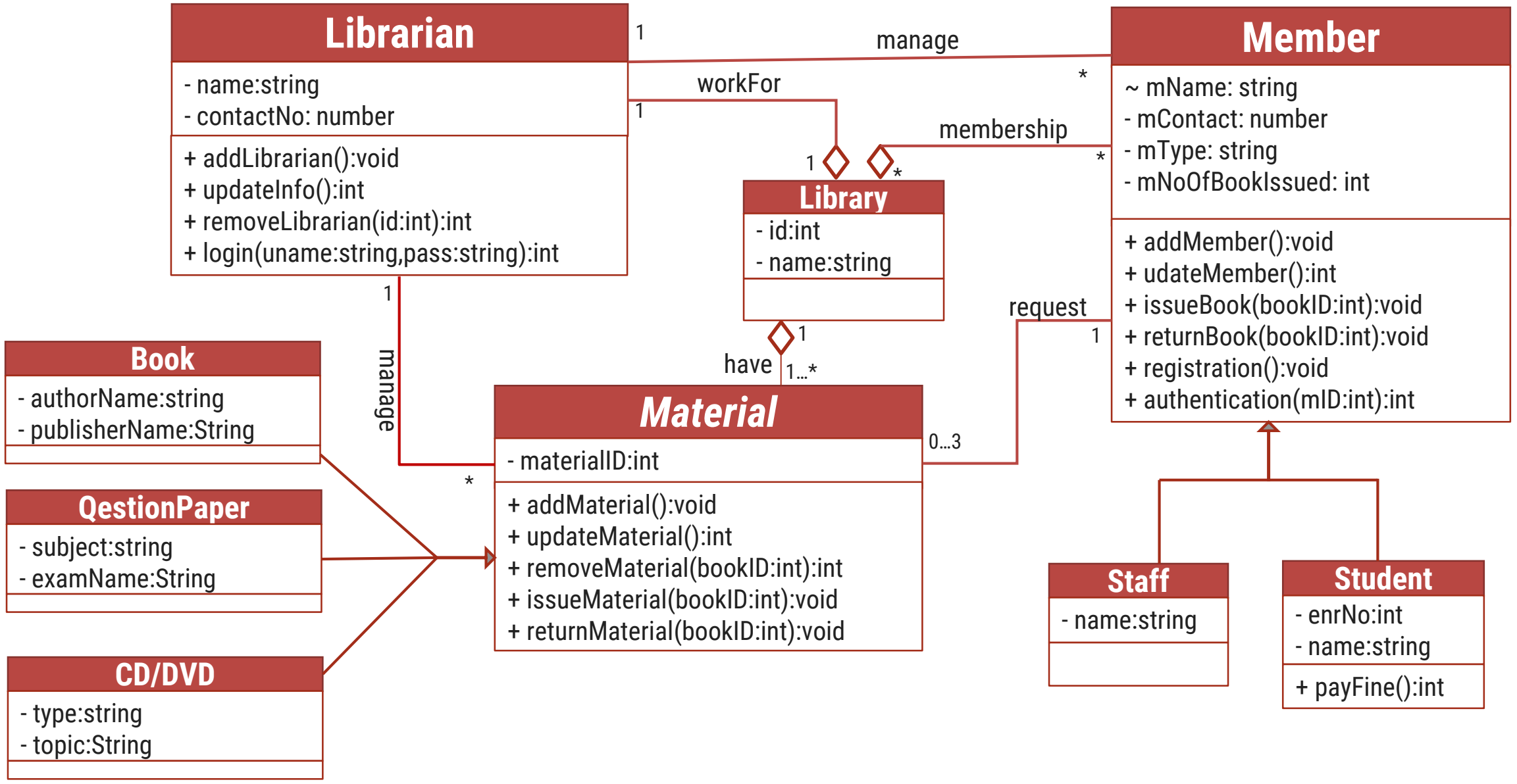


The bank should have at least one branch.

Class Diagram Of Bank Management System



Class Diagram Of Library Management System



Unit 3
Requirement Analysis and Specification

Analysis Models Part 4

- Sequence Diagram

Sequence Diagram

A Sequence diagram shows the participants (Objects) in an interaction and the sequence of message among them.

- A sequence diagram shows the interaction of a system with its actors to perform all or part of a use case.
- Sequence diagram represent the dynamic communication between object during execution of task.
- Each use case requires one or more sequence diagram to describe its behavior.
- Each sequence diagram shows a particular behavior sequence of the use case.
- It is best to show a specific portion of a use case and not attempt to be too general.
- You can draw a separate sequence diagram for each task.

Components of Sequence Diagram

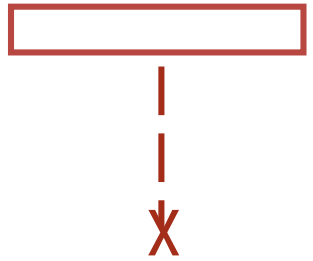
Object - Class Roles or Participants

- Class roles describe the way an object will behave in context.
- Use the UML object symbol to illustrate class roles, but don't list object attributes.

Activation or Execution Occurrence

- Activation boxes represent the time an object needs to complete a task.
- When an object is busy executing a process or waiting for a reply message, use a thin gray rectangle placed vertically on its lifeline.

Lifeline



- A lifeline represents a Object in an interaction.
- When that object's lifeline ends, you can place an X at the end of its lifeline to denote a destruction occurrence.

Components of Sequence Diagram Cont.

Messages

- Messages are arrows that represent communication between objects.
- Use the following arrows and message symbols to show how information is transmitted between objects.

Synchronous message 

- Represented by a solid line with a solid arrowhead.
- This symbol is used when a sender must wait for a response to a message before it continues.
- The diagram should show both the call and the reply.

Asynchronous message 

- Represented by a solid line with a lined arrowhead.
- Asynchronous messages don't require a response before the sender continues.
- Only the call should be included in the diagram.

Components of Sequence Diagram Cont.

Reply message

- Represented by a dashed line with a lined arrowhead.
- these messages are replies to calls.

Delete message

- Represented by a solid line with a solid arrowhead, followed by an X.
- This message destroys an object.

Guideline for Sequence Diagram

Prepare at least one scenario per use case

- The steps in the scenario should be logical commands, not individual button clicks.
- You can specify the exact syntax of input.
- Start with the simplest mainline interaction - no repetitions, one main activity, and typical values for all parameters.
- If there are substantially different mainline interactions, write a scenario for each.

Abstract the scenarios into sequence diagrams.

- The sequence diagrams clearly show the contribution of each actor.
- It is important to separate the contribution of each actor as a prelude to organizing behavior about objects.

Divide complex interactions

- Break large interactions into their constituent tasks and prepare a sequence diagram for each of them.

Prepare a sequence diagram for each error condition.

- Show the system response to the error condition.

Steps to Draw a Sequence Diagram

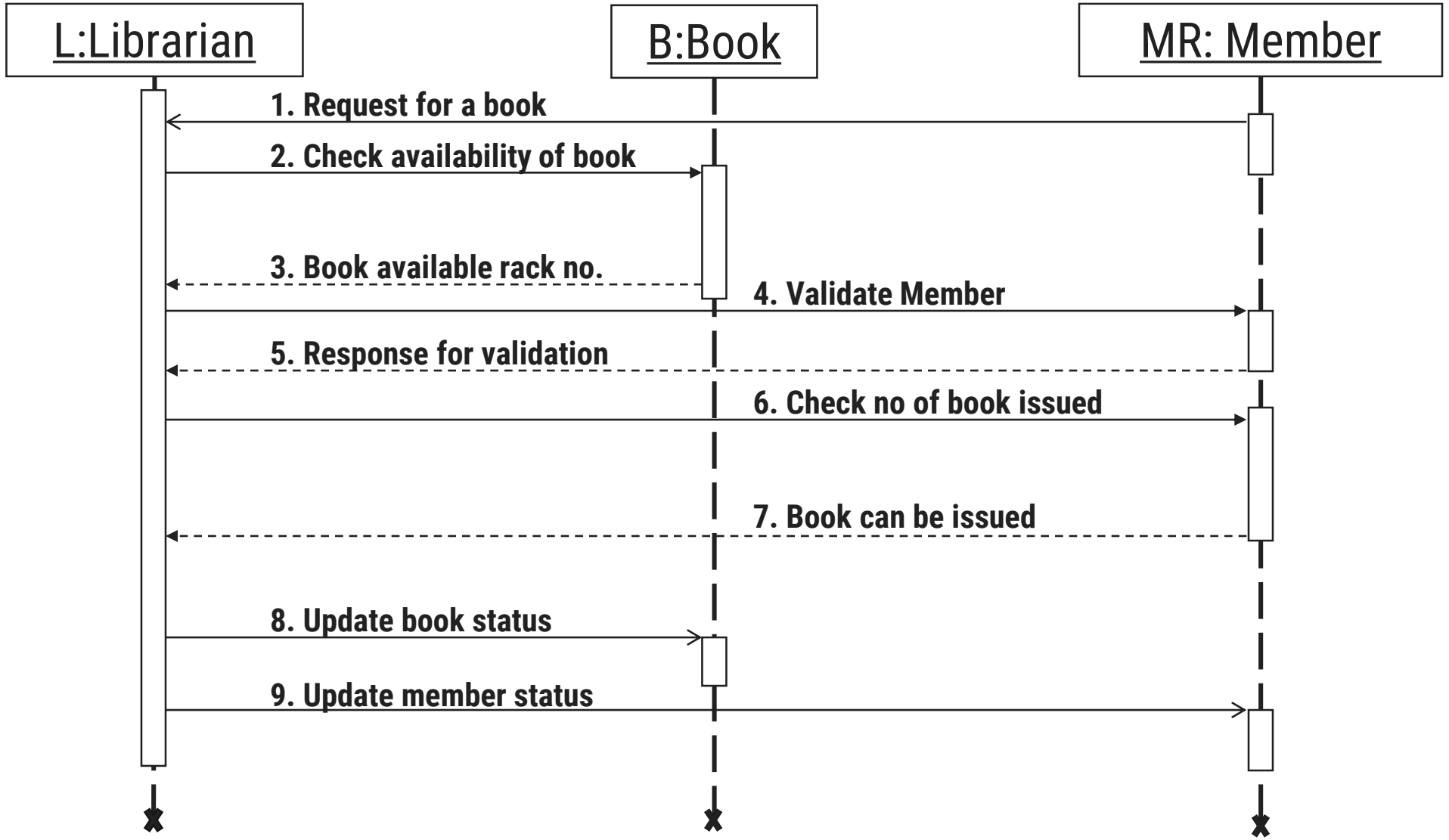
Step-1	Select one scenario
Step-2	Identify the necessary set of the objects. Who is taking part ?
Step-3	Identify the necessary interactions/steps.
Step-4	Describe the message exchange between object.
Step-5	Identify the sequence of interactions and who starts Interactions.

Example: Sequence Diagram for Book Issue

- **Book issue** is a one business process or a function in Library Management System.
- Necessary **objects for book issue process** are **Librarian**, **Book**, **Member** and **Transaction** .
- Member class object starts the interaction.
- Various **interactions** in book issue process are

1	Request for a book
2	Check availability of book
3	Validate the member
4	Check No. of books issued by member
5	Add book issue details to transaction
6	Update no of book issued by member
7	Update book status

Sequence Diagram for Book Issue



Unit 3
Requirement analysis and Specification

Analysis Models Part 1

- Class Diagram

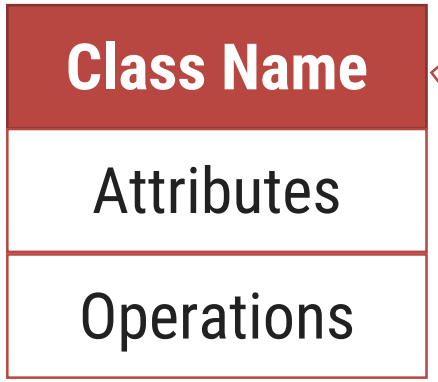
Class diagram

The **purpose** of class modeling is to describe **objects in systems** and **different types of relationships between them**.

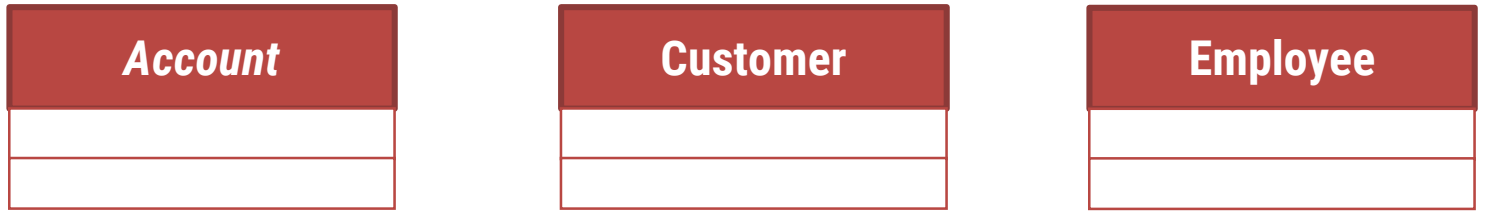
The class diagram is used to **construct** and **visualize** object-oriented systems.

- Class modeling is used to specify the structure of the objects, classes, or components that exist in the problem domain or system.
- Class diagram provides a graphic notation for modeling classes and their relationships.
- Class is a blueprint of an object.
- An object is a concept, abstraction, or thing with an identity that has meaning for an application.
- Class diagrams represent an overview of the system like classes, attributes, operations, and relationships.

Elements of Class Diagram (Class Name)



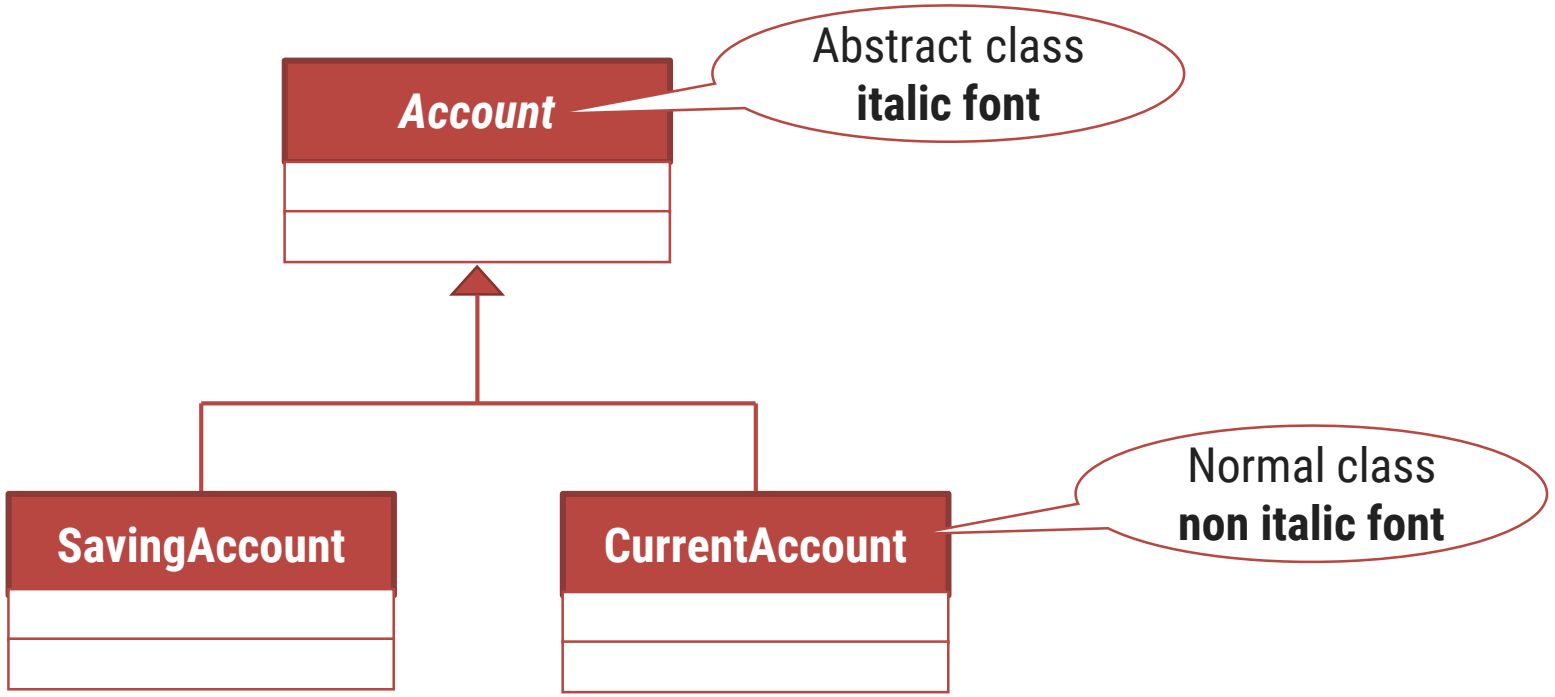
- The name of the class appears in the upper section.
- Class name should be meaningful.
- Class name should always be aligned center of the upper section.
- Class name should start with capital letters, and intermediate letter is a capital.
- Class name should be always bold format.
- For e.g.:



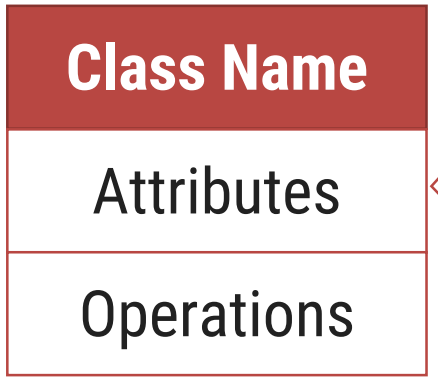
- Abstract class name should be written in italic format.

Elements of Class Diagram (Class Name) Cont.

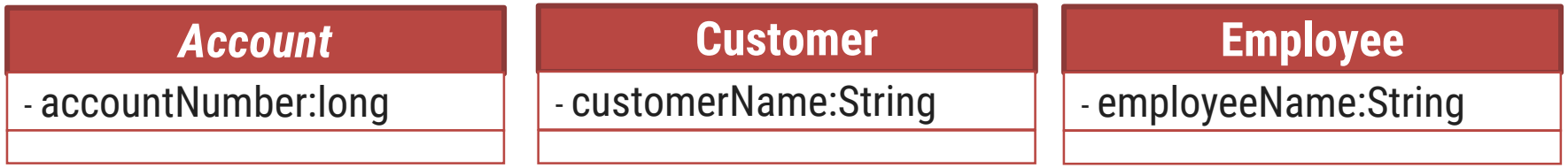
- For e.g. in the banking system, there are two types of accounts; one is a saving account and another is a current account.
- **Account is an abstract class** and saving account and the current account is a subclass of Account.
- The system can't directly access the Account class. It is accessible by only saving accounts and current accounts.



Elements of Class Diagram (Attributes)

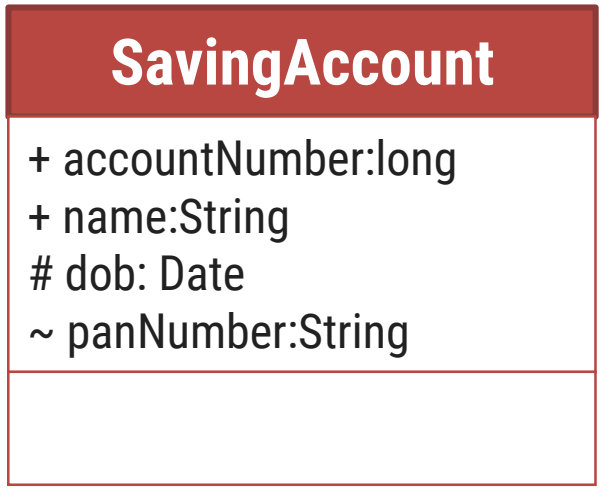


- An attribute is a named **property of a class** that describes a value held by each object of the class.
- The UML notation lists attributes in the second compartment of the class box.
- The attribute name should be in the regular face, left align in the box & use the lowercase letters for the first 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*

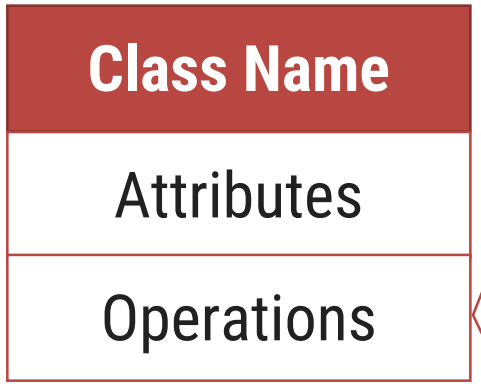


Elements of Class Diagram (Access Modifiers)

- 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.
- Static (underlined) : Member can be accessed using class name only.
- In example you can see how to use access specifier

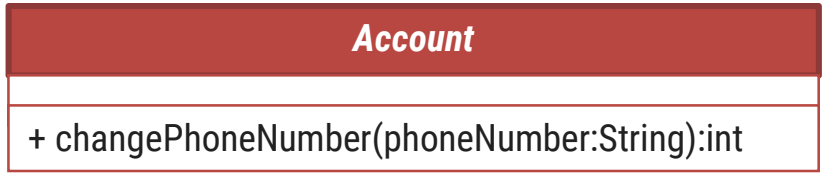


Elements of Class Diagram (Operation)



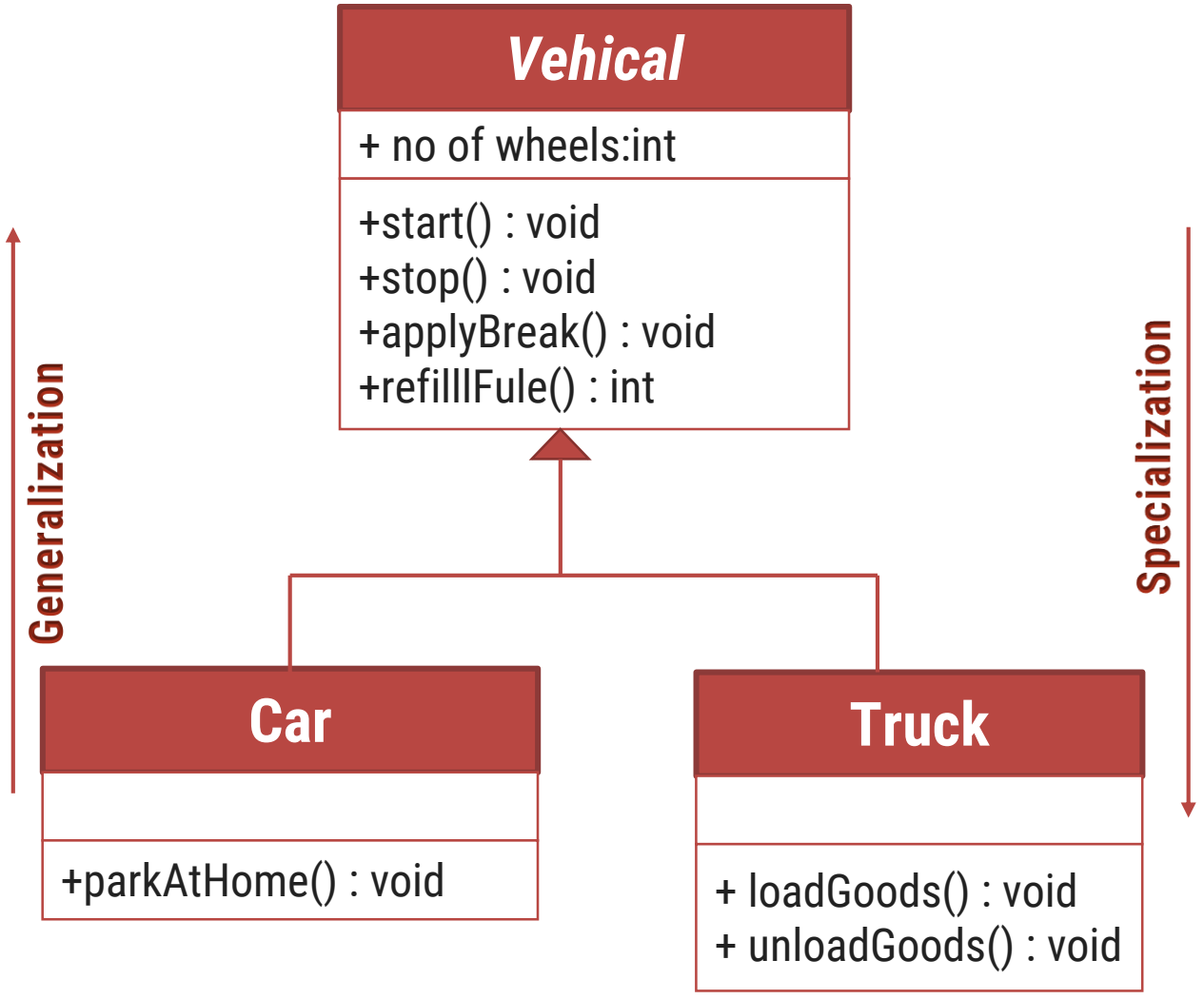
- The operation is a function or procedure that may be applied to objects in a class.
- The UML notation is to list operations in the third compartment of the class box.
- The operation name in the regular face, left align the name in the box, and use a lowercase letter for the first character.
- Optional 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

For e.g.: you can see change phone number is a method that accepts phone number as an argument and return the int value as a response.



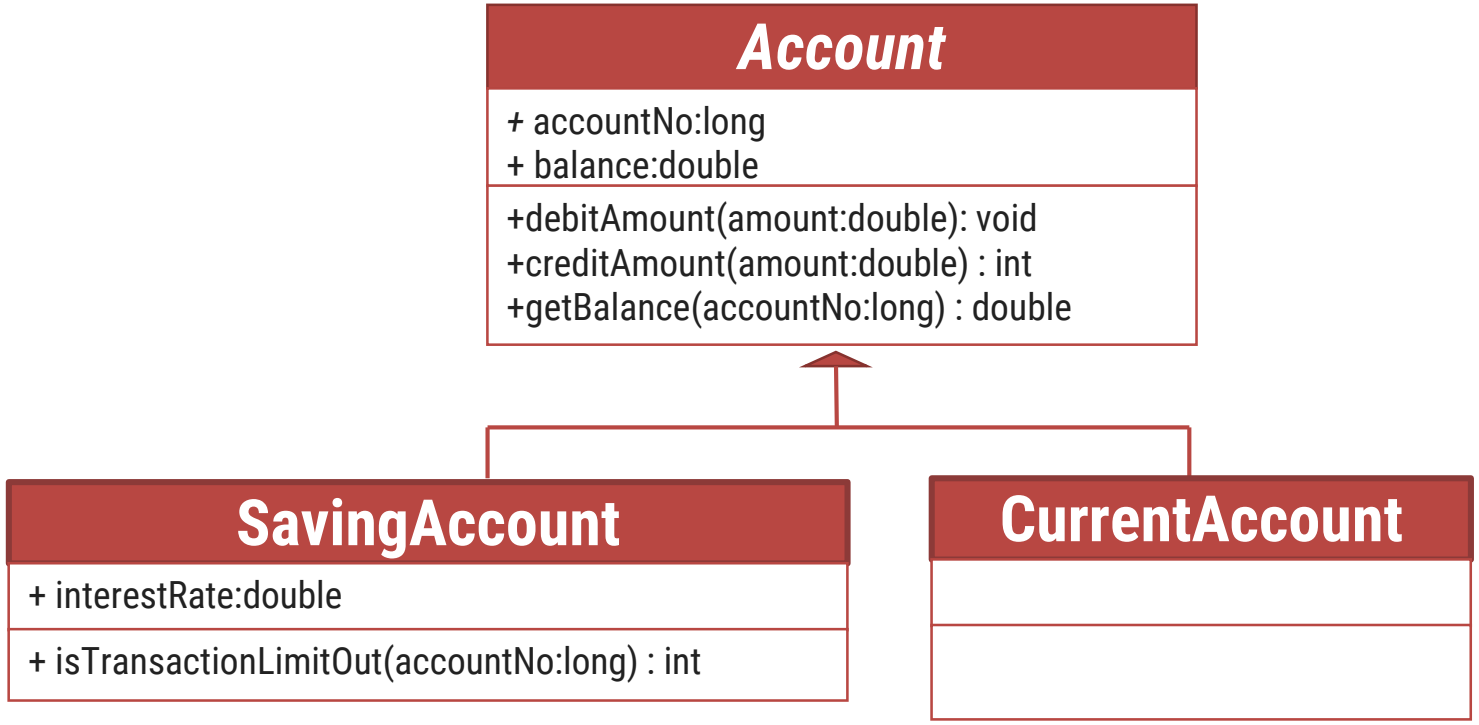
Generalization & Specialization

- 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.



Generalization & Specialization

- For example in a bank, any Customer opens an account.
- The account can be either a savings account or a current account. In saving account, customer earns fixed interest on the deposit. But this facility is not available in the current account.



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.

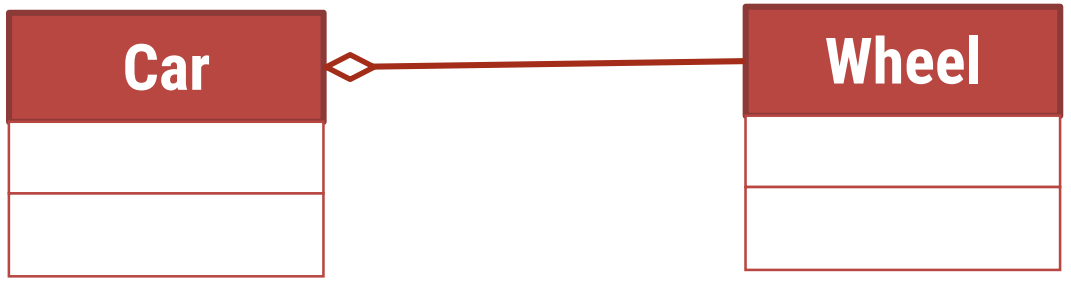
Aggregation

Aggregation is a **subset of association**. it is a collection of different things.
It is more specific than an association.

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.



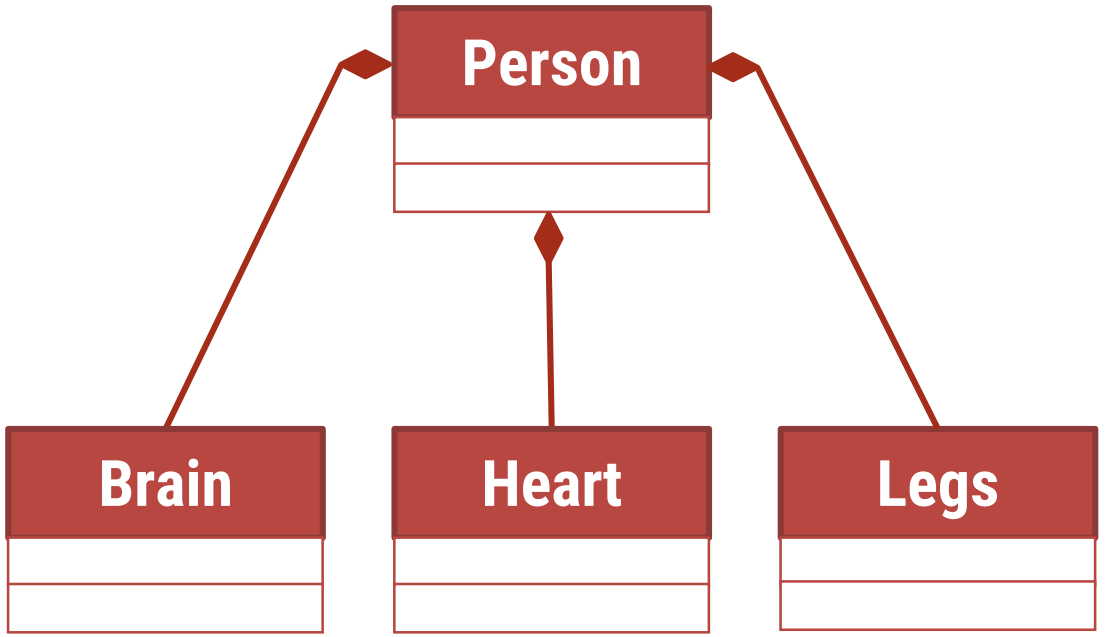
Composition

The composition is a part of the aggregation. 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.



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:

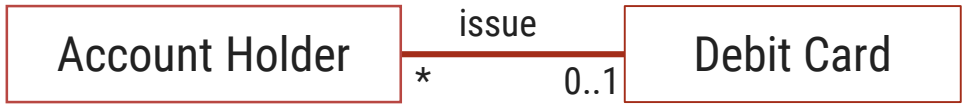
Multiplicity	Option	Cardinality
0..1		No instances or one instance
1..1	1	Exactly one instance
0..*	*	Zero or more instances
1..*		At least one instance
5..5	5	Exactly 5 instances
m..n		At least m but no more than n instances

Example Of Multiplicity

One to One Association

One account holder has one cheque book

Many to Zero or One Association

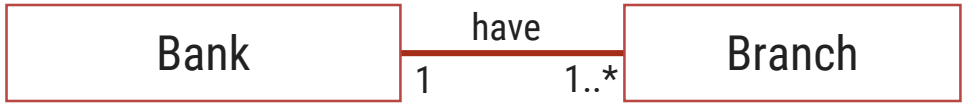


An account holder can issue at most one debit card.

Many to Many Association

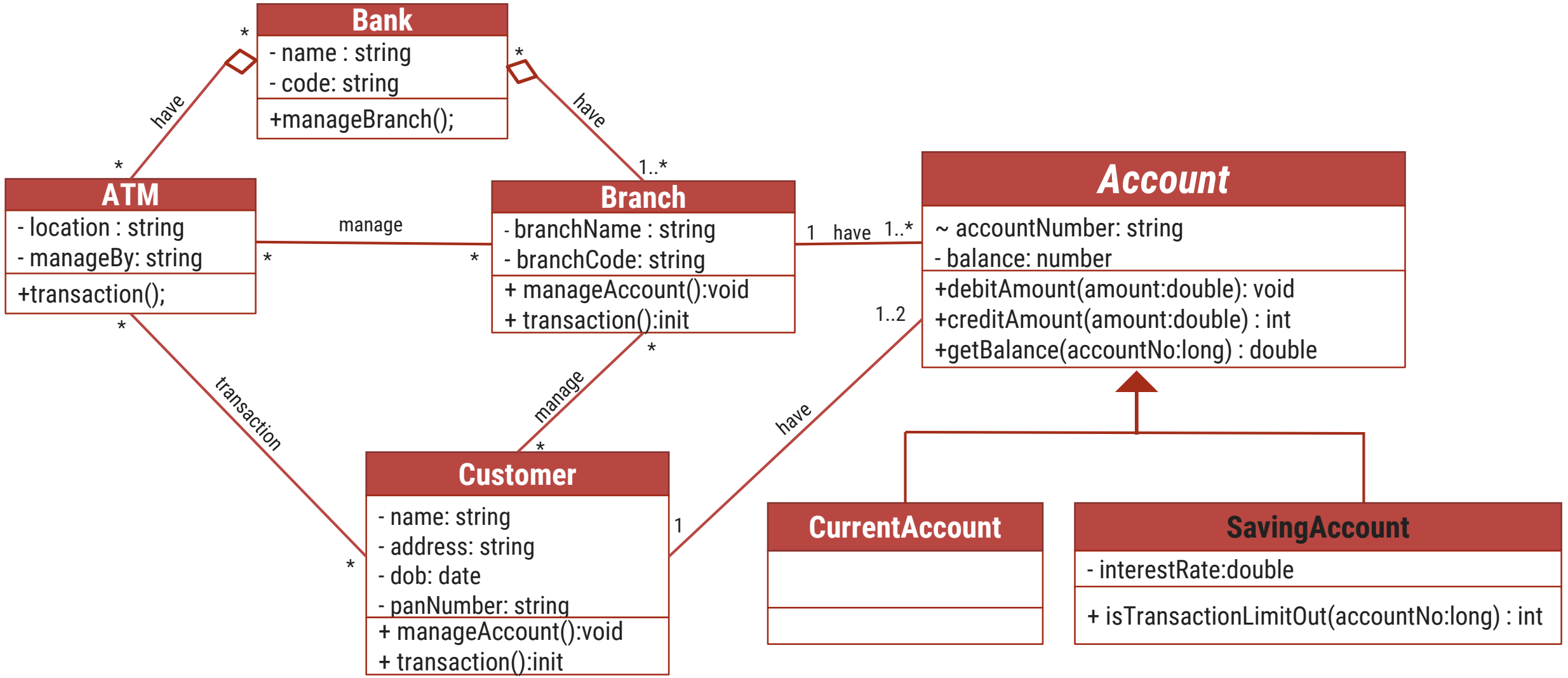
Every account holder can withdraw money from all ATMs.

One to One or Many Association

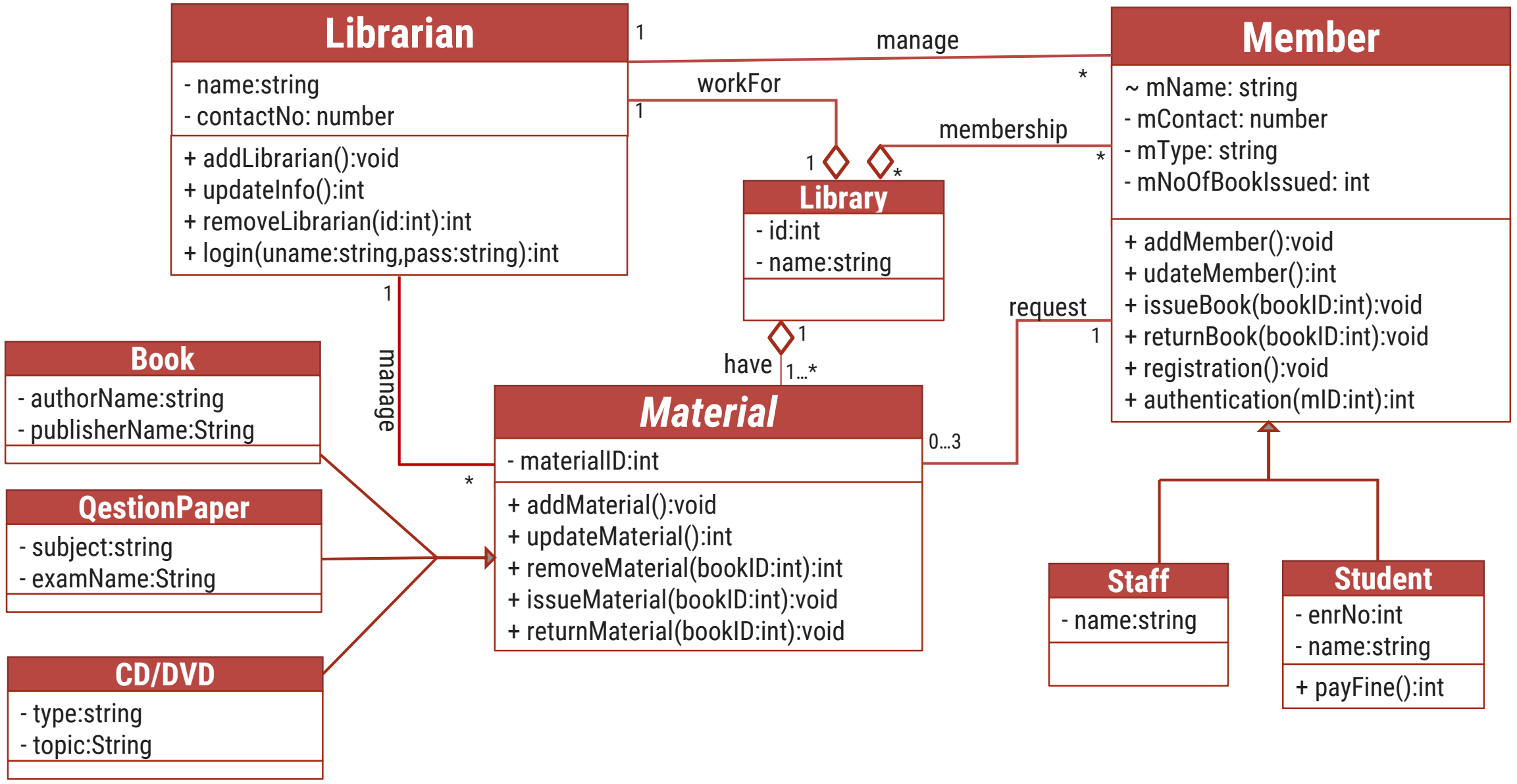


The bank should have at least one branch.

Class Diagram Of Bank Management System



Class Diagram Of Library Management System



Unit 3
Requirement Analysis and Specification

Analysis Models Part 2

- Use Case Diagram
- Usage Scenarios & Story Writing

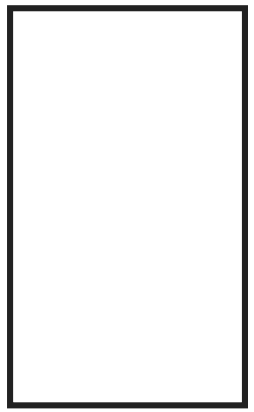
Use Case Diagram

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.

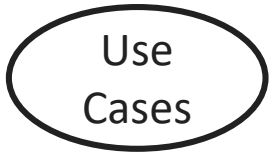
Components of Use Case diagram

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

Use case



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

Actor

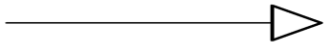
- An actor is an entity that interacts directly with the system but that is not part of system
- Actor may be people, computer hardware, other systems, etc.

Association

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

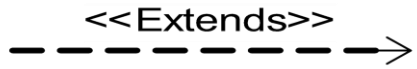
Components of Use Case diagram Cont.

Generalization



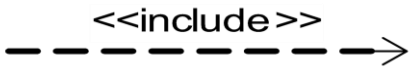
- A generalization relationship is used to represent the inheritance relationship between model elements of the same type

Extends



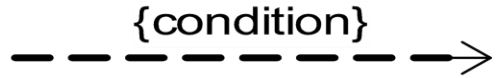
- The extend relationship specifies that the incorporation of the extension use case is dependent on what happens when the base use case executes.

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.

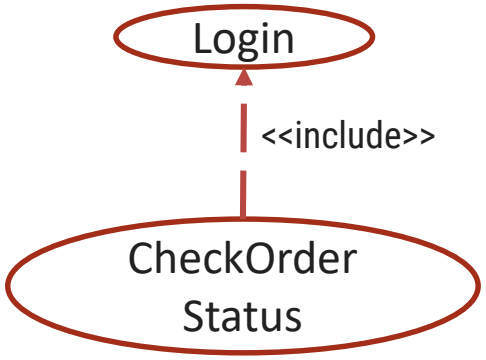
Constraint



- Show condition exists between actors and activity

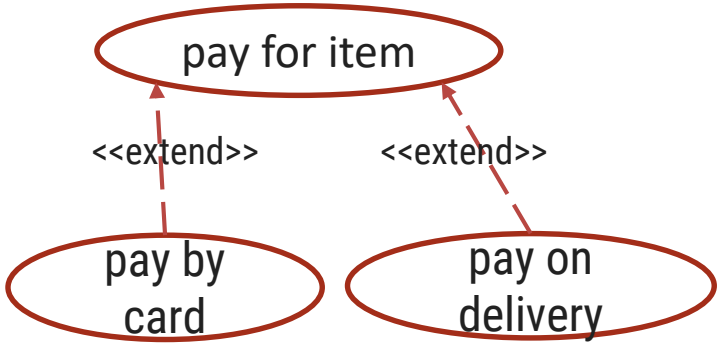
Example of Extends and Include

Includes



- 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



- 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.

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

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.

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

Relationship Between Functionality & Stakeholders

Librarian

- Register user
- Add Article
- Update Article
- Delete Article
- Inquiry Members
- Inquiry Issuance
- Check out Article
- Check in Article
- Set user Permission

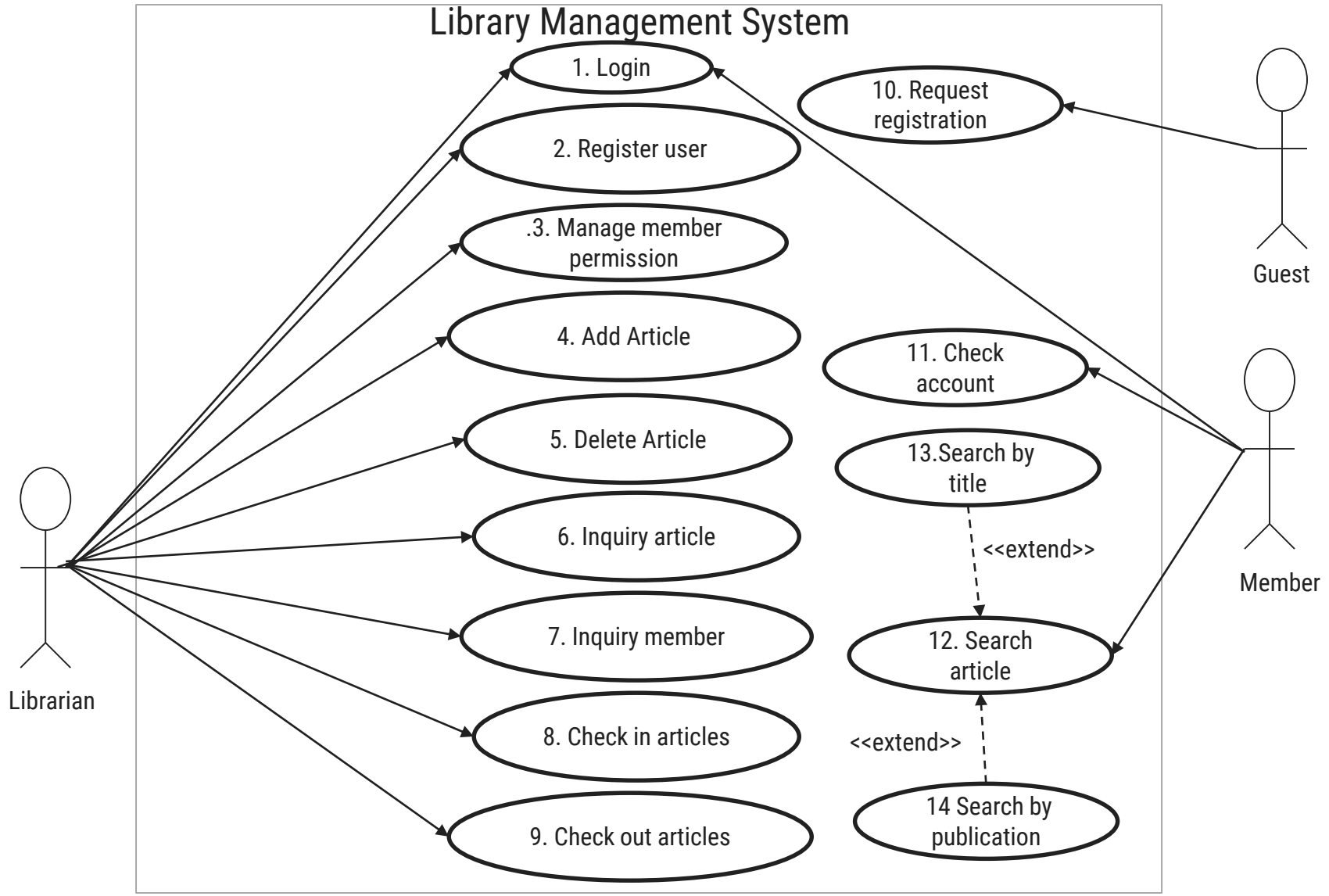
Guest

- Request for registration

Member

- Reserve Article
- Search Article
- Check Account

Use Case Diagram Library Management



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?
- 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?

Usage Scenarios & Story Writing

- 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.
- User stories provide a rapid way of handling customer requirements instead of formal requirement documents
- Gherkin language is used to writing an effective story of the system requirement.
- Gherkin is a human-readable language for system behavior description, which uses indentation to define the structure of the document.
- Each line starts with one of the keywords and describes one of the steps.

Login Usage Scenarios and Story

Feature	Login
Scenario	User Login with valid username and password
Prerequisite	User must have proper client installed on user terminal.
Story	**Given**: User navigated to the Login Screen. **When**: User enters the correct User Name. **And** the user enters the correct password. **And** user Click “login” button. **Then**: System verify user information. **display** dashboard of user. **display** username on top of the right side. **display** logout button.

Login Usage Scenarios and Story

Feature	Login
Scenario	User Login with invalid username and password
Prerequisite	User must have proper client installed on user terminal.
Story	**Given**: User navigated to the Login Screen. **When**: User enters the wrong User Name. **And** the user enters the wrong password. **And** user Click “login” button. **Then**: System verify user information. **display** error message for invalid username and password.