01. Class Diagram

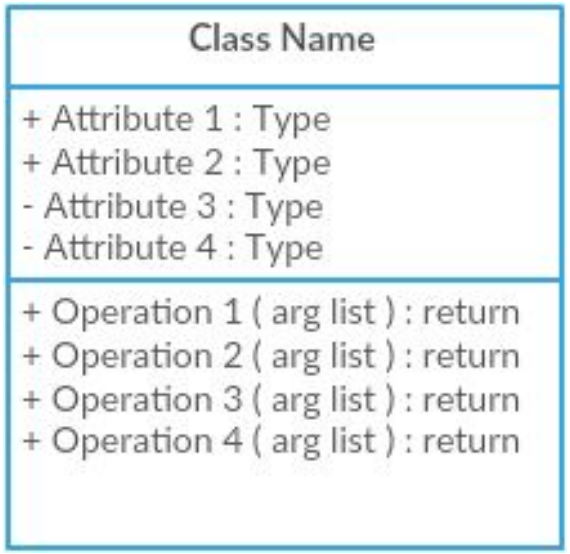
Class

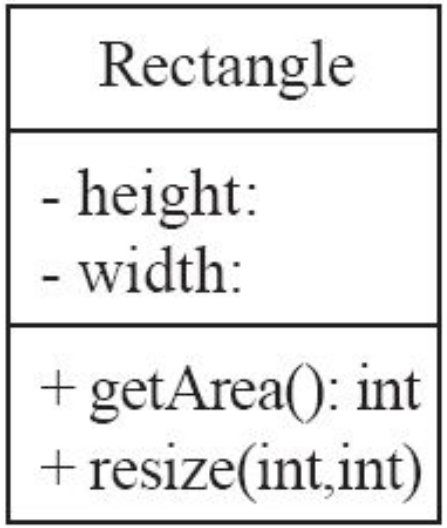
- Class is an entity which have
  - Data
  - Behavior
- Example Student
  - Data : roll no, Name, Mobile No, GPA
  - Behavior: add course, drop course, withdraw course

- All objects that exhibit same data and behavior can be formed into a class.
- Class are usually nouns in requirement document.

Main components of class

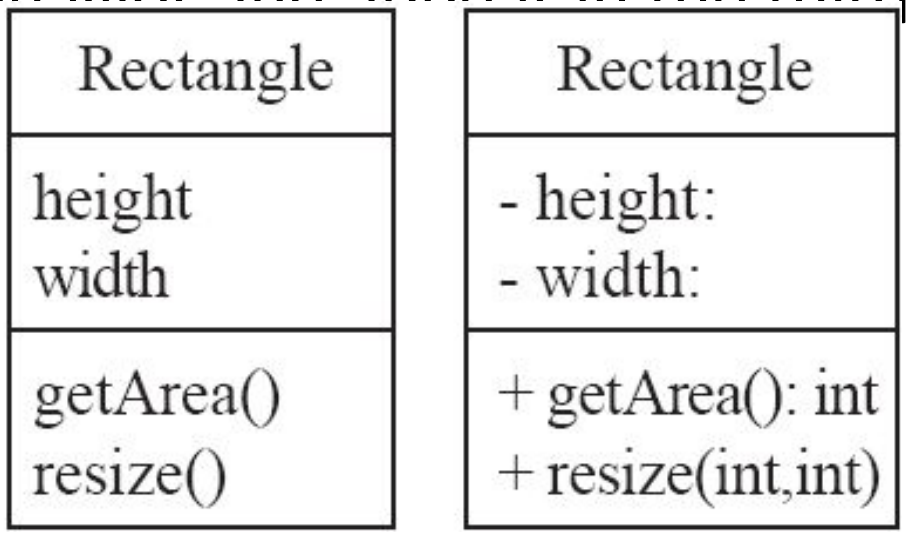
- Class
- Attributes
- Operations





Attributes and operations

- An attribute is a named property of a class that describes the object being modeled.
- In the class diagram, attributes appear in the second compartment just below the name compartment.
- Operations or methods are the functions of the class
- Operations describe the class behavior and appear in the third compartment.



What is Class Diagram

- A class diagram is a UML diagram type that describes a system by visualizing the different types of classes within a system and the kinds of static relationships that exist among them.
- It illustrates
  - the operations and attributes of the classes.
  - Relationship between the classes

Class Diagram Relationships

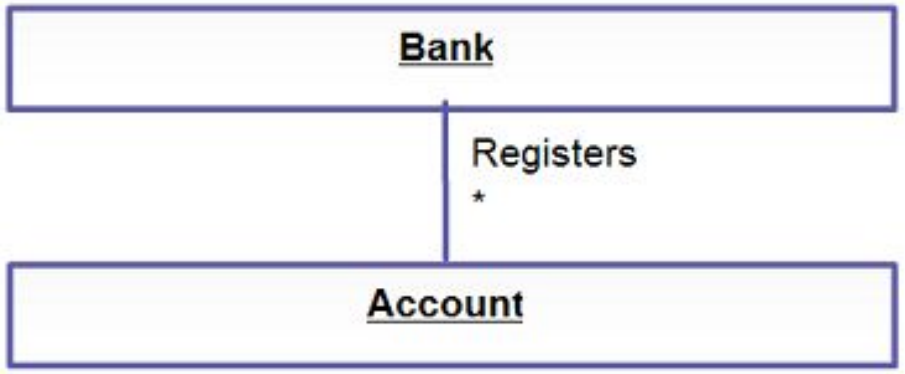
Class Diagram Relationship Type	Notation
Association	
Inheritance	
Aggregation	
Composition	

1. Associations

- An association relation is established when two classes are connected to each other in any way.
- For example:
- A bank registers account.
- An *association* is used to show how two classes are related to each other

Multiplicity

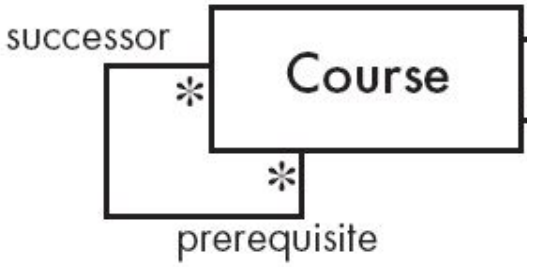
- An example of this kind of association is many accounts being registered by the bank. Hence, the relationship shows a star sign near the account class (one to many and many to many etc).
- Symbols indicating multiplicity are shown at each end of the association



- Each association can be labelled, to make explicit the nature of the association

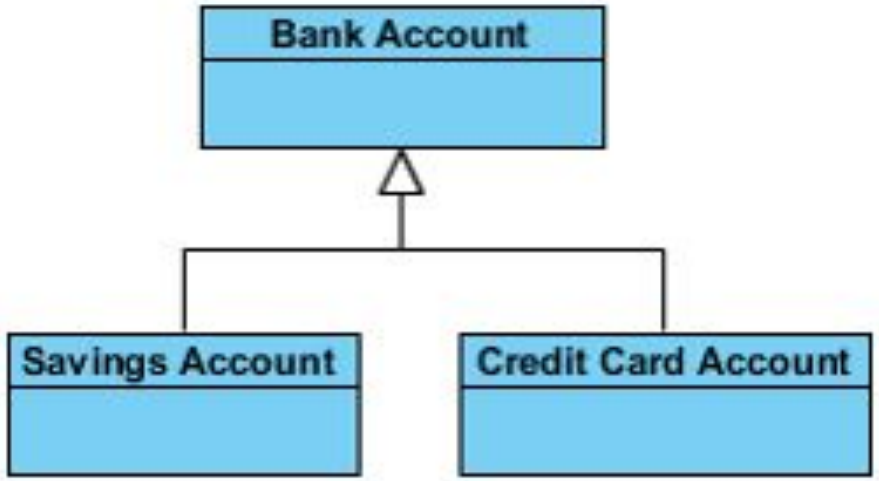
Reflexive Associations

- It is possible for an association to connect a class to itself



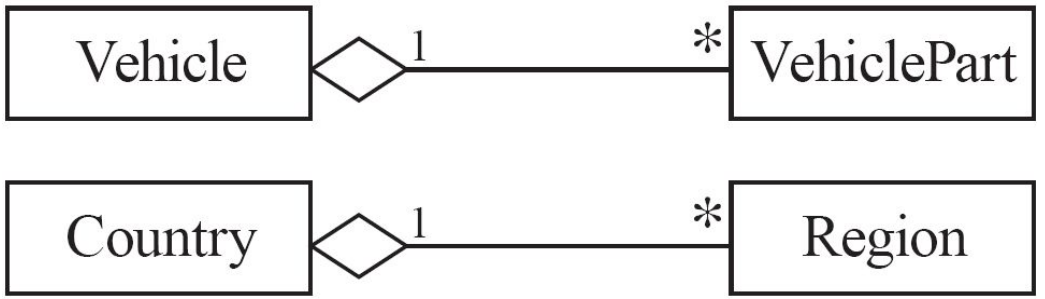
2. Generalization

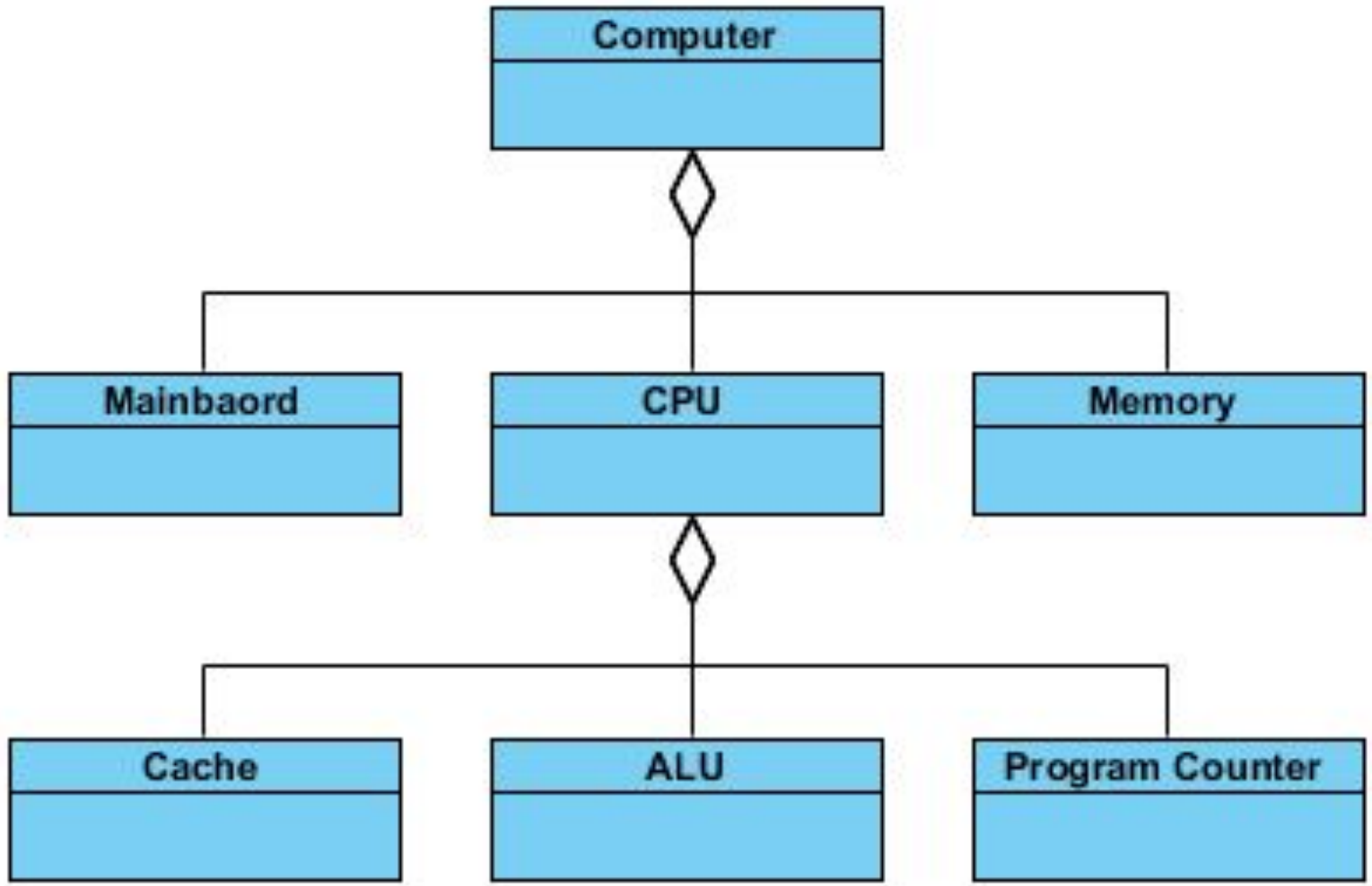
- Generalization is known as an “is a” relationship
- the child class is a type of the parent class
- Generalization is the ideal type of relationship that is used to showcase reusable elements in the class diagram.
- The child classes “inherit” the common functionality defined in the parent class.



3. Aggregation

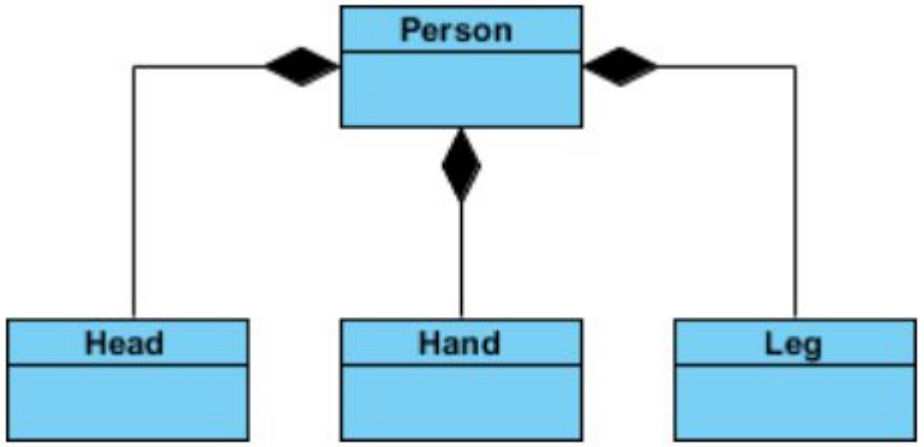
- It is also called a “**has a**” relationship.
- Aggregations are special associations that represent ‘part-of’ relationships.
  - While if A and B are associated with each other, such that B can exist without being associated with A, then this association in known as Aggregation.




Computer
Mainbaord
CPU
Memory
Cache
ALU
Program Counter

4. Composition

- The composition is a variation of the aggregation relationship.
- A composition is a strong kind of aggregation
  - if the aggregate is destroyed, then the parts
  - If A and B two classes are related to each ot
    destroyed, then the association between two objects is known as Composition. An example is Building and room.



How to draw class diagram

Step 1: Identify the class names

Step 2: Identify the attributes and operations of the class

Step 3: Distinguish relationships

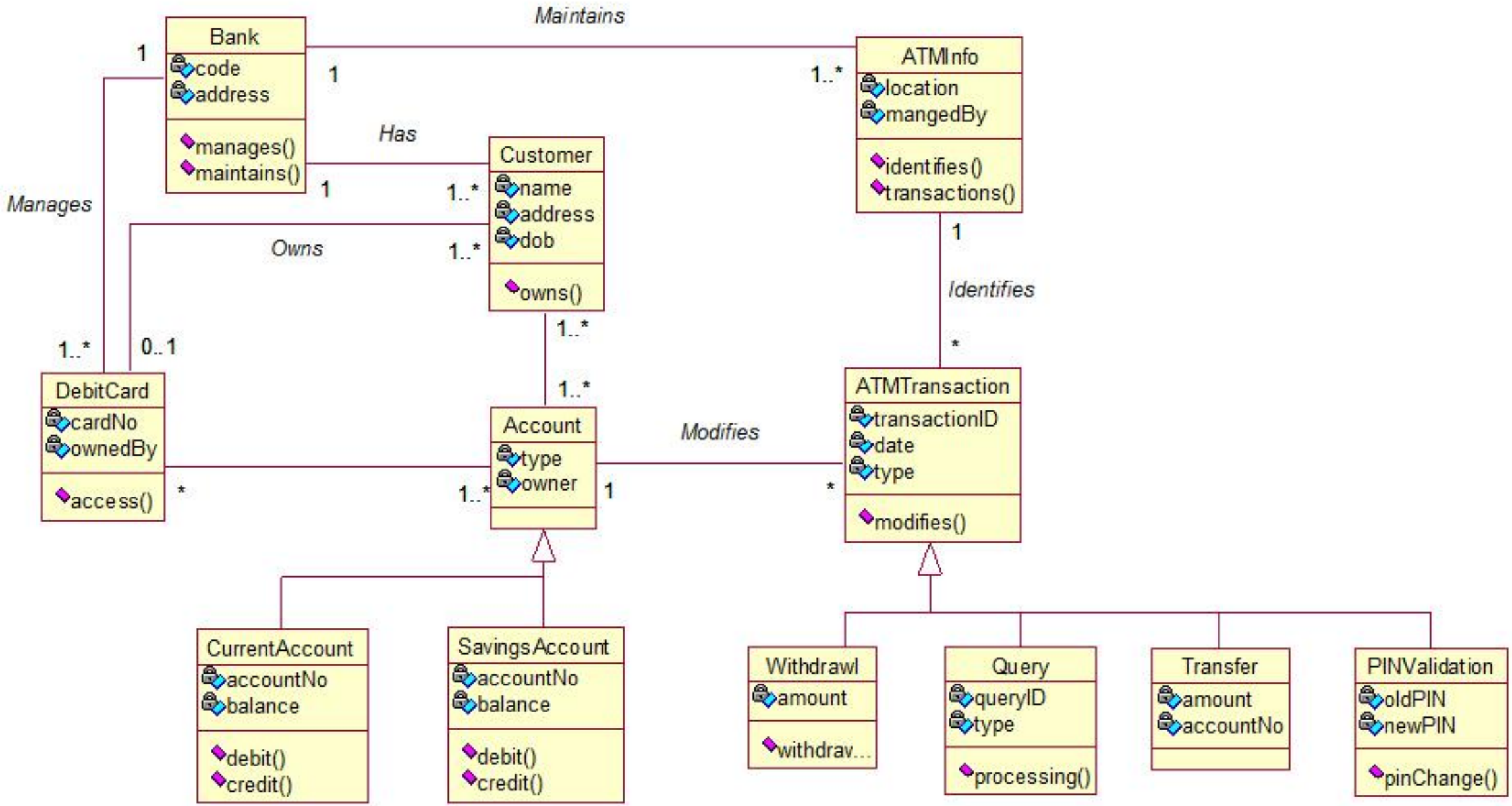
Next step is to determine how each of the classes or objects are related to one another**Step**

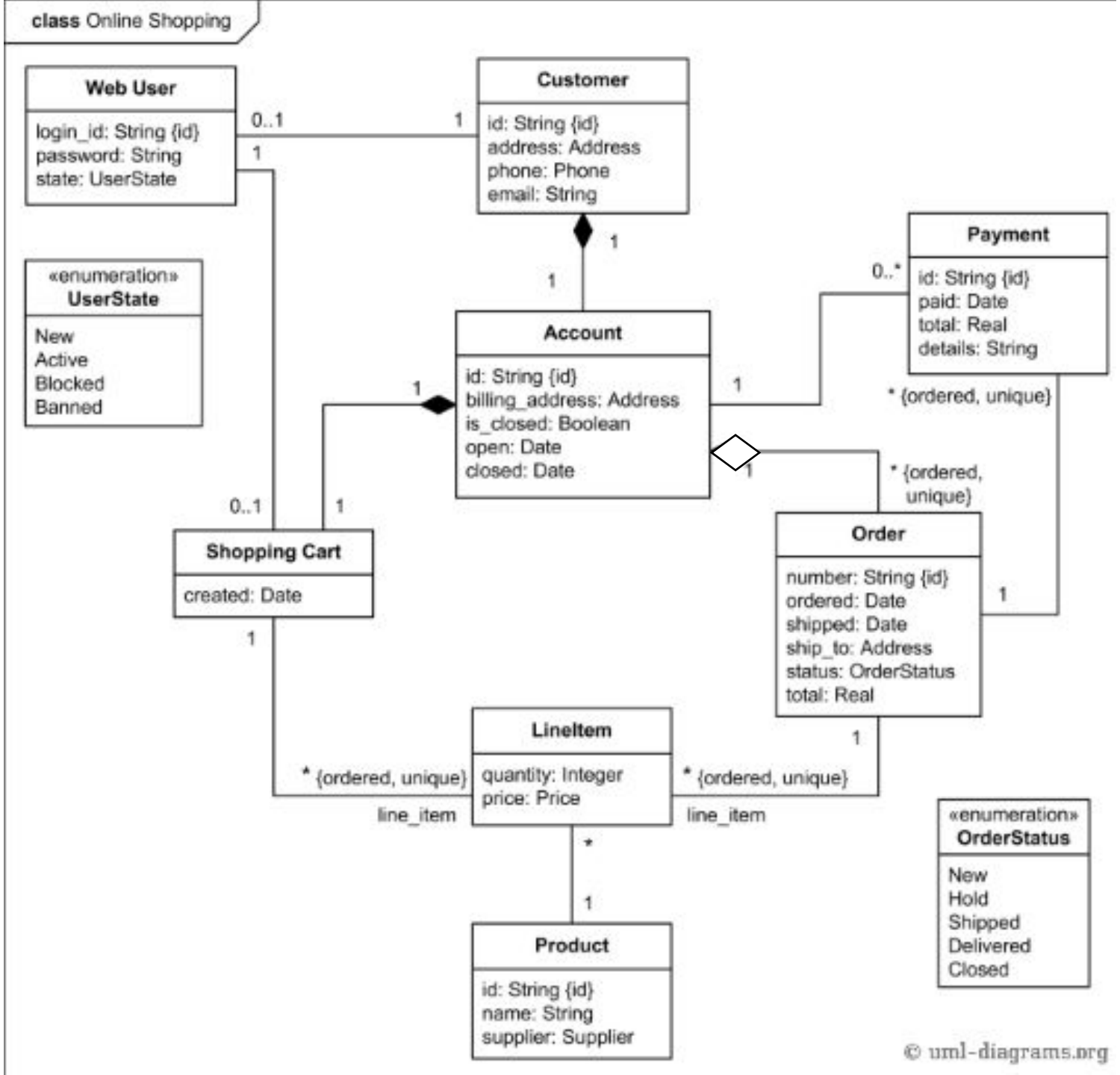
Step 4: Create the Structure

First, add the class names and link them with the appropriate connectors. You can add attributes and functions/ methods/ operations later.

Points to Remember

- The name of the class diagram should be meaningful to describe the aspect of the system.
- Each element and their relationships should be identified in advance.
- Responsibility (attributes and methods) of each class should be clearly identified
- Use notes whenever required to describe some aspect of the diagram.
- At the end of the drawing it should be understandable to the developer/coder.
- Finally, before making the final version, the diagram should be drawn on plain paper and reworked as many times as possible to make it correct.

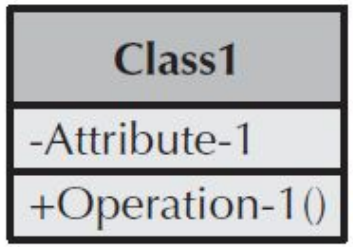

Maintains
Bank
code
address
manages()
maintains()
ATMInfo
location
mangedBy
identifies()
transactions()
Has
Customer
name
address
dob
owns()
Manages
Owns
Identifies
DebitCard
cardNo
ownedBy
access()
Account
type
owner
Modifies
ATMTransaction
transactionID
date
type
modifies()
CurrentAccount
accountNo
balance
debit()
credit()
SavingsAccount
accountNo
balance
debit()
credit()
Withdrawl
amount
withdrav...
Query
queryID
type
processing()
Transfer
amount
accountNo
PINValidation
oldPIN
newPIN
pinChange()


class Online Shopping
Web User
login_id: String {id}
password: String
state: UserState
0..1
1
Customer
id: String {id}
address: Address
phone: Phone
email: String
1
1
«enumeration»
UserState
New
Active
Blocked
Banned
Payment
0..*
id: String {id}
paid: Date
total: Real
details: String
Account
id: String {id}
billing_address: Address
is_closed: Boolean
open: Date
closed: Date
1
1
* {ordered, unique}
1
* {ordered, unique}
0..1
1
Shopping Cart
created: Date
Order
number: String {id}
ordered: Date
shipped: Date
ship_to: Address
status: OrderStatus
total: Real
1
1
1
LineItem
quantity: Integer
price: Price
* {ordered, unique}
line_item
* {ordered, unique}
line_item
*
1
Product
id: String {id}
name: String
supplier: Supplier
«enumeration»
OrderStatus
New
Hold
Shipped
Delivered
Closed
© uml-diagrams.org

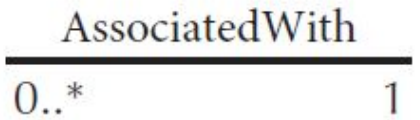
CLASS DIAGRAMS

- A *class diagram* is a *static model* that shows
  - the classes and the relationships among classes that remain constant in the system over time.
- The class diagram depicts classes, which include
  - both behaviors and states, with the relationships between the classes.

Class Diagram Syntax

A class: • Represents a kind of person, place, or thing about which the system will need to capture and store information. • Has a name typed in bold and centered in its top compartment. • Has a list of attributes in its middle compartment. • Has a list of operations in its bottom compartment. • Does not explicitly show operations that are available to all classes.	
An attribute: • Represents properties that describe the state of an object. • Can be derived from other attributes, shown by placing a slash before the attribute's name.	attribute name /derived attribute name

Class Diagram Syntax

An operation: • Represents the actions or functions that a class can perform. • Can be classified as a constructor, query, or update operation. • Includes parentheses that may contain parameters or information needed to perform the operation.	operation name ()
An association: • Represents a relationship between multiple classes or a class and itself. • Is labeled using a verb phrase or a role name, whichever better represents the relationship. • Can exist between one or more classes. • Contains multiplicity symbols, which represent the minimum and maximum times a class instance can be associated with the related class instance.	

Class Diagram Syntax

A generalization: • Represents a-kind-of relationship between multiple classes.	
An aggregation: • Represents a logical a-part-of relationship between multiple classes or a class and itself. • Is a special form of an association.	
A composition: • Represents a physical a-part-of relationship between multiple classes or a class and itself • Is a special form of an association.	

Attributes

- Attributes are properties of the class about which we want to capture information
- At times, you might want to store ***derived attributes***, which are attributes that can be calculated or derived;
  - these special attributes are denoted by placing a slash (/) before the attribute's name.
- Notice how the person class contains a derived attribute called /age,
  - which can be derived by subtracting the patient's birth date from the current date.

Attribute Visibility

- Visibility relates to the level of information hiding to be enforced for the attribute.
  - The visibility of an attribute can be public (+), protected (#), or private (−).
- A ***public*** attribute is one that is not hidden from any other object.
  - As such, other objects can modify its value.
- A ***protected*** attribute is one that is hidden from all other classes except its immediate subclasses.
- A ***private*** attribute is one that is hidden from all other classes.
  - The default visibility for an attribute is normally private.

Operations

- *Operations* are actions or functions that a class can perform .
- The functions that are available to all classes
  - (e.g., create a new instance, return a value for a particular attribute, set a value for a particular attribute, delete an instance) are not explicitly shown within the class rectangle.
  - Instead, only operations unique to the class are included, such as the cancel without notice operation in the Appointment class and the calculate last visit operation in the Patient Class

Visibility of Operations

- As with attributes, the visibility of an operation can be designated
  - public, protected, or private.
- The default visibility for an operation is normally public.

Relationships

- A primary purpose of a class diagram is to show the relationships, or associations, that classes have with one another.
- When multiple classes share a relationship (or a class shares a relationship with itself), a line is drawn and labeled with either
  - the name of the relationship or the roles that the classes play in the relationship.

Multiplicity

Relationships also have *multiplicity*, which documents how an instance of an object can be associated with other instances.

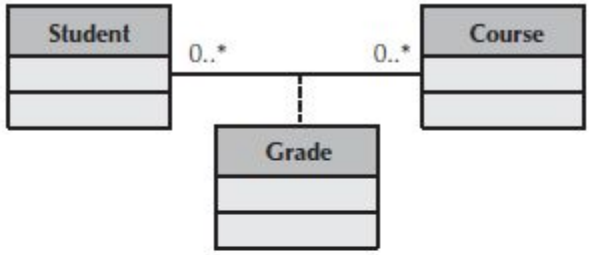
Exactly one	1	Department — 1 — Boss	A department has one and only one boss.
Zero or more	0..*	Employee — 0..* — Child	An employee has zero to many children.
One or more	1..*	Boss — 1..* — Employee	A boss is responsible for one or more employees.
Zero or one	0..1	Employee — 0..1 — Spouse	An employee can be married to zero or one spouse.
Specified range	2..4	Employee — 2..4 — Vacation	An employee can take from two to four vacations each year.
Multiple, disjoint ranges	1..3,5	Employee — 1..3,5 — Committee	An employee is a member of one to three or five committees.

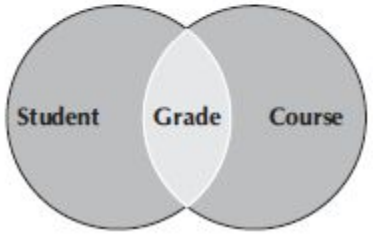
Association Classes

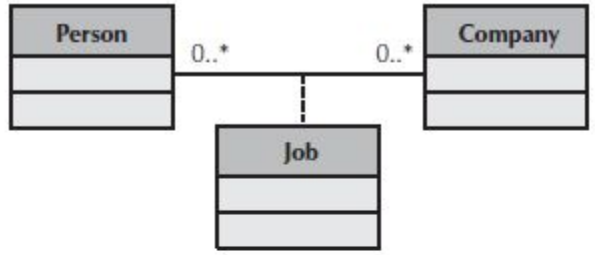
- There are times when a relationship itself has associated properties,
  - especially when its classes share a many-to-many relationship.
  - In these cases, a class called an *association class* is formed, which has its own attributes and operations.
  - It is shown as a rectangle attached by a dashed line to the association path, and
  - the rectangle's name matches the label of the association.

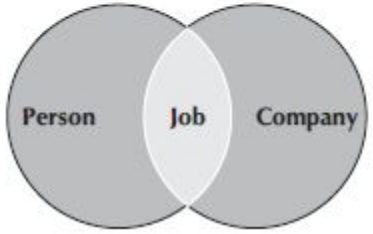
Example

- the Grade idea is really an intersection of the Student and Course classes, because a grade exists only at the intersection of these two ideas.
- a job may be viewed as the intersection between a Person and a Company.

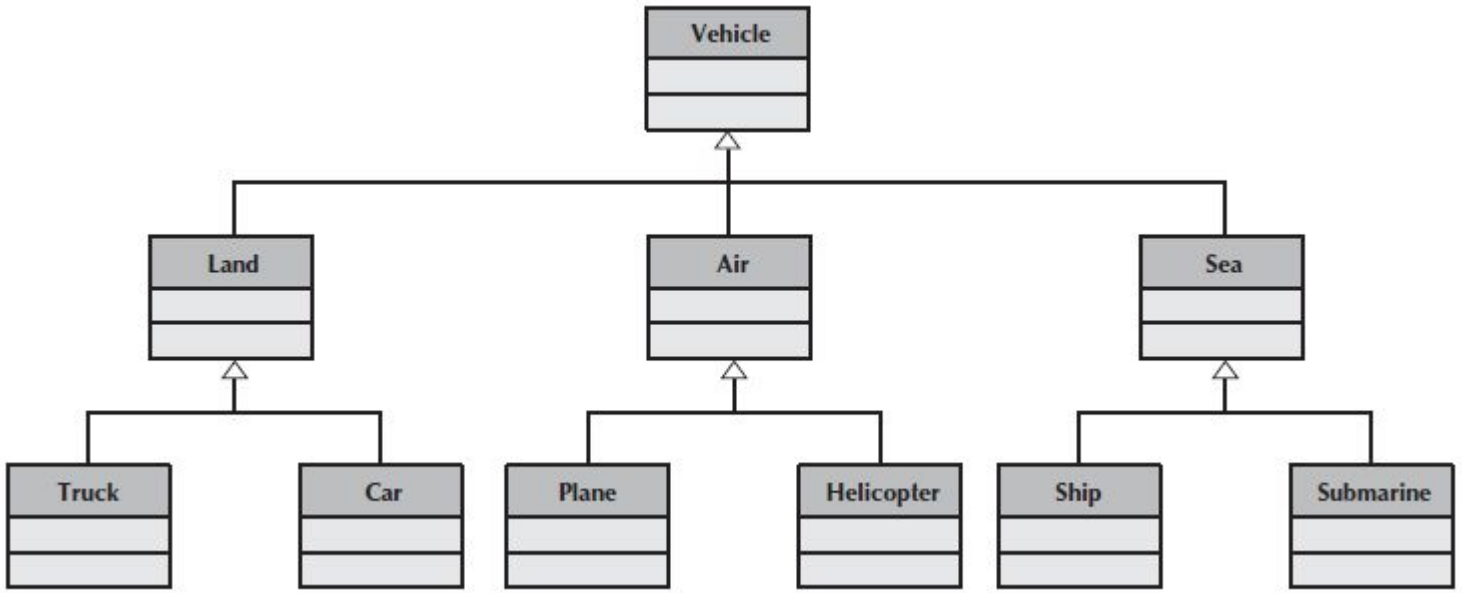




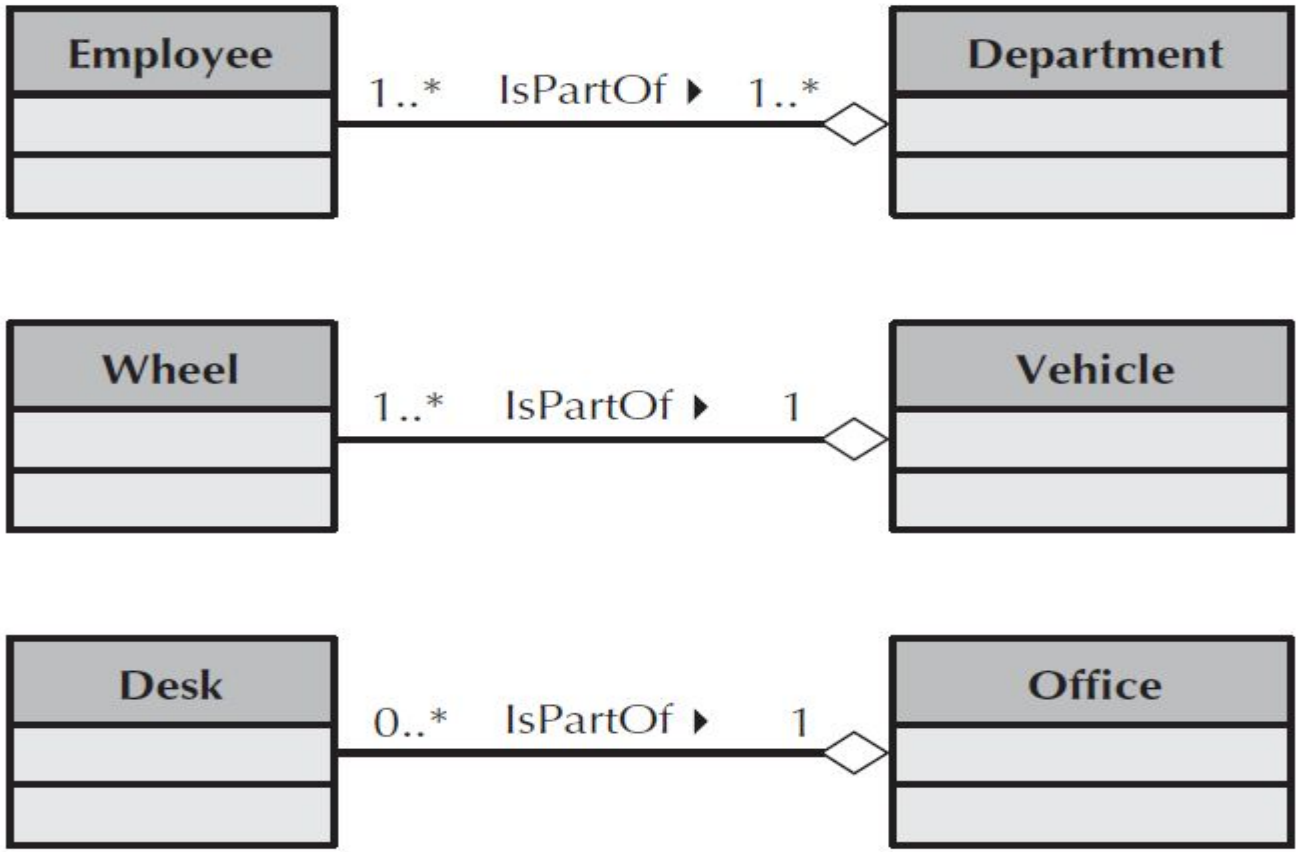




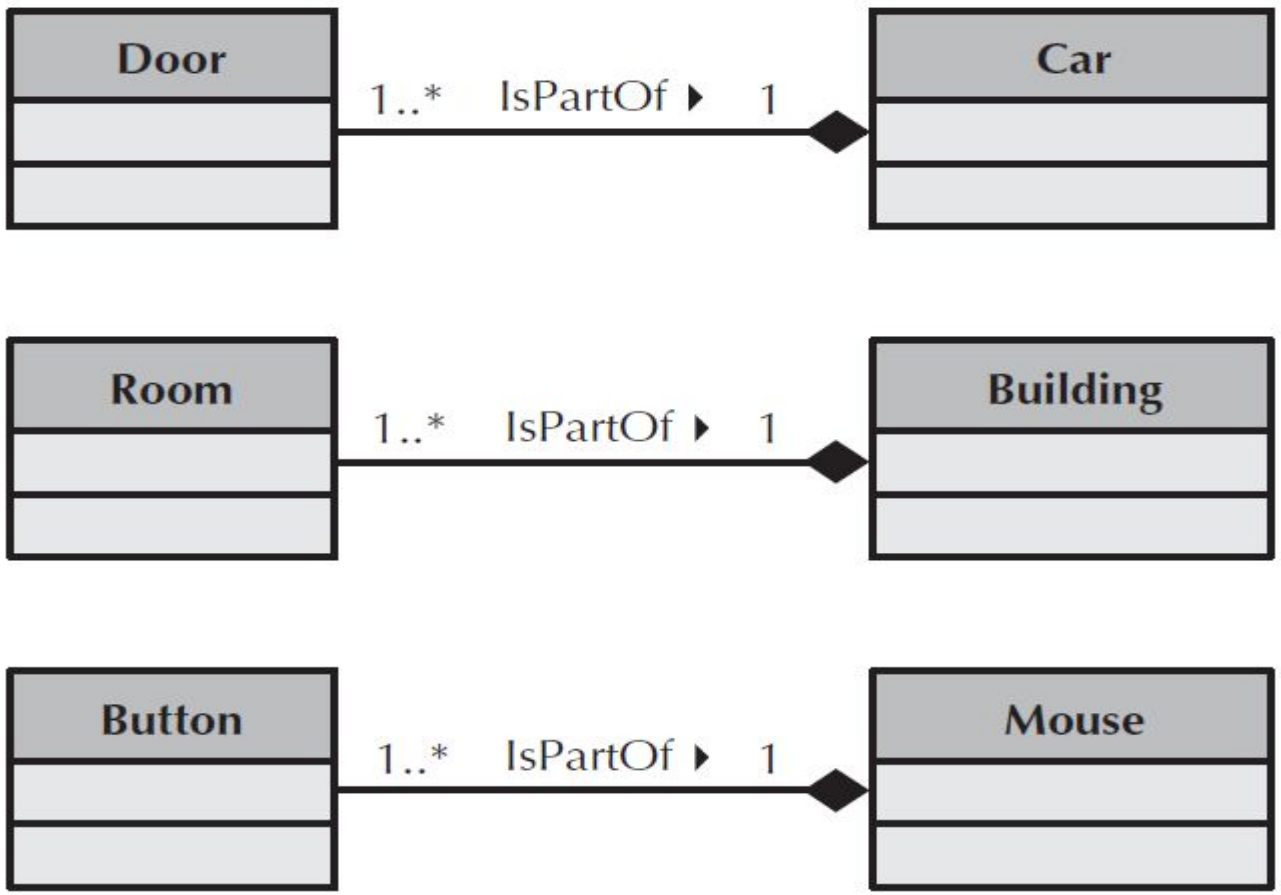
Generalizations



Aggregation



Composition

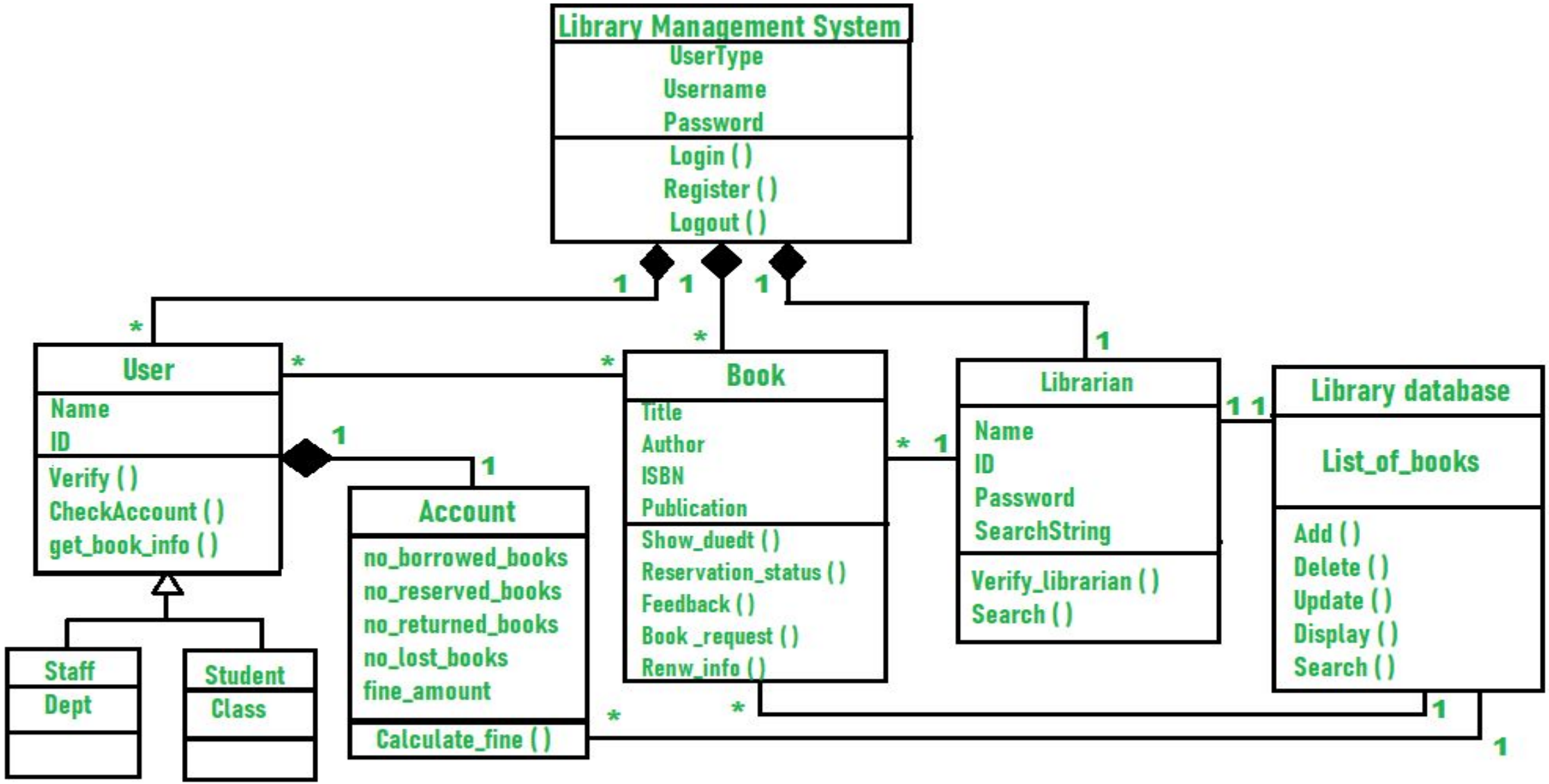


Aggregation Vs Composition

- Aggregation is used to portray logical part.
- Logical implies that it is possible for a part to be associated with multiple wholes or that is relatively simple for the part to be removed from the whole.
  - For example,
    - An instance of the Employee class IsPartOf an instance of at least one instance of the Department class,
    - an instance of the Wheel class IsPartOf an instance of the Vehicle class, and
    - an instance of the Desk class IsPartOf an instance of the Office class.

Obviously, in many cases an employee can be associated with more than one department, and it is relatively easy to remove a wheel from a vehicle or move a desk from an office.

- Composition is used to portray a physical part of relationships and is shown by a black diamond.
- *Physical* implies that the part can be associated with only a single whole.
  - For example
    - an instance of a door can be a part of only a single instance of a car,
    - an instance of a room can be a part of an instance only of a single building, and
    - an instance of a button can be a part of only a single mouse



CLASS DIAGRAM FOR LIBRARY MANAGEMENT SYSTEM