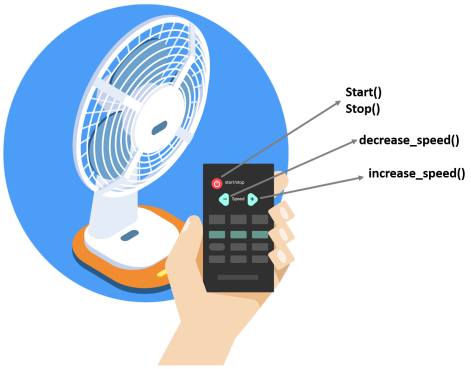
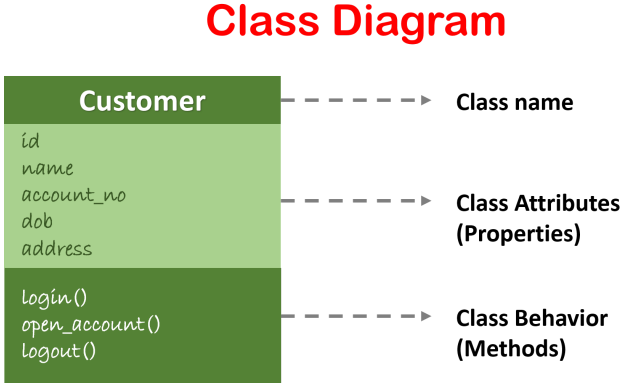
# Class and Object Diagrams

## Class Diagrams

- In the example of a virtual bank, we have seen that the bank consists of the objects of mainly three types: Employee, Customer, and Bank Account.
- We have also seen that the objects of these classes have certain properties, which are as shown
- In the real world, if you observe, you can see that all the objects have some properties as well as functionalities.
- For example: a **fan has properties such as color, type, no_of_blades, and made.**
- With these attributes it also has some functionalities such as **start()**, **stop()**, **increase_speed()**, **decrease_speed()**.



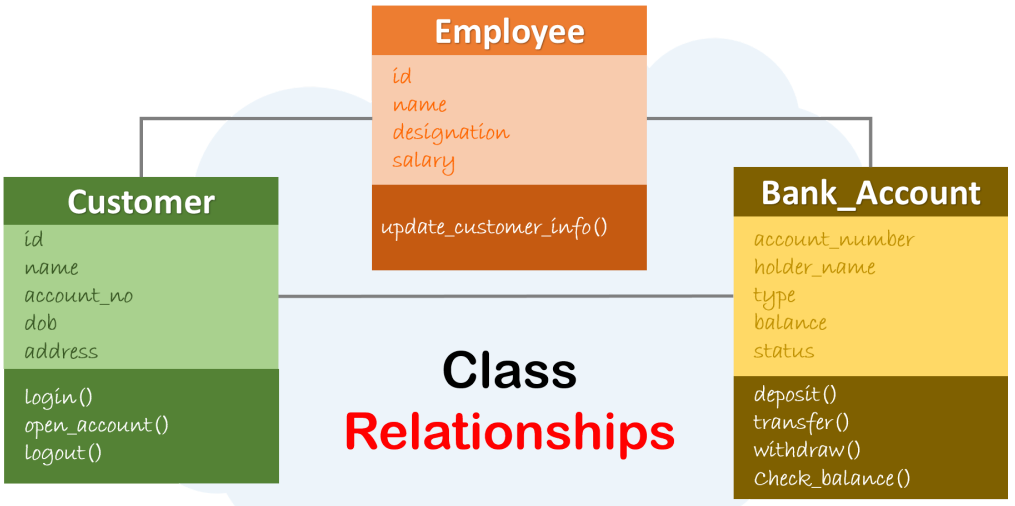

- We can see that in our example of a bank, a Customer can Login() to the bank system and can open_accont() in the bank.
  Similarly, the employee can update the info of customers in the bank database.
- These functionalities need to be expressed in the software designs also, so we keep these functionalities with the properties of the class. And this leads to the concept of a class diagram.
- In a class diagram a class is represented with the help of a rectangular box having *three* partitions:
  - The first partition contains the class name.
  - The second partition contains all the attributes of the class.
  - The third partition contains all the functionalities of the class.



- Time to define the class diagram:
  - **A class diagram in the Unified Modeling Language (UML) is a type of structural diagram that describes the structure of a system by showing the System's Classes, their Attributes, Operations (or methods), and the relationships among objects**

## Relationships

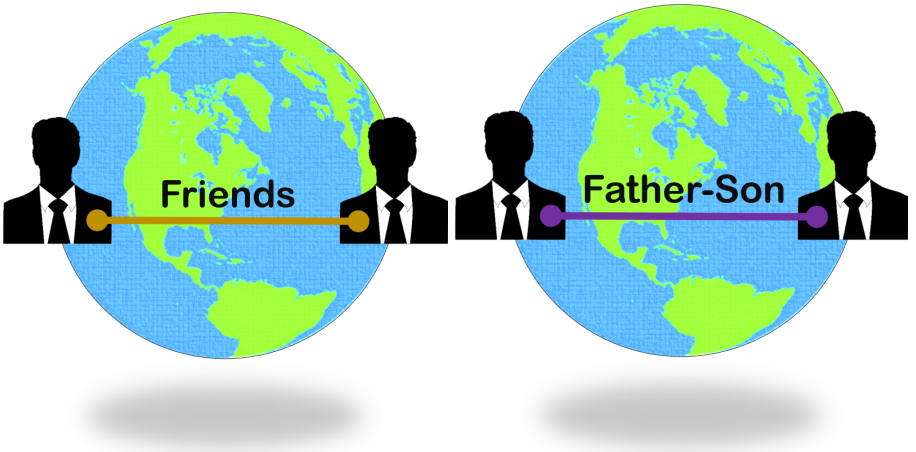
- Till the previous section, we learned to figure out different classes in our system, then we represented these classes with the help of class diagrams.
- But there was one strange thing, these classes were independent of each other which is quite a rarest thing in the real world.
- In the real world, the objects are related to each other in one or the other way and the same should also be reflected in our class diagrams.
- There can be a connection between CUSTOMER and ACCOUNTS, EMPLOYEES can be related to CUSTOMER, and EMPLOYEES might have some relation with ACCOUNTS as well.
- The connection between the two classes in a class diagram is called a Relationship. A class may be involved in one or more relationships with other classes.



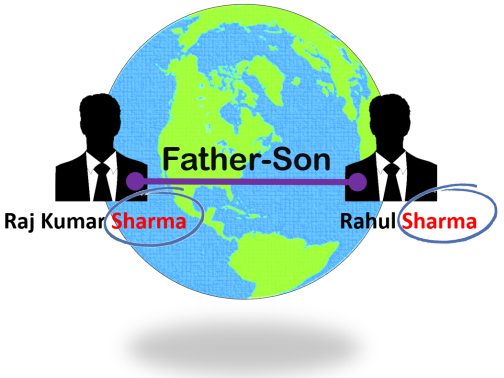

**A class may be involved in one or more relationships with other classes.**

Virtual XYZ Bank

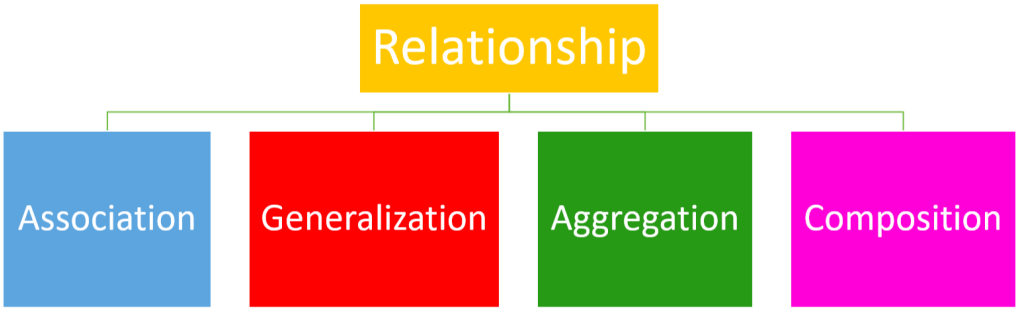
- Now let's take an example of two people in the real world. There are many ways in which these two persons can be related to each other, for example, they can be friends or they can be father and son.
- Now their actions will be different depending on the type of relationship they share.




- In case, they share a father and son relationship, the son will inherit the surname of the father which would have never been possible in case of friendship.

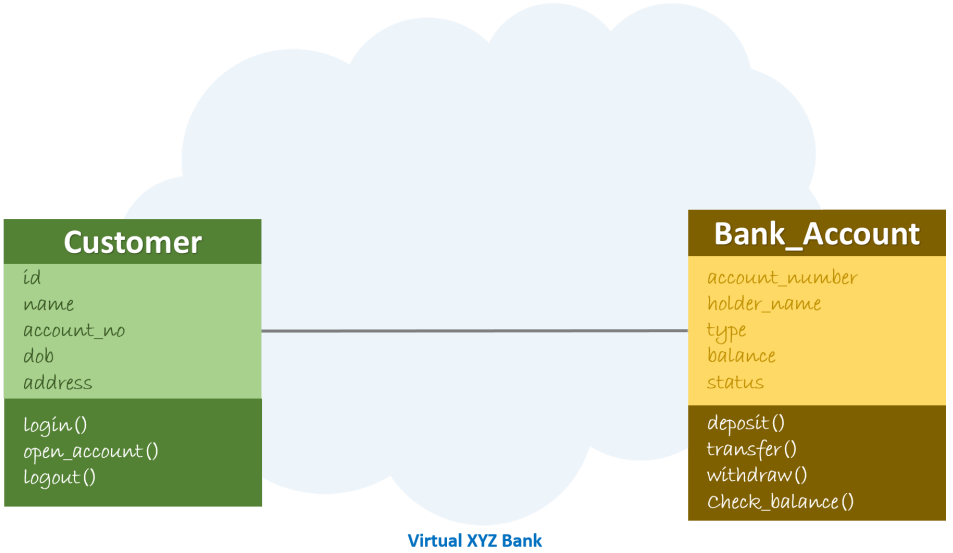



- Similarly, in class diagrams, two classes can be related to each other in many different ways. Hence we have different types of relationships to represent different relations.
- In the class diagram, the classes can be related to each other through Association, Generalization, Aggregation, or Composition.



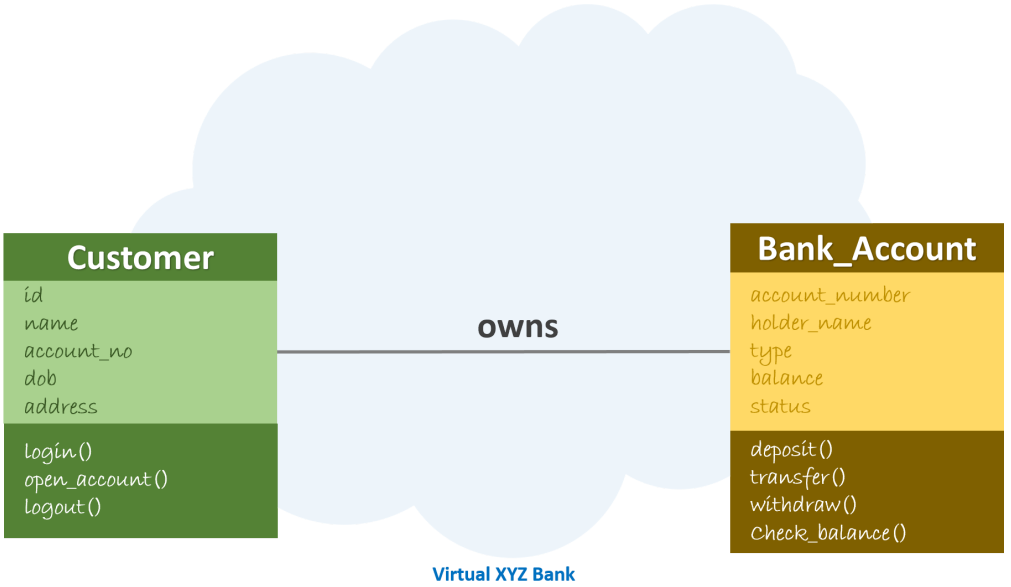
## Associations

- In class diagrams, we can connect two classes with a solid line.
- This relationship is known as Association and shows that one class is associated with other classes.

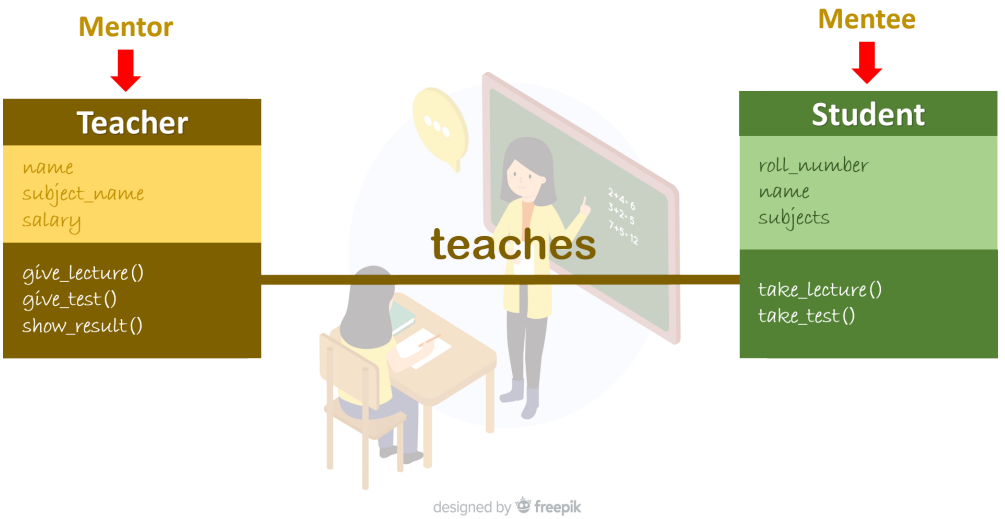



- Now we are aware of the fact that the classes actually do not exist, but the objects created from these classes do have the existence. So we must understand one thing here, that it is actually the objects that are connected to each other but in class diagrams, we connect classes to show how the objects of these classes will be referencing each other.
- In this diagram, we can see that Customer is associated with Bank Account. But the question is how?
- I will try to answer this, but first, let's think of the real world again. In our real world, every relationship has some name.
- For example, suppose there are a teacher and a student. Now a teacher is associated with a student. But when we connect these two through an association, we need to give some name to this association. Let's name this as teaches.

- Now we can easily see that a TEACHER is associated with a STUDENT through teaching. This is called the labeling of the association through Association Names.
- So, Association Name is a verb placed in the middle of association between two classes.
- Here in our example Customer owns the bank account, and hence we can label this association with the verb **OWNS**.

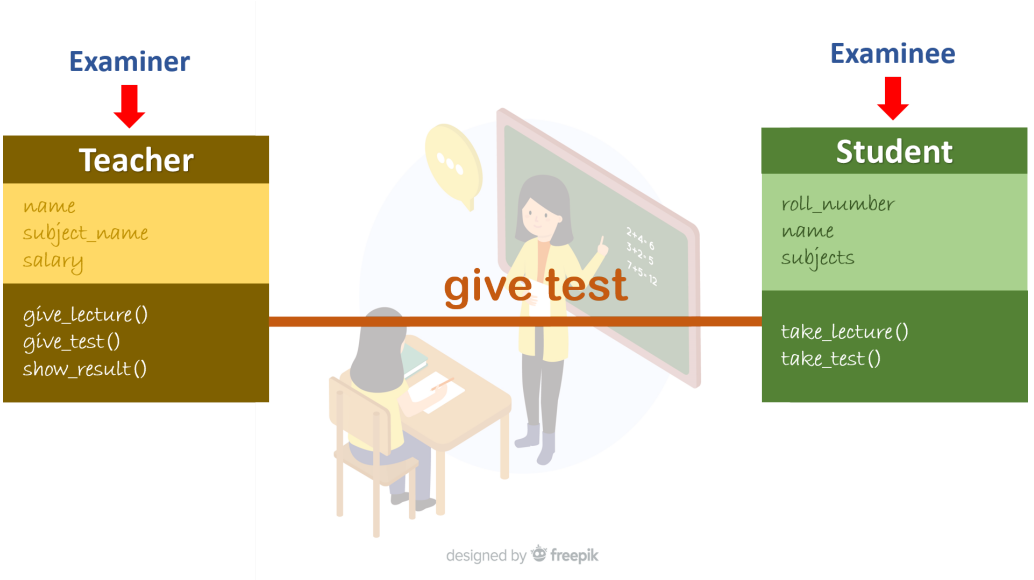



- Now, when you have learned to label association with association name, one more type of labeling can be done on associations i.e. Role names.
- Let's get back to the same example again,
- Here we can see that teacher teaches students. Also, in other words, we can say that when a teacher is teaching a student, then teacher is playing a role of mentor and the student is playing the role of mentee.

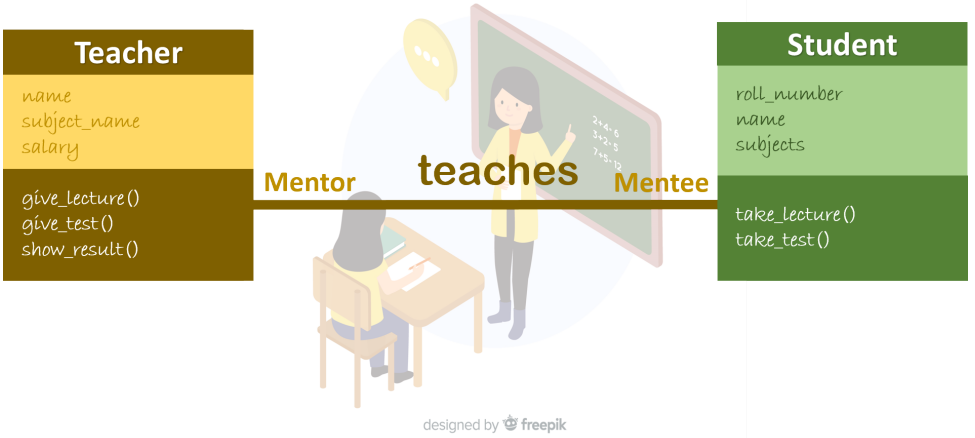



- Now suppose that the teacher and student have another association, wherein a teacher gives the test to students.

- In this association, the teacher is playing the role of EXAMINER whereas the STUDENT is EXAMINEE.



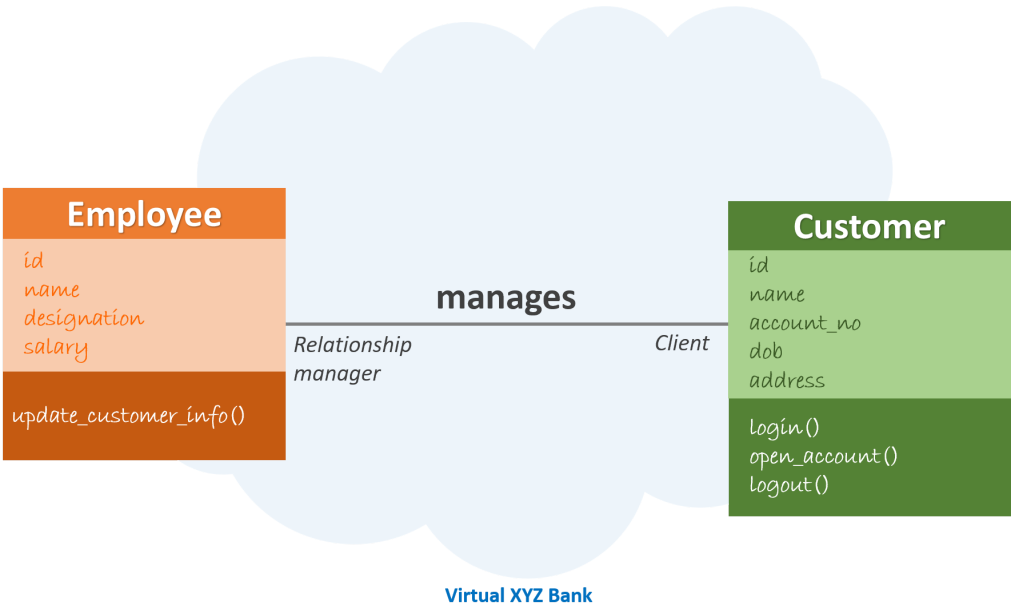

- So a teacher is playing a different role in different associations with the student.
- We can reflect these roles in class diagrams as well. We just put the role-name at the class end as shown in these images.
- Here in this association, the teacher is a mentor and the student is the mentee.




- And in this association Teacher is an examiner and the student is an examinee.


- We can add these role-names to our bank example as well.

- As we can notice here that when an employee manages the customer, the employee is actually a relationship manager, and the customer is the client.

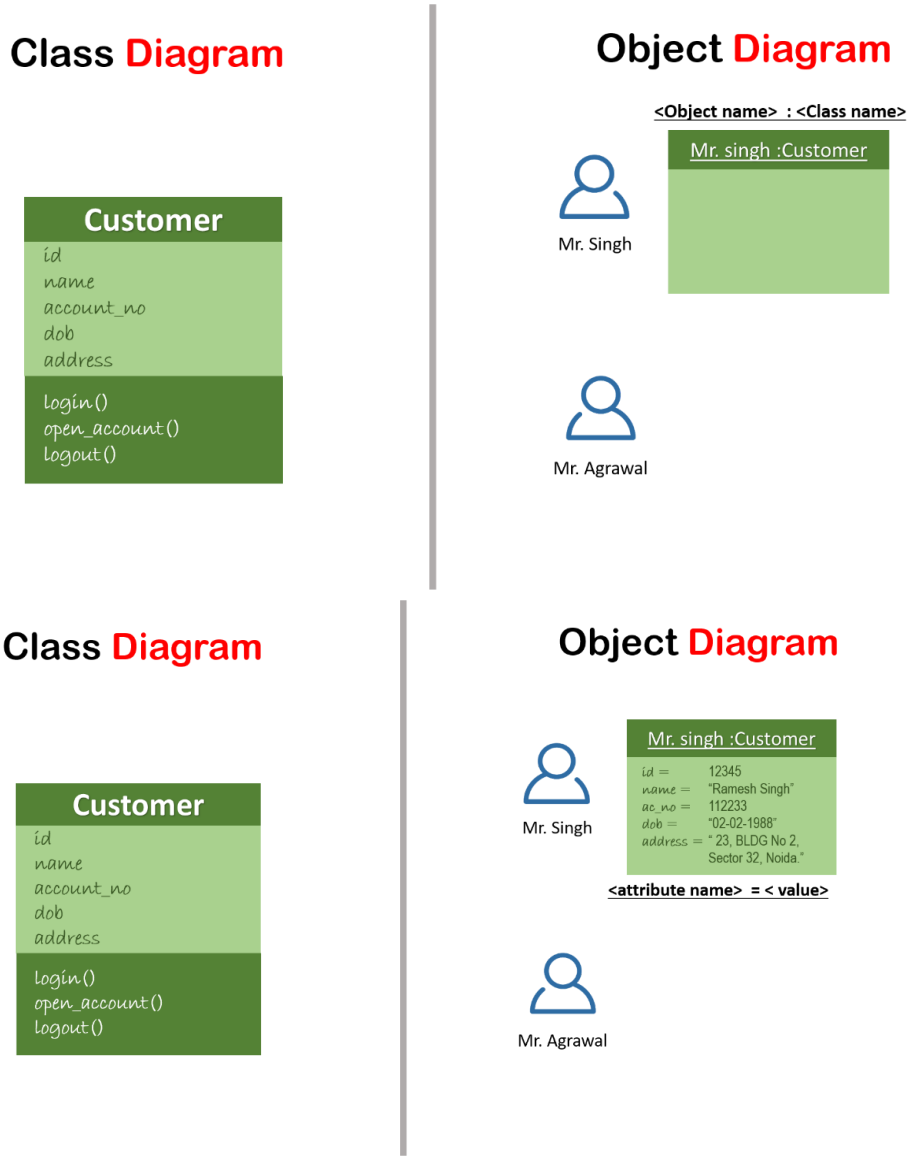



- Also, we can see in the class diagram that when the customer owns the bank account, the customer is the OWNER as mentioned in the association.

## Object Diagrams

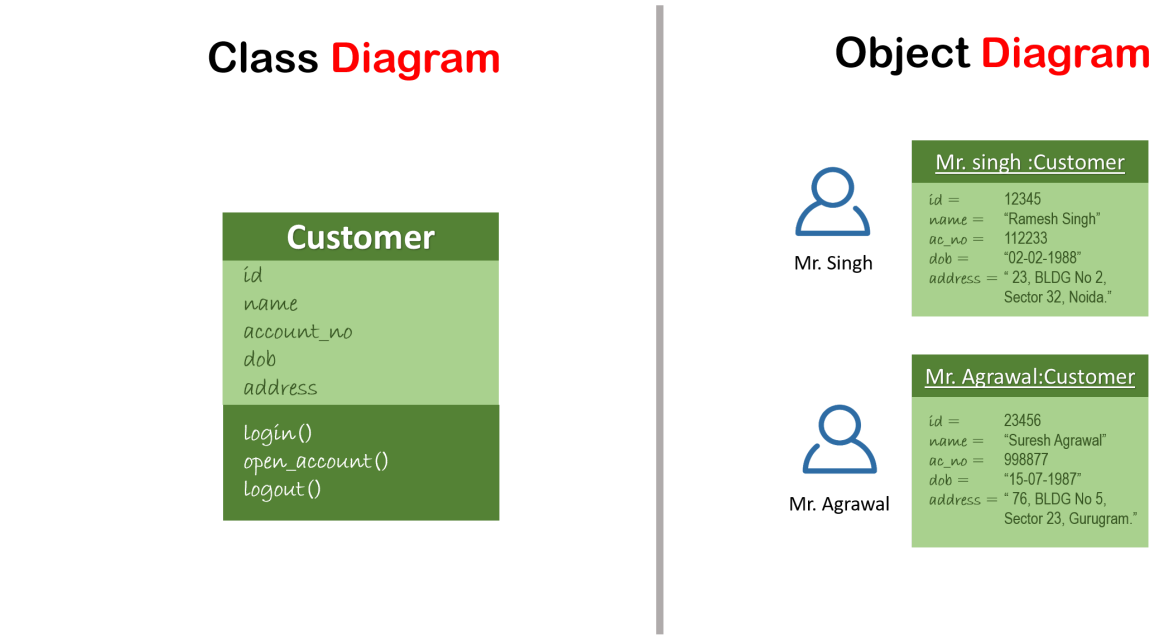
- We are ready to design any class using UML:
- But we have also learned that class is just a concept, in the real world, it is the objects that are created from it that exist.
- Now the question is, Is there any way of representing objects in UML? The answer is yes!
- We can use OBJECT DIAGRAMS to show objects in software designs.
- Let's check out how?
- In our example, the class customer was shown with the help of a Customer class diagram.
- And Mr. Singh and Mr. Agrawal are the two objects of class Customer.
- Now we can create an object diagram for Mr. Singh by creating a rectangular box having two partitions.
- The first partition contains the object name and class name separated by :

- *Here Mr. Singh is the object of the class Customer, so we have written Mr. Singh: Customer.* **Also, note that this is underlined.**

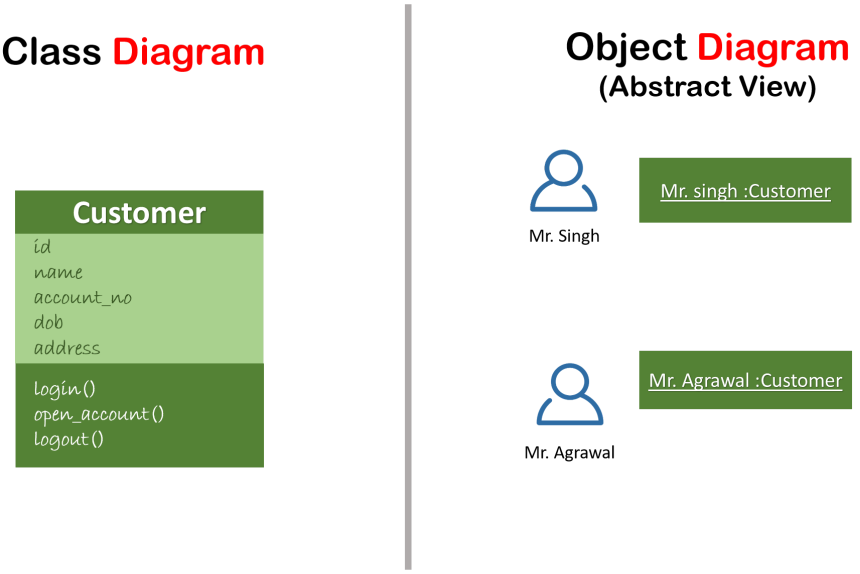



- In the second partition, we write the name of the attributes for the object and their values, separated by assignment.
- So, this is our object diagram of Mr. Singh.

- Similarly, we can have an object diagram for Mr. Agrawal as well.

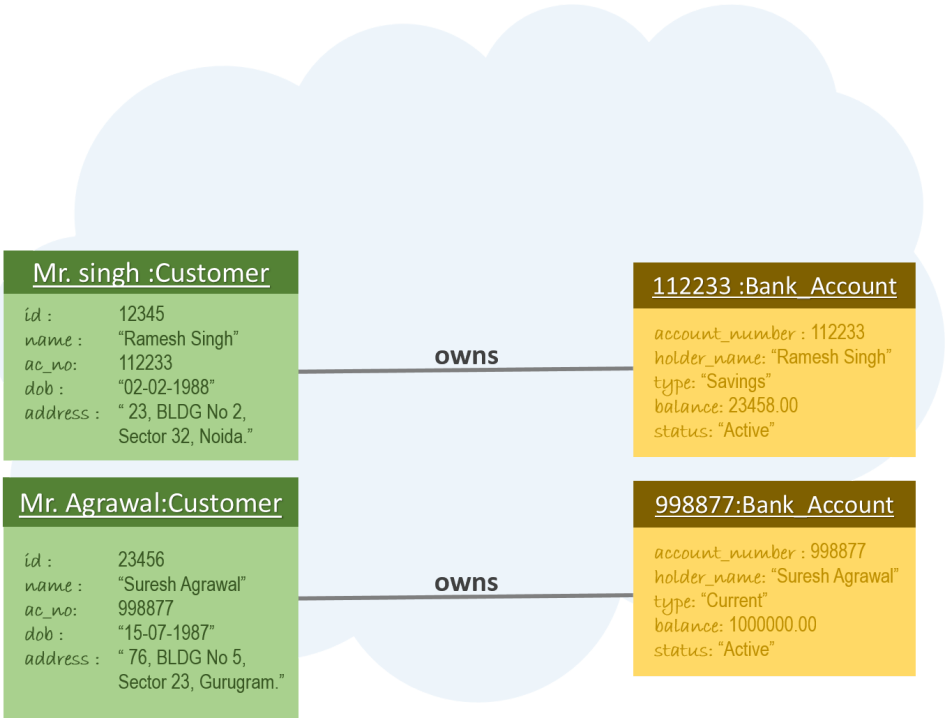



- We can also use slight abstract convention for showing our objects using only the first partition of the object diagram as shown:

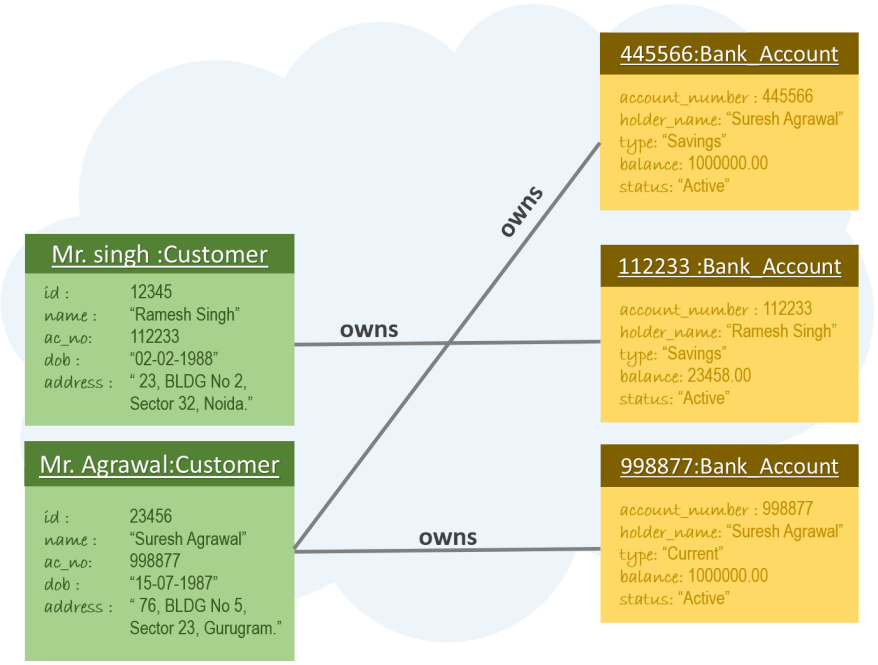


## Multiplicity

- We saw that our bank has two customers, Mr. Singh and Mr. Agrawal, now we can represent them with the help of this object diagram.
- Now if they own accounts in the bank, it can be shown like this:

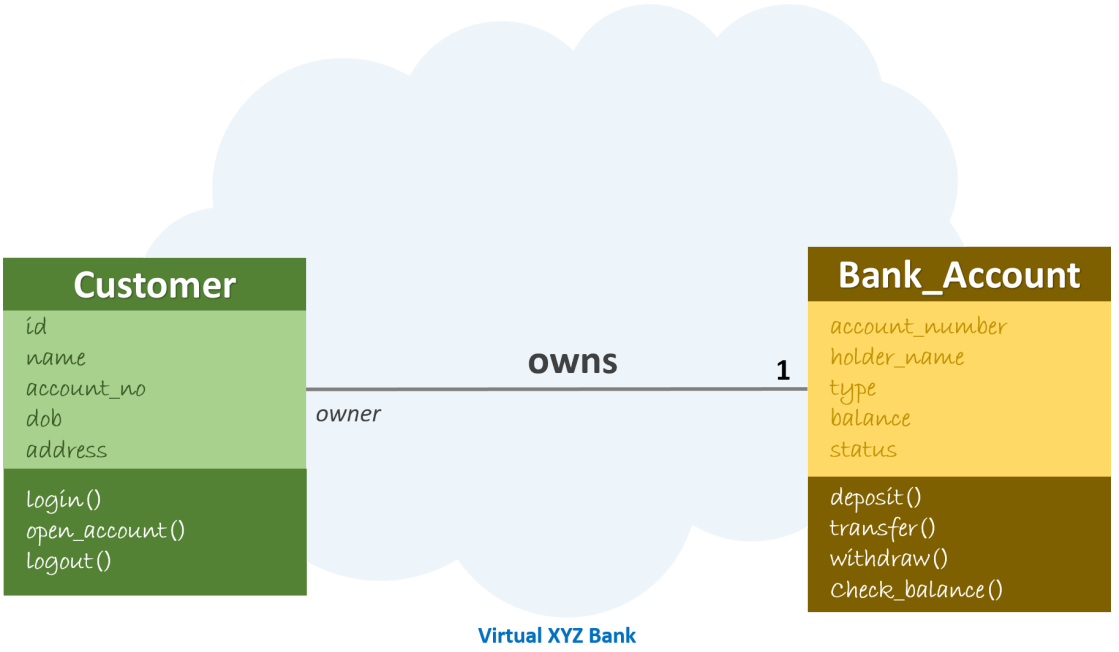



- Here we can see that one customer has one account, but our object diagram can be like this as well:

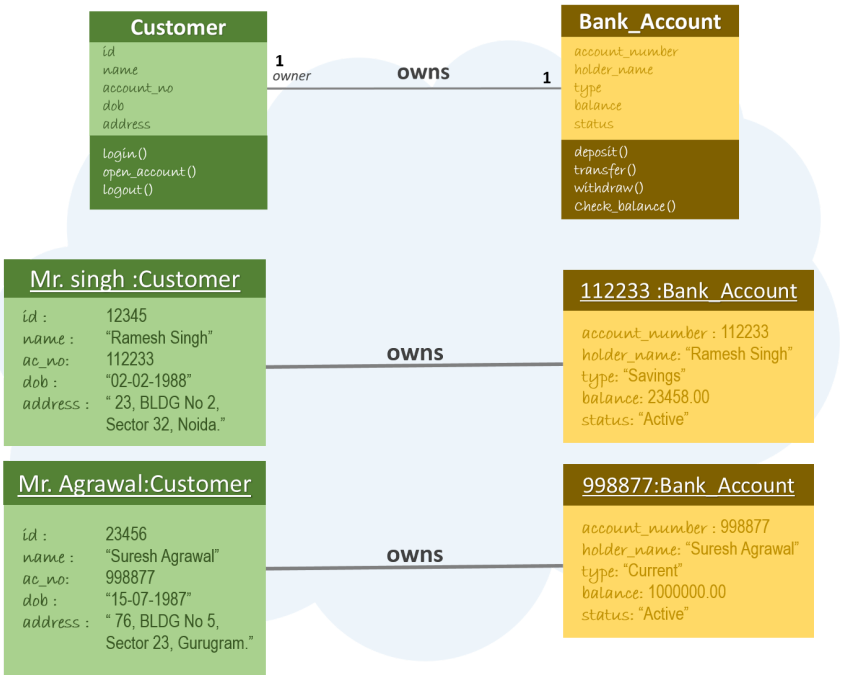



- Here you can notice that Mr. Agrawal owns two bank accounts whereas Mr. Singh owns only one.
- Now suppose that the bank doesn't allow this or in this bank, a customer can have only one account, in that case, this object diagram is invalid.
- Now the question is how can we let our class diagram speak this thing out.

- Can we have a class diagram that can control how many objects of one class can be associated with the objects of another class?
- The answer is Yes!
- Here comes the role of the multiplicity of any association.
- Let us see this example,

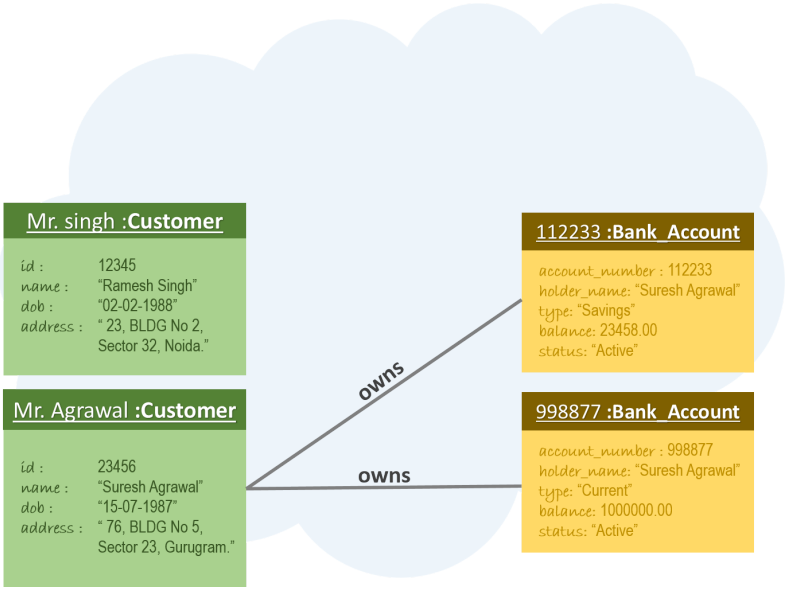



- Here we have written **1** at the end of Bank Account class.
- By seeing this class diagram we can say that any object of customer class can be associated with only one object of BankAccount.
- or customer can be associated with only one account.
- So our object diagrams will always look like this:

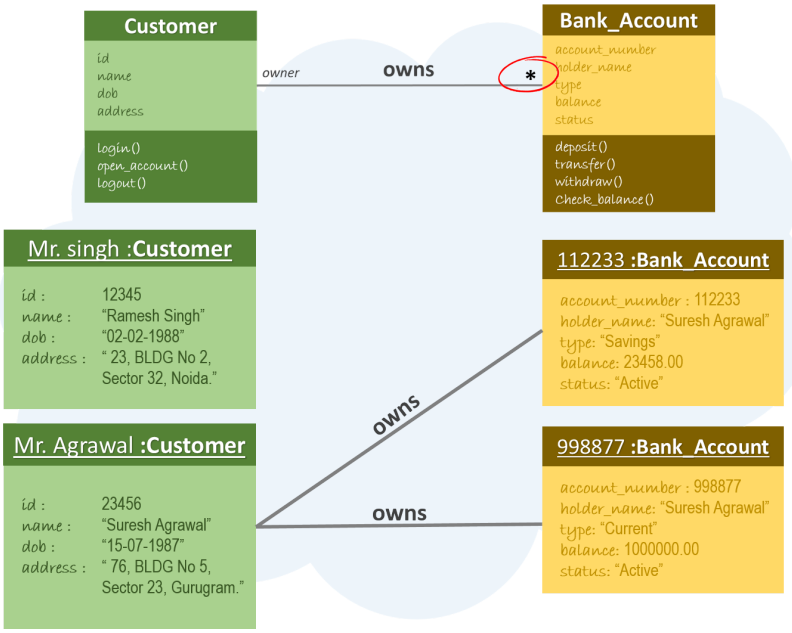



- We can say that the Multiplicity of Association tells us: how many objects of each class can take part in the relationship?

- Now let's take another example:

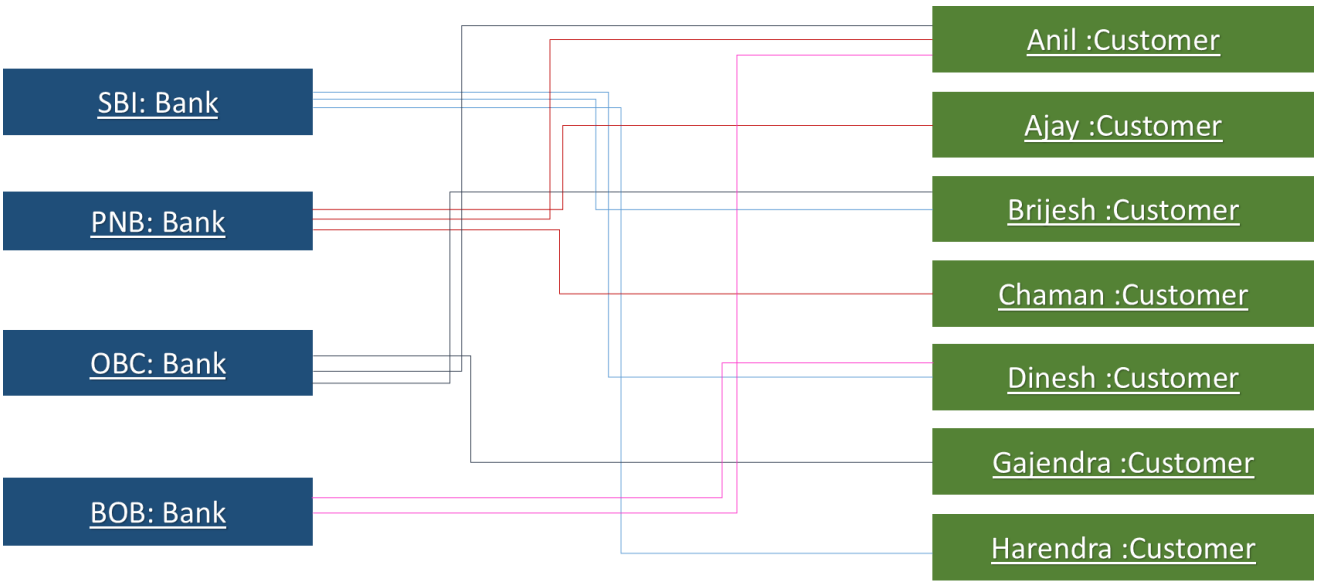



- Here we can see that one customer has no account whereas another has multiple accounts, if we want to have a class diagram in which objects of one class can be associated with 0 or more objects of another class, this can be represented with the help of *, as shown here:

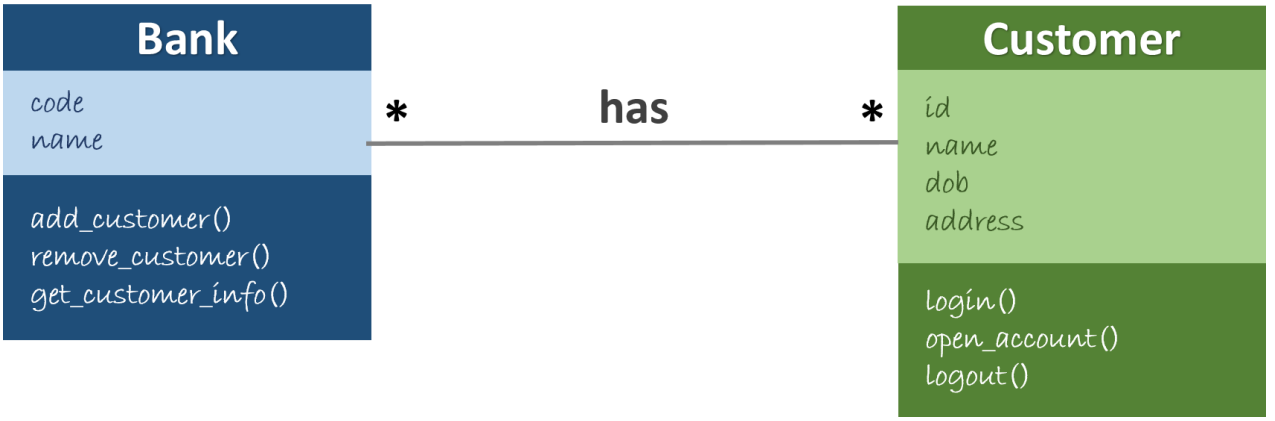



- In our next example, we have 4 banks and 7 customers. We can see that one bank has many customers like here SBI has 3 customers.

- Similarly, One customer may be associated with many banks, like here Anil is associated with PNB, OBC, and BOB banks.



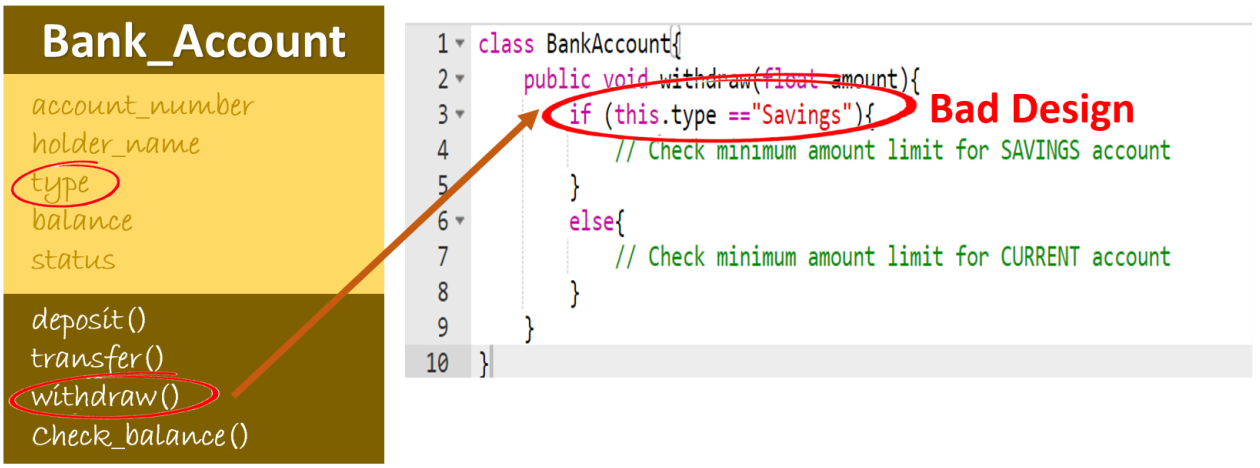

- In this object diagram, we can see that one bank can be associated with many customers, and also one customer is associated with many banks. Hence we have many to many relationships, which can be shown with the help of asterisks at both ends as shown:



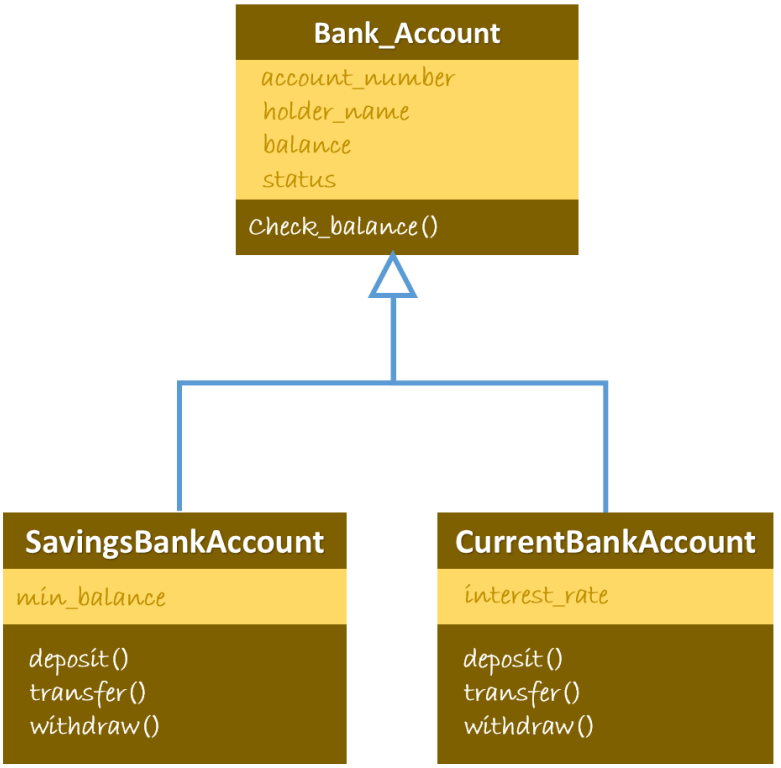

## Generalization

- Let's analyze our BankAccount Class, we can see that it has one attribute type which means that there can be different types of accounts. let us suppose that there are two types of accounts, SAVINGS and CURRENT.
- If you have some idea, you might be aware that banks impose different rules for deposit and withdrawal on different types of accounts.
- That means when we write the deposit or withdrawal function, we need to check first the type of account.

- This is an example of poor designs, wherein your code will be full of if-else statements.

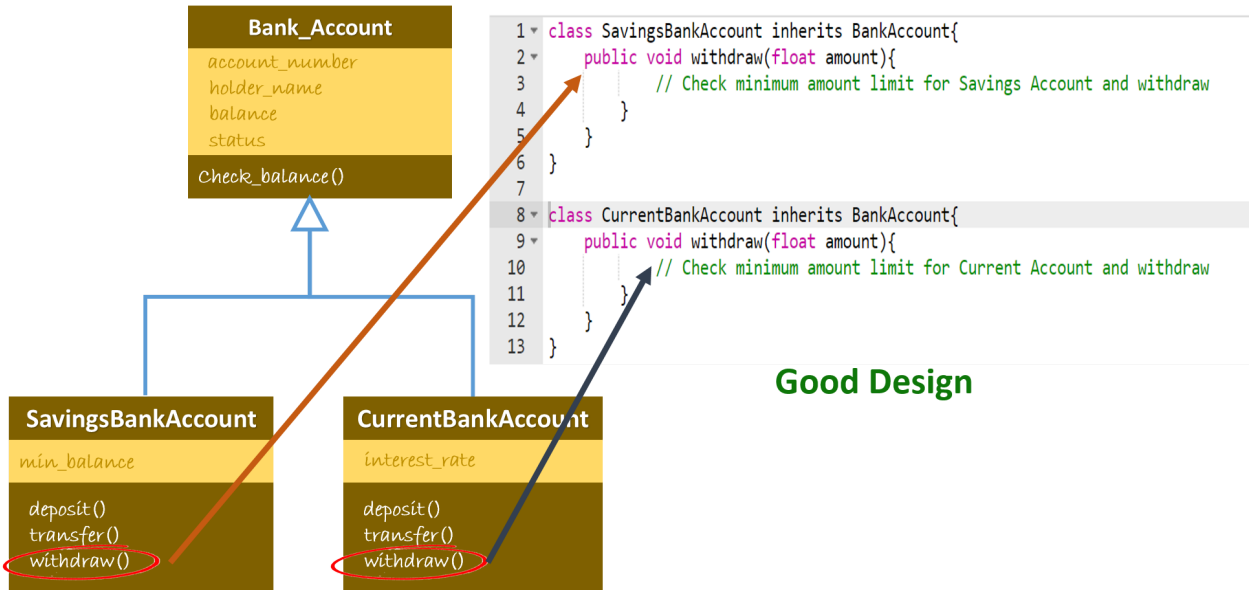



- Now let's see this scenario, where we have created two new classes one for the savings account and another for the current account. Now, these classes have their own definition of deposit, withdraw, and transfer methods.




- Hence if a customer withdraws from the current account, then the withdrawal method corresponding to CurrentBankAccount Object will only be called.

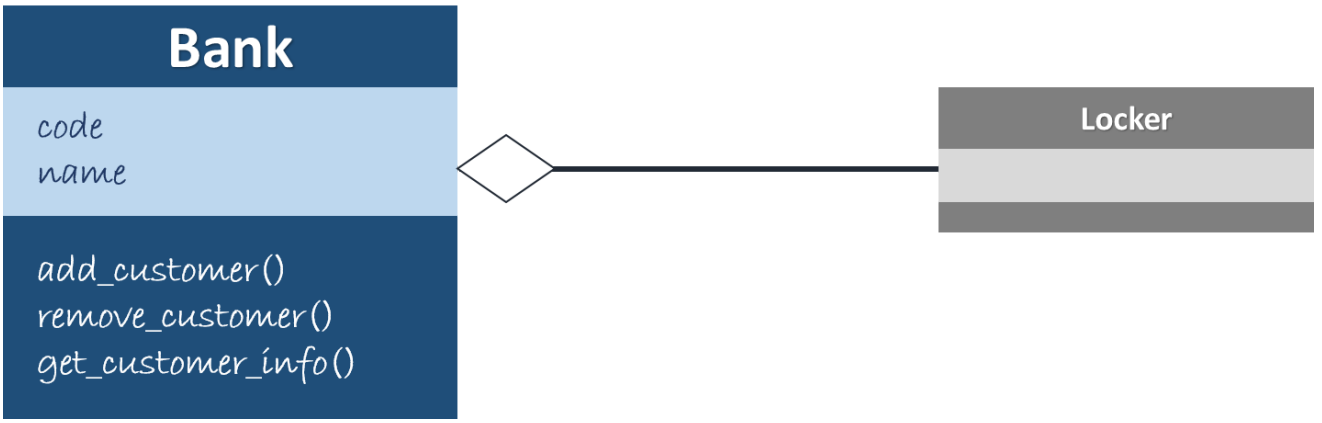
- Hence our code is cleaner and is an example of Good Design.




- In the second case we have created two new classes from one class, these new classes are called subclasses or child classes whereas the class from which they are created is called Base class or Parent class.
- In our class diagram, We can show this Parent-child class relationship via a solid line with a hollow arrowhead pointing from the child class to Parent class.
- This type of relationship is known as inheritance or generalization.

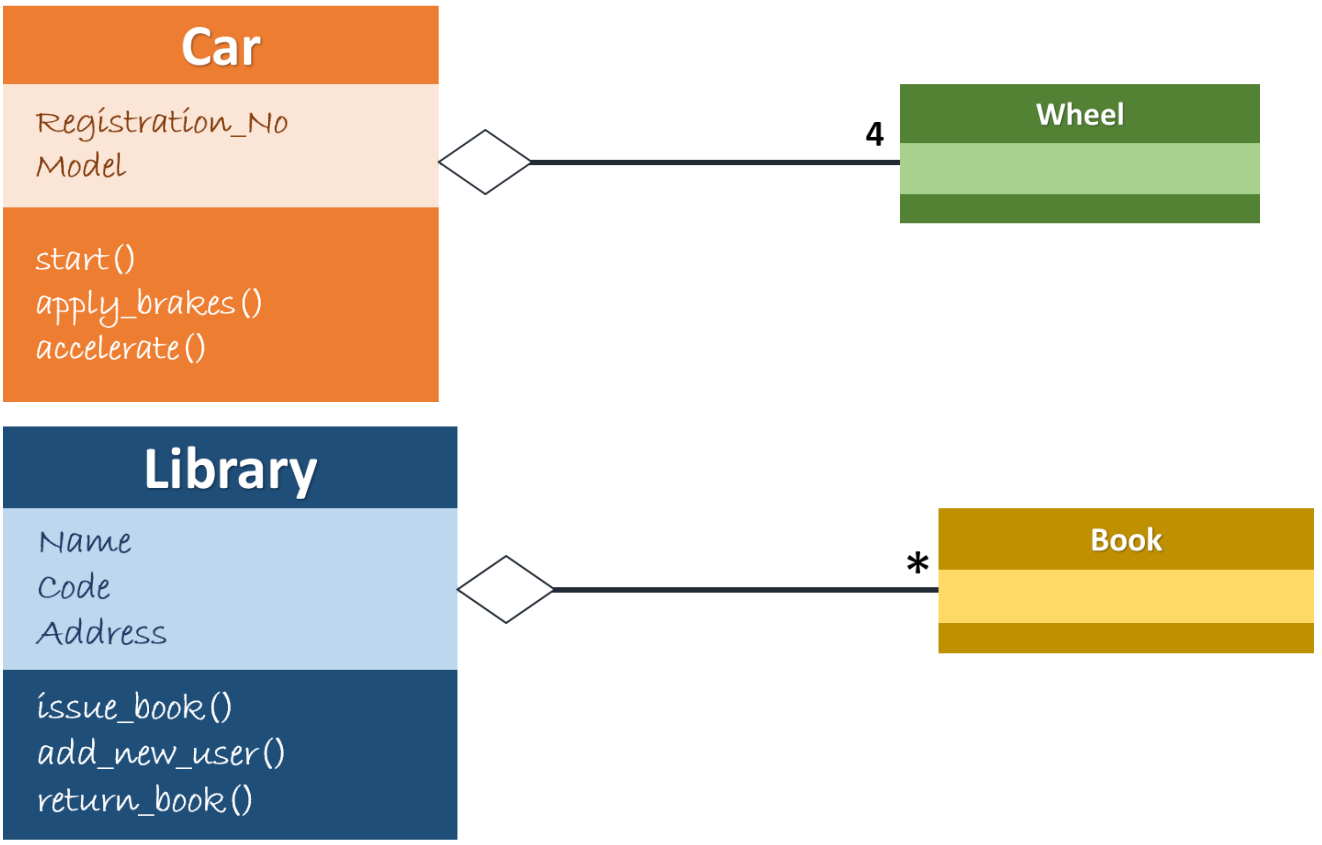
## Aggregation

- We have seen two types of relationship between the classes, that is Association and Generalization,
- Now it's time to understand the third type of relationship that is Aggregation.
- Let's understand this with the help of shown class diagram:
- Here we can see that two classes are connected with a solid line having an unfilled diamond at one end.



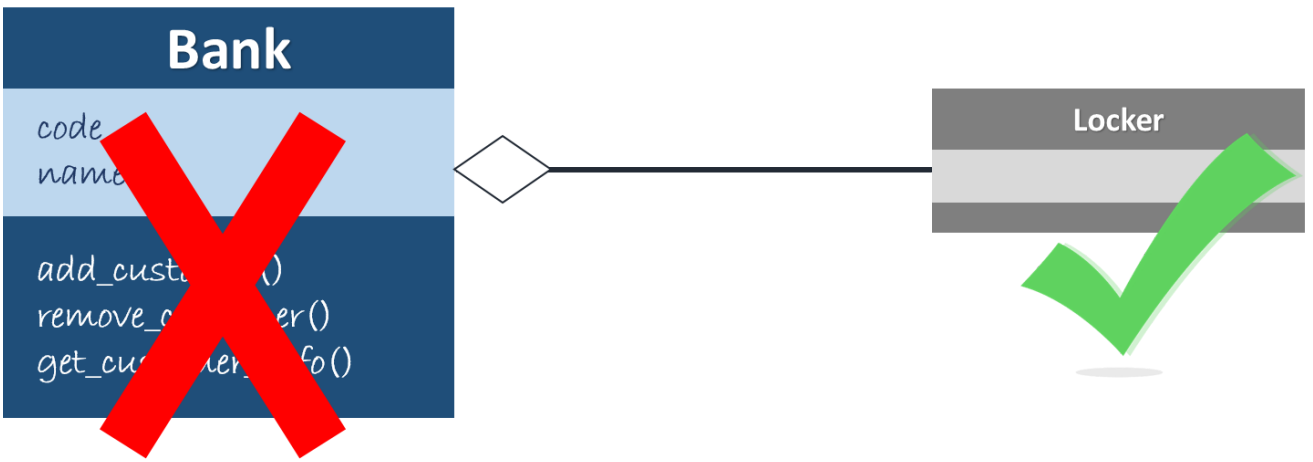

- We can read this as Bank has a part locker.
- Or inversely we can say locker is a part of Bank

- This relationship is like a whole-part relationship, where class at the diamond end is the WHOLE thing or the Aggregate, whereas the class at the other end is a PART of the WHOLE thing.
- So whenever we need to represent any class that is a part of another class, we can use aggregation.
- Another example may be:

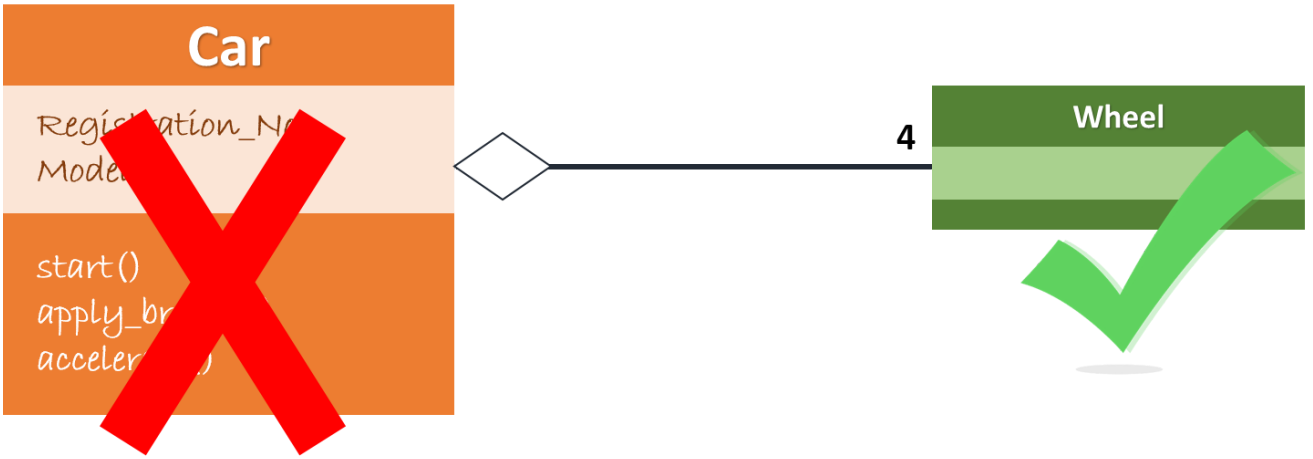



## Composition

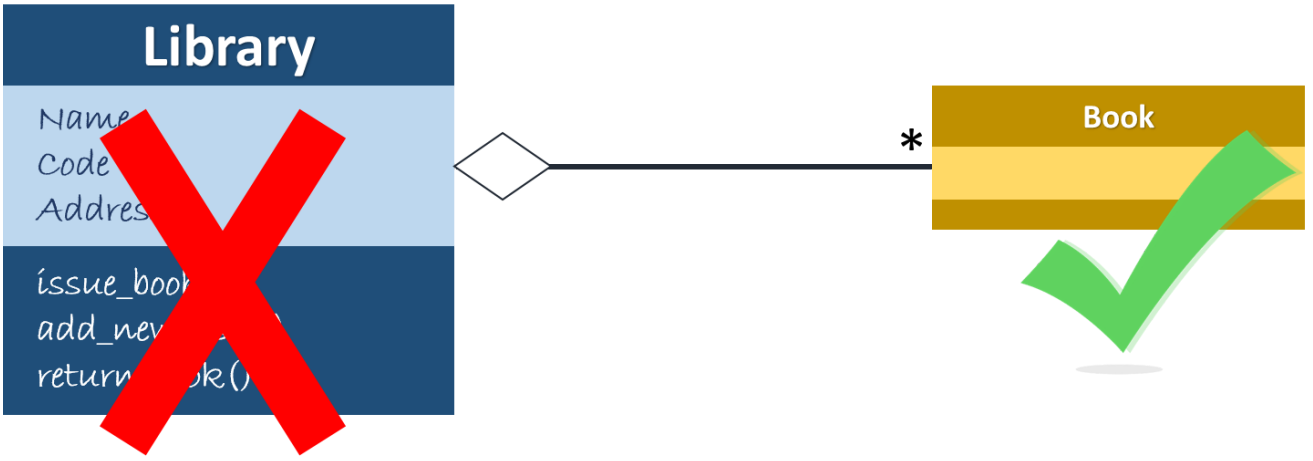
- In the case of aggregation, we saw that two classes had a part-whole relationship i.e. one class was part of another class.
- Let's see the previous example of bank and locker, here the locker is a part of the bank, but a locker has its own existence, i.e. locker can exist even if there is no bank.



- Similarly, we saw a car has wheels, here also the wheels can exist if the car is doesn't exist.



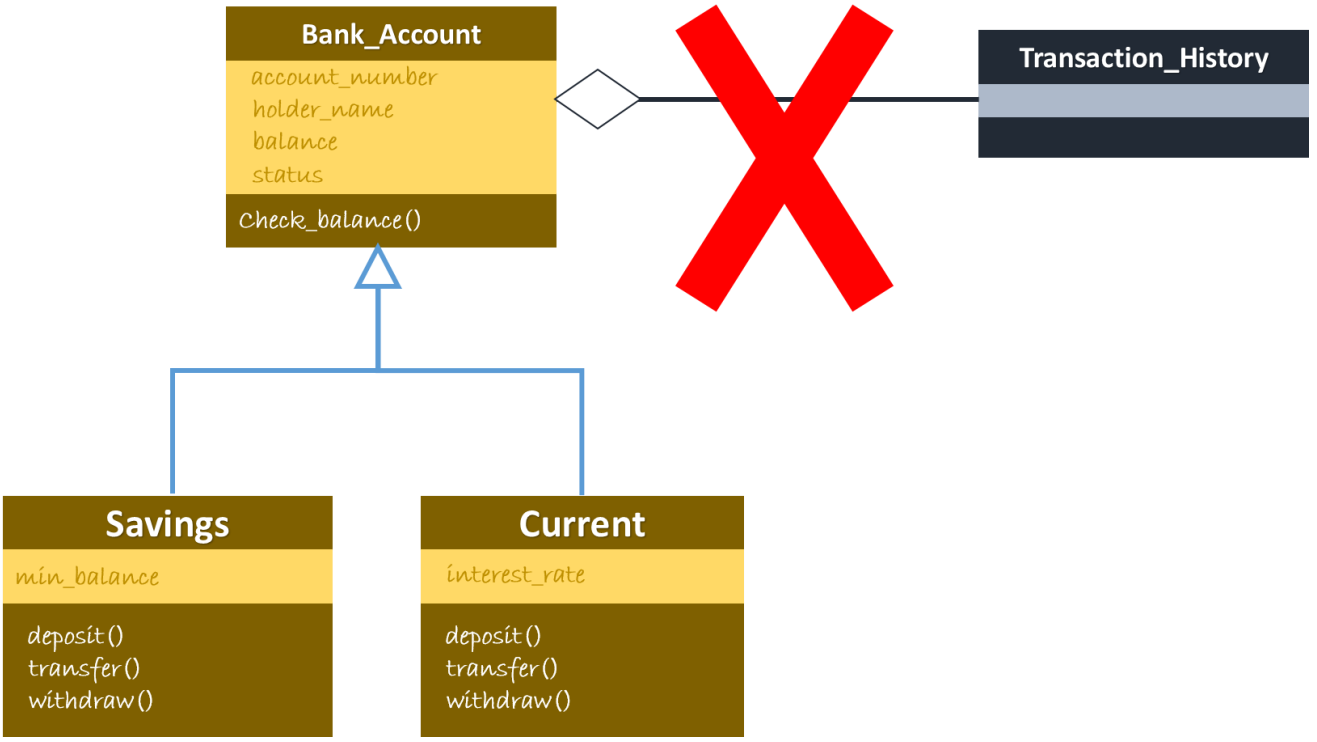

- Also, in our third example where the library had books, books can exist without the library as well.




- So, we can summarize that in the case of aggregation, the part can exist without the whole class also.

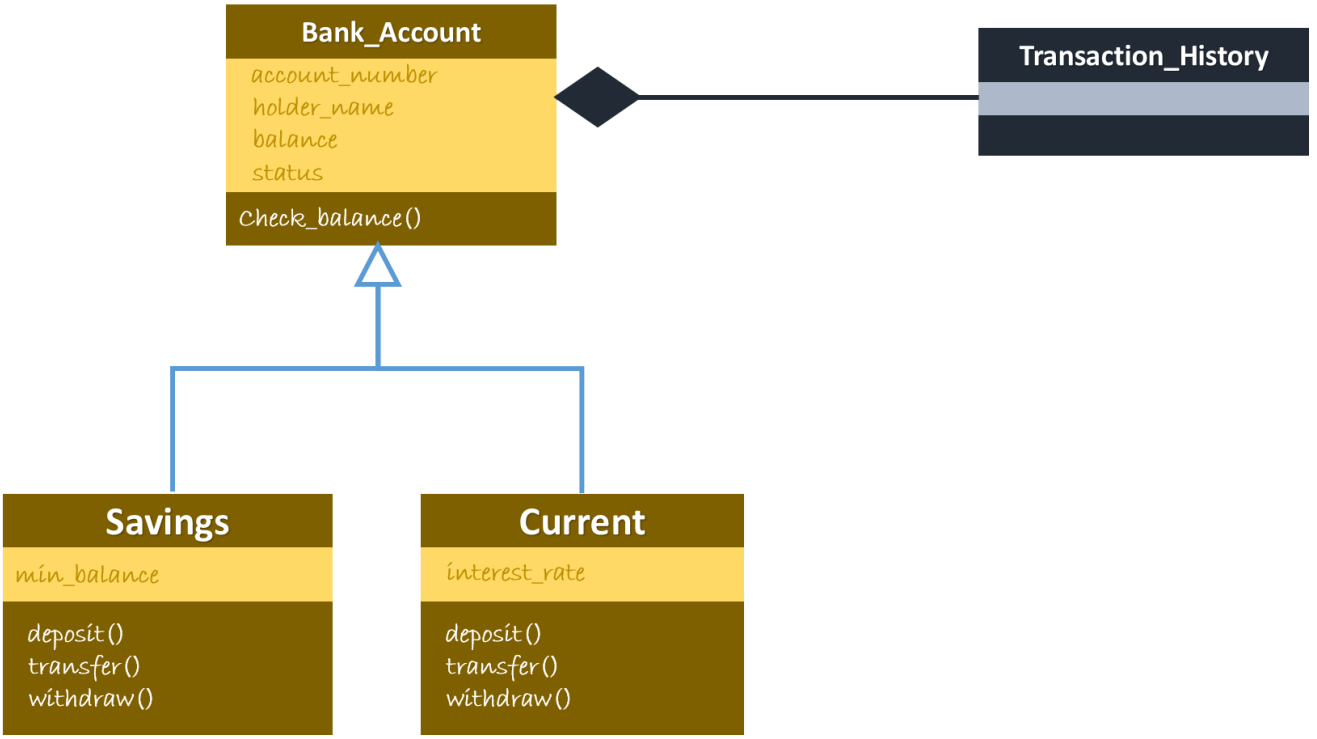
- Let's take another example.

- Just think of any bank account, the account always has a transaction history associated with it. So if we consider Bank account and transaction history as two different entities, we can say that transaction history is part of the Bank account.



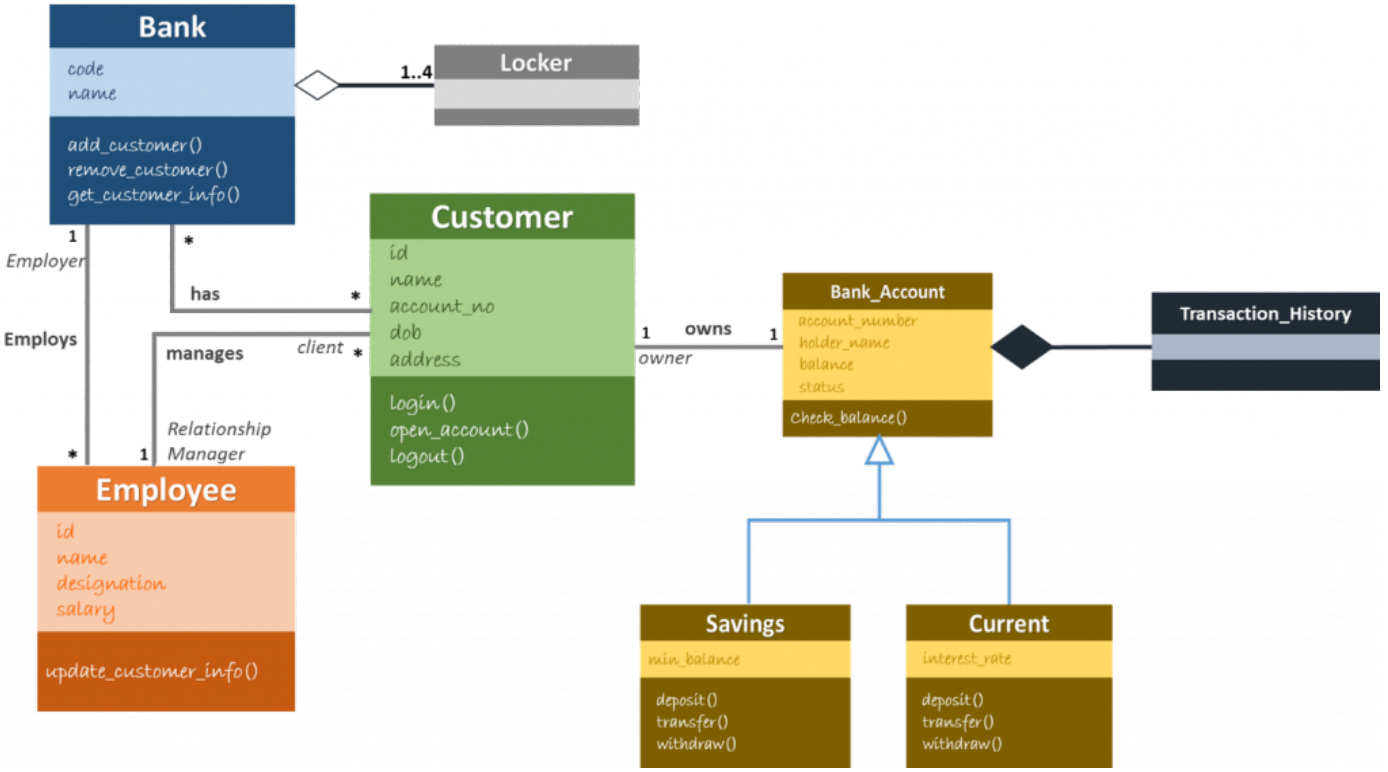

- But if the bank account doesn't exist, there will be no transaction history so this can not be represented with aggregation.

- Actually this is an example of composition. The composition is a stronger form of aggregation in which if the aggregate is destroyed, the parts are destroyed as well. We represent this with the help of a filled diamond as shown.



## Bank Class Diagram

- Its time to bring together all the concepts that we have learned so far, to create a final class diagram for a bank. Here it is:




- We can see here that the major classes that will be involved in a banking system are the bank, customer, employee, bank account, transaction history, and locker.
- Let's see bank and customer first, here bank and customer has many to many relationships that means a bank can have multiple customers, and also one customer can be associated with multiple banks.

- Now if you see a bank and employee relationships is, one to many, that is, One bank can have multiple employees, but an employee can work only for one bank.
- Let's have a look at Employee and customer, here also the relationship is one to many, that means one relationship manager can manage multiple clients, but for every client, there will be only one relationship manager.
- Now Customer and bank account relationship is one to one, that means, In this banking system, customer can have only one bank account.

- This banking system supports two types of bank accounts, CurrentBankAccount and SavingsBankAccount, which is shown by is-A relationship or inheritance.

- Also, Every bank account will have transaction history, this transaction history will exist only if the bank account exists, hence shown by Composition.

- Every bank has a locker, or we can say locker is a part of the bank, hence its shown with aggregation. Did you notice the multiplicity of this aggregation, here the bank can have a minimum of one and a maximum of 4 lockers?

---

Quiz:

```
1. attributes of FAN class
    Color
    Number of attributes

2. behaviour methods of FAN class
    Decrease Speed
    Stop

3. Class Diagram is a ----- UML Diagram
    Structural

4. Parts of a class in a class diagram
    Name component
    Operation Component
    properties Component

5. Not a type of relationship among the classes
    Accreditation

6. A solid line between the two classes is called
    Association

7. A solid between two object is called
    Link

8. The verb at the center of association between two classes is called
    Association Name

9. the noun on the association at the class end is called
    Role Name

10. Maximum number of associations between two classes is:
    infinite

11. Self - association is represented as a solid line from a class to the same class
    True

12. How many sections are there in the object diagram
    Two

13. the first compartment of object diagram contains
    Object name : class name

14. if we create an annonymous object of class person, then its name in object diagram's
first compartment will be -
    :Person
```

15. Correct Association multiplicity
    one to one
    one to many
    many to many

16. Suppose there are two classes Person and Bank Account. A person can have 2 to 5 bank accounts( minimum : 2 and maximum : 5). what multiplicity needs to be written at the bank account end of the association
    2 .. 5

17. in previous question, what will be the multiplicity, if the user has atleast one-bank account?
    1 .. *

18. Generalization (is-A) relationship is used for:
    Good code Structure
    Code Reusablity
    Implement Inheritance

19. types of inheritance
    Single
    Mulitiple
    Multilevel
    Hybrid

20. In which part-whole relationship, the part can be active even if its aggregate is destroyed
    Aggregation

21. is/are example of composition
    Banck Account and Transaction history
    Team and Players

22. How aggregation is different from the association?
    - Aggregation is a part of relationship, the association has a relationship
    - Association can be bi-directional whereas aggregation will always be from whole part of class