USE CASE DIAGRAMS

Use case diagram

Use case diagram is a behavioral **UML diagram type** and frequently used to analyze various systems. They enable us to visualize the different types of roles in a system and how those roles interact with the system.

Importance of Use Case Diagrams

As mentioned before use case diagram are used to gather a usage requirement of a system. Depending on your requirement you can use that data in different ways. Below are few ways to use them.

- **To identify functions and how roles interact with them –** The primary purpose of use case diagrams.
- **For a high level view of the system –** Especially useful when presenting to managers or stakeholders. You can highlight the roles that interact with the system and the functionality provided by the system without going deep into inner workings of the system.
- **To identify internal and external factors –** This might sound simple but in large complex projects a system can be identified as an external role in another use case.

Use Case Diagram objects

Use case diagrams consist of 4 objects.

1.Actor

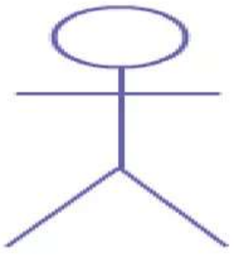
2.Use case

3.System

4.Package

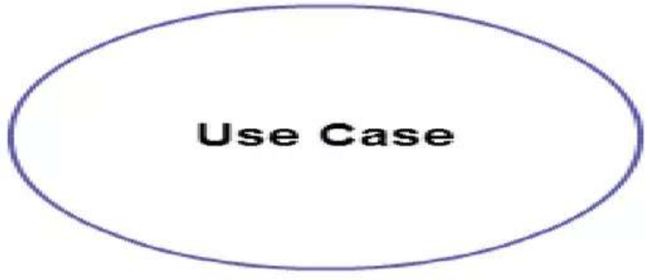
- **Actor**

Actor in a use case diagram is **any entity that performs a role** in one given system. This could be a person, organization or an external system.



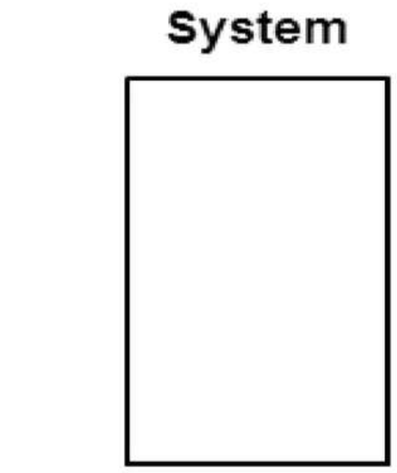
- **Use Case**

A use case **represents a function or an action within the system**. Its drawn as an oval and named with the function.



- **System**

System is used to **define the scope of the use case** and drawn as a rectangle. Create all the use cases and then use the system object to define the scope covered by your project. Or we can even use it to show the different areas covered in different releases.



- **Package**

Package is another optional element that is extremely useful in complex diagrams. Similar to **class diagrams**, packages are **used to group together use cases**. They are drawn like the image shown below.

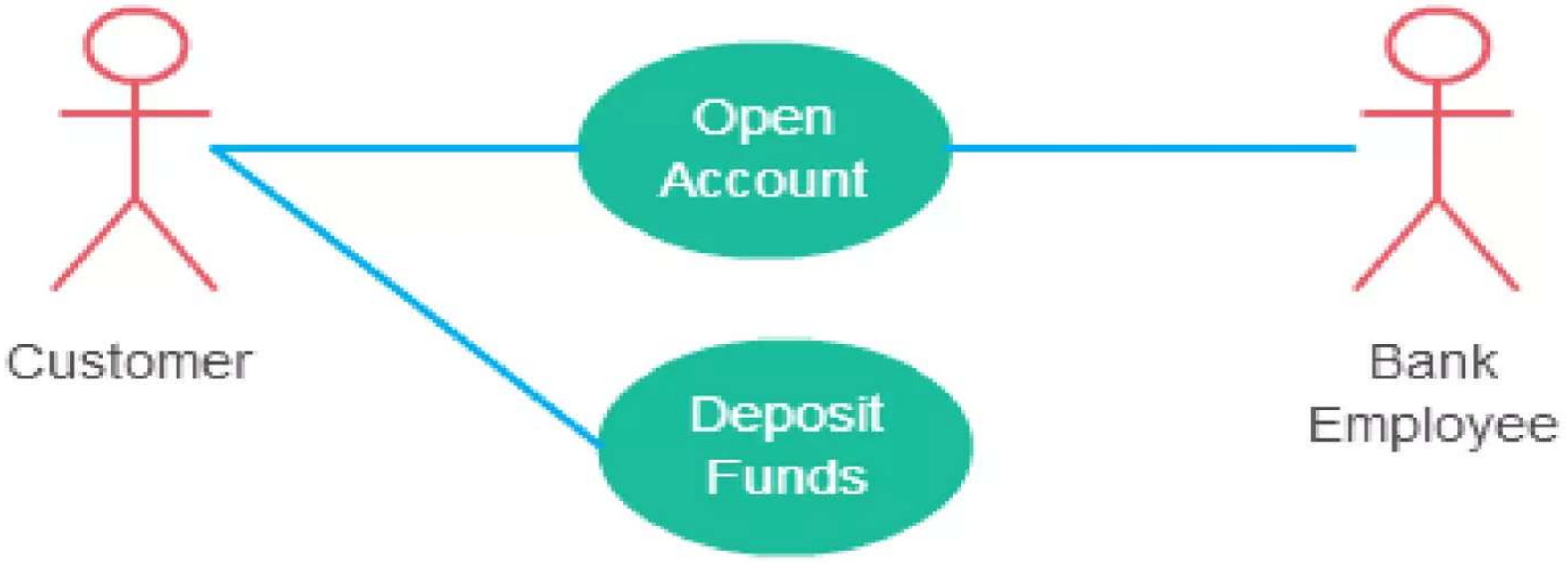
Relationships in Use Case Diagrams

- There are five types of relationships in a use case diagram. They are:

1. Association between an actor and a use case
2. Generalization of an actor
3. Extend relationship between two use cases
4. Include relationship between two use cases
5. Generalization of a use case

Association Between Actor and Use Case

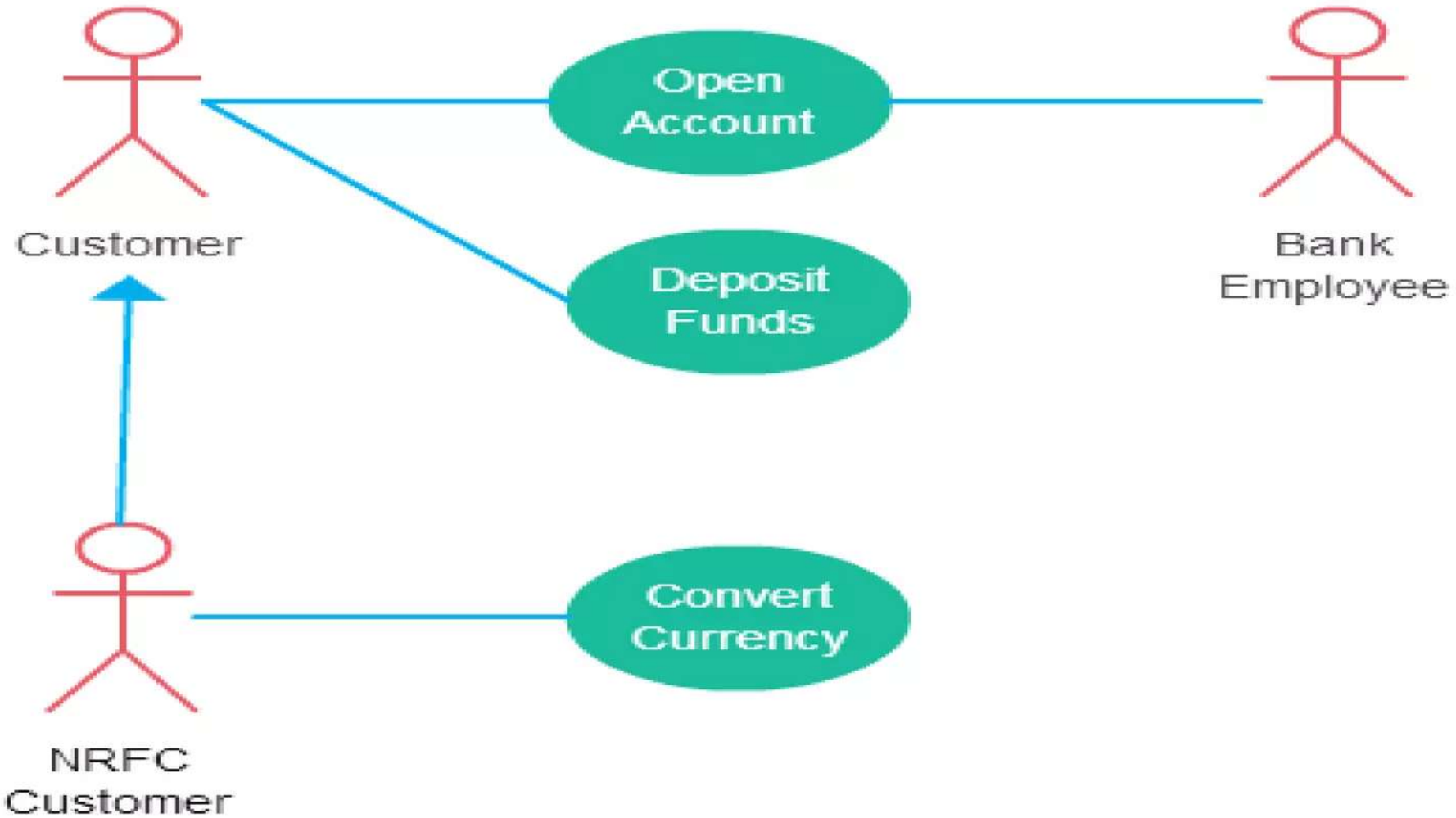
- This one is straightforward and present in every use case diagram.
- An actor must be associated with at least one use case.
- An actor can be associated with multiple use cases.
- Multiple actors can be associated with a single use case.



Different ways association relationship appears in use case diagrams

Generalization of an Actor

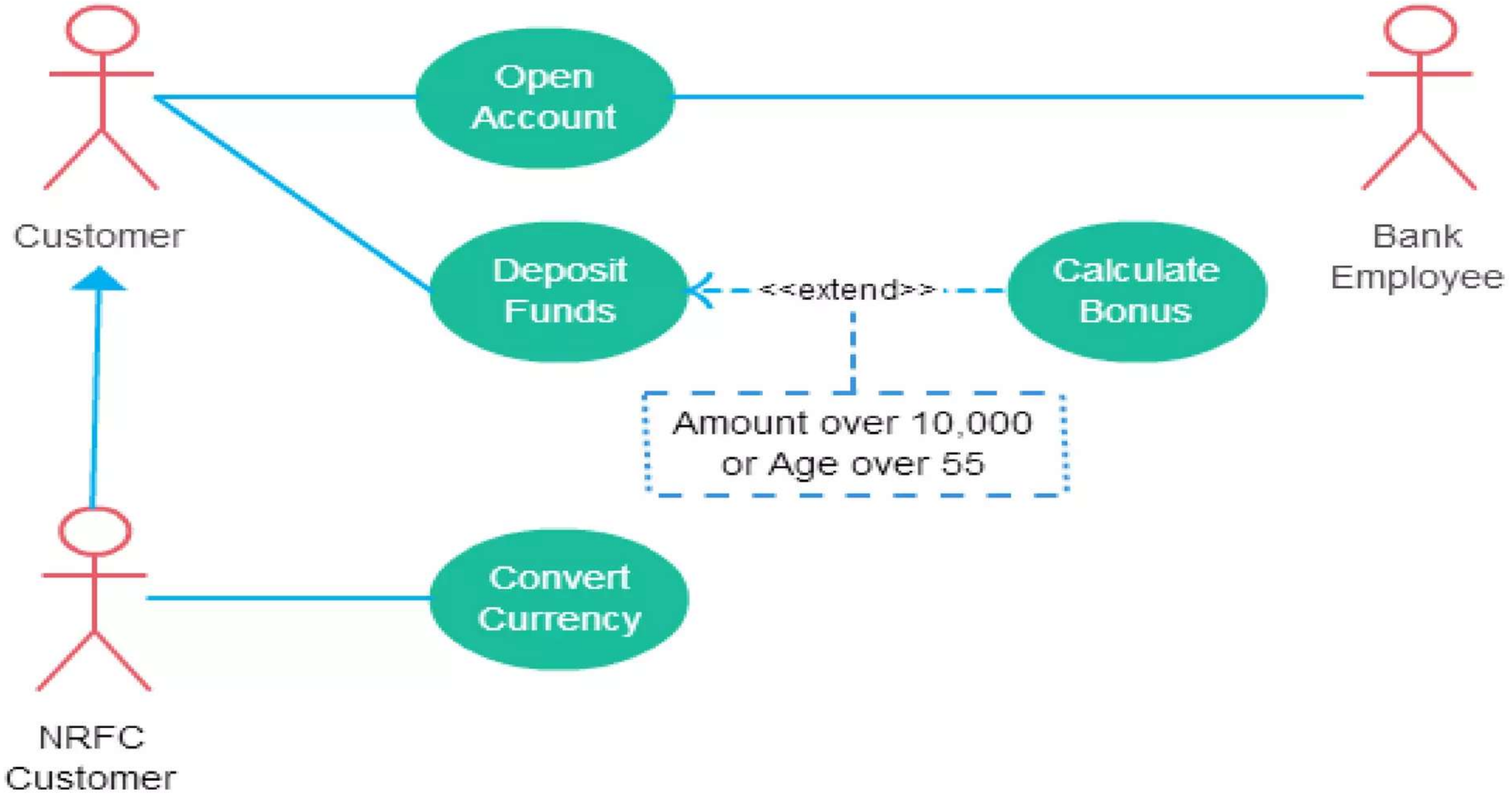
- Generalization of an actor means that one actor can inherit the role of an other actor. The descendant inherits all the use cases of the ancestor. The descendant have one or more use cases that are specific to that role.



A generalized actor in an use case diagram

Extend Relationship Between Two Use Cases

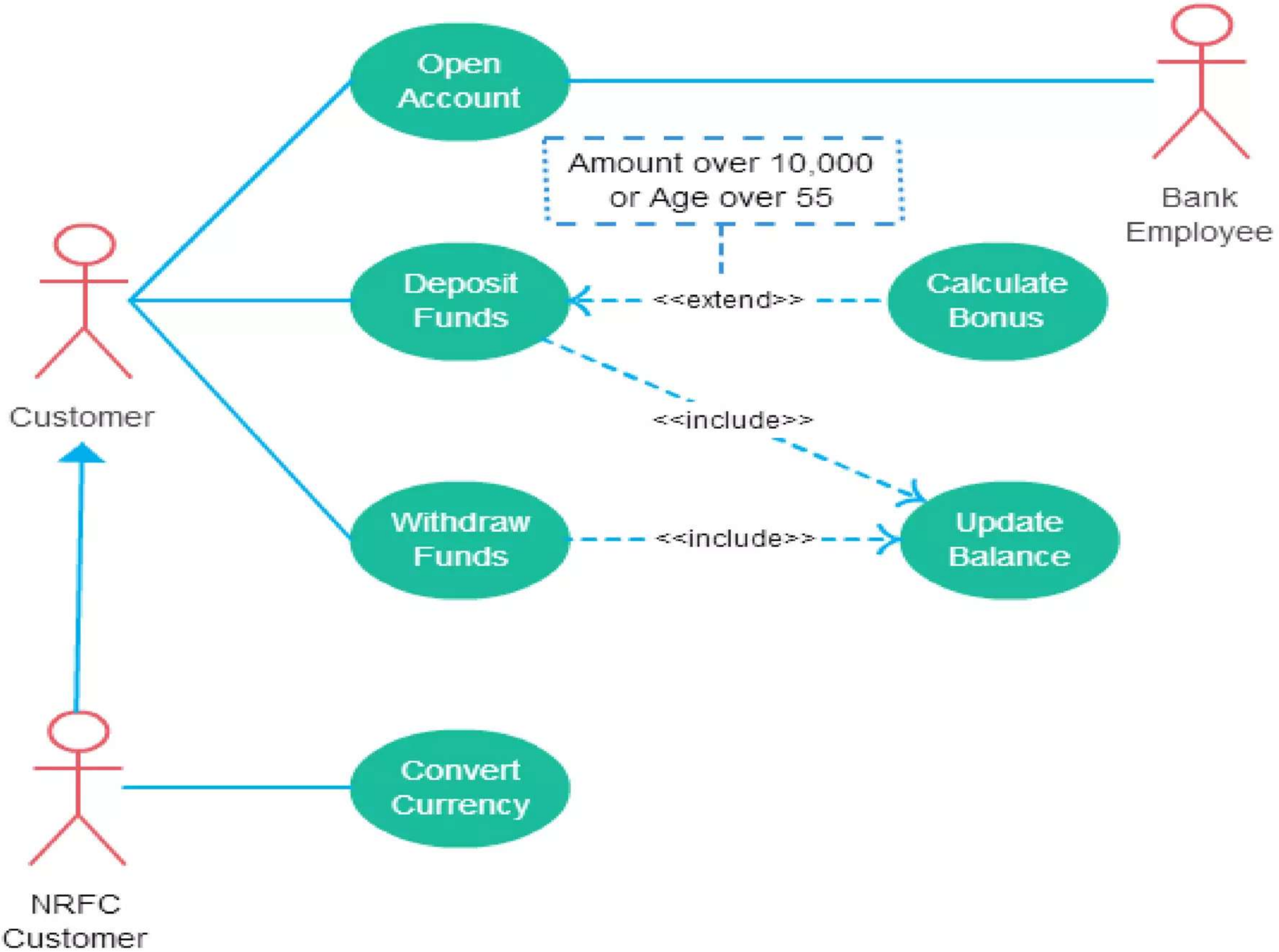
- As the name implies it extends the base use case and adds more functionality to the system. Here are few things to consider when using the **<<extend>>** relationship.
- **The extending use case is dependent on the extended (base) use case**. In the below diagram the "Calculate Bonus" use case doesn't make much sense without the "Deposit Funds" use case.
- **The extending use case is usually optional** and can be triggered conditionally. In the diagram you can see that the extending use case is triggered only for deposits over 10,000 or when the age is over 55.
- **The extended (base) use case must be meaningful on its own**. This means it should be independent and must not rely on the behavior of the extending use case.



Extend relationship in use case diagrams

Include Relationship Between Two Use Cases

- Include relationship show that the behavior of the **included** use case is part of the **including (**base) use case. The main reason for this is **to reuse the common actions across multiple use cases**. Few things to consider when using the <<include>> relationship.
- The base use case is incomplete without the included use case.
- The included use case is mandatory and not optional.
- Lest expand our banking system use case diagram to show include relationships as well.



Includes is usually used to model common behavior

Generalization of a Use Case

- This is similar to the generalization of an actor. The behavior of the ancestor is inherited by the descendant. This is used when there are common behavior between two use cases and also specialized behavior specific to each use case.
- For example in the previous banking example there might be an use case called “Pay Bills”. This can be generalized to “Pay by Credit Card”, “Pay by Bank Balance” etc.

How to Create a Use Case Diagram

Identifying Actors

- Actors are external entities that interact with your system. It can be a person, another system or an organization. In a banking system the most obvious actor is the customer. Other actors can be bank employee or cashier depending on the role we are trying to show in the use case.
- An example of an external organization can be the tax authority or the central bank. Loan processor is a good example of external system associated as an actor.

Identifying Use Cases

- Now it's time to identify the use cases. A good way to do this is to identify what the actors needs from the system. In a banking system a customer will need to open accounts, deposit and withdraw funds, request check books and similar functions. So all of these can be considered as use cases.
- Top level use cases should always provide a complete functions required by an actor. You can extend or include use cases depending on the complexity of the system.

How to Create a Use Case Diagram (cont..)

- **Look for Common Functionality to use Include**

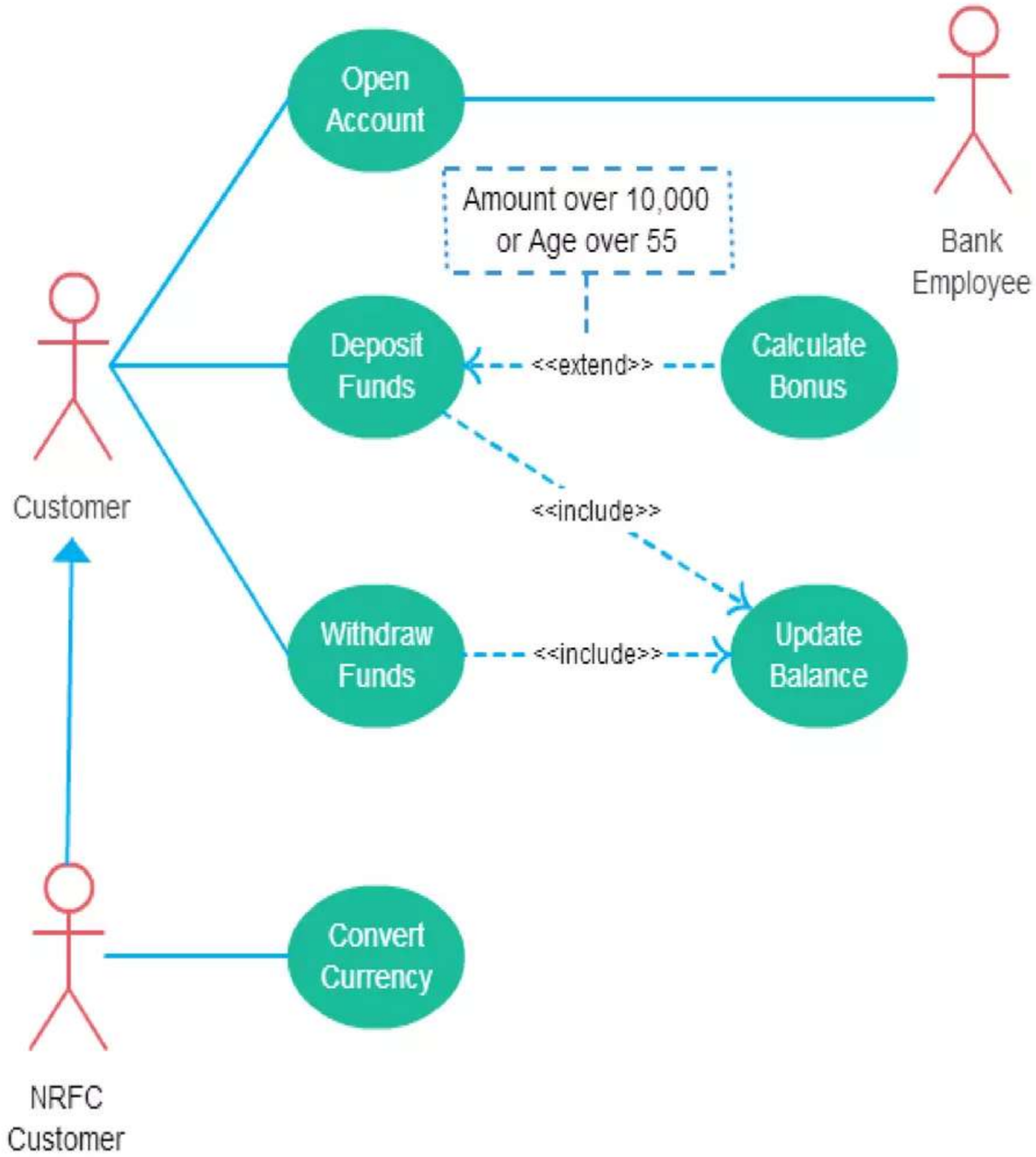
Look for common functionality that can be reused across the system. If you find two or more use cases that share common functionality you can extract the common functions and add it to a separate use case. Then you can connect it via the include relationship to show that its always called when the original use case is executed.

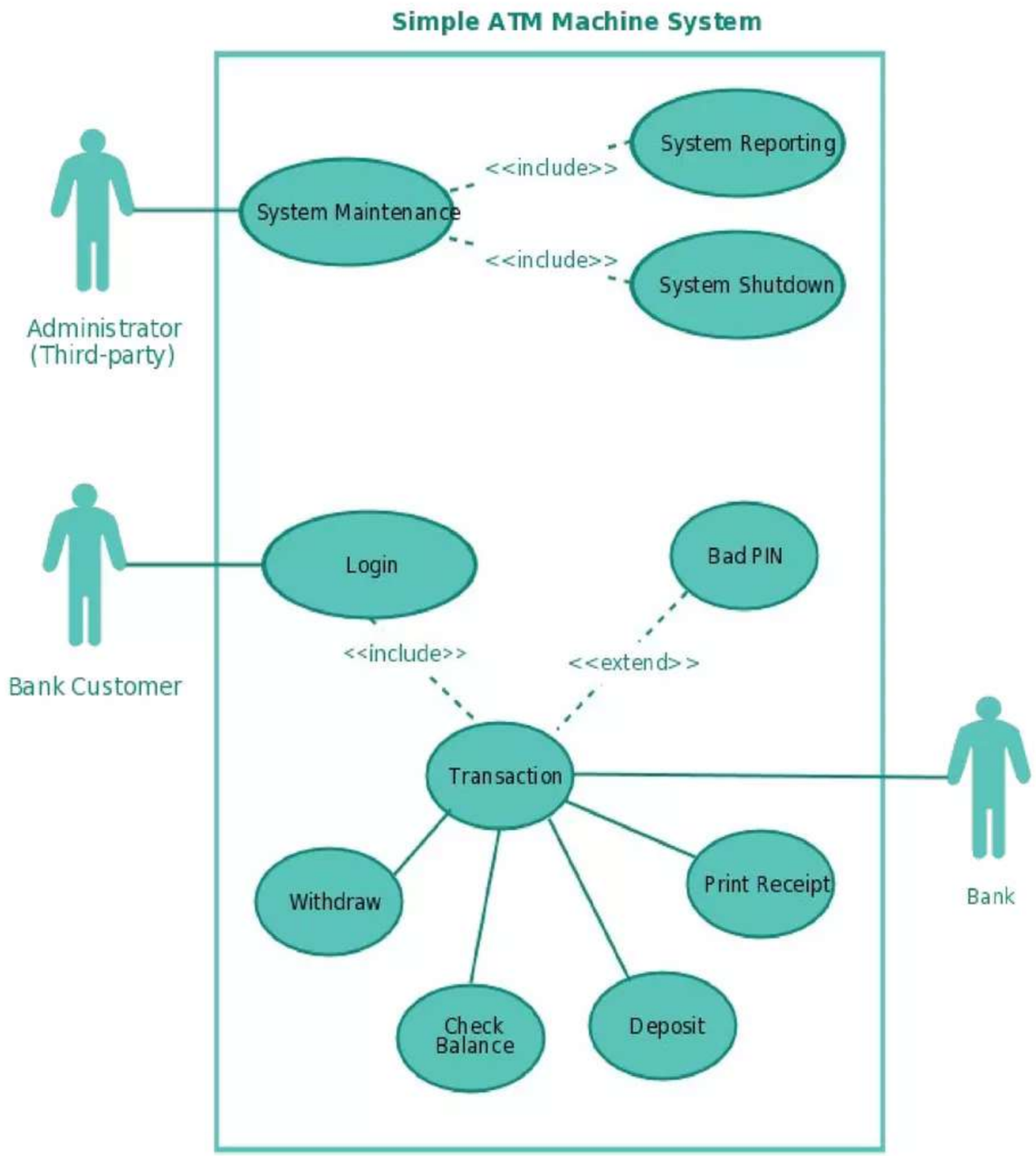
- **Is it Possible to Generalize Actors and Use Cases**

There maybe instances where actors are associated with similar use cases while triggering few use cases unique only to them. In such instances you can generalize the actor to show the inheritance of functions. You can do a similar thing for use case as well.

- **Optional Functions or Additional Functions**

There are some functions that are triggered optionally. In such cases you can use the extend relationship and attach and extension rule to it. In the below banking system example "Calculate Bonus" is optional and only triggers when a certain condition is matched.


Open Account
Amount over 10,000 or Age over 55
Bank Employee
Deposit Funds
<<extend>>
Calculate Bonus
Customer
<<include>>
Withdraw Funds
<<include>>
Update Balance
Convert Currency
NRFC Customer


Simple ATM Machine System
Administrator (Third-party)
System Maintenance
<<include>>
System Reporting
<<include>>
System Shutdown
Bank Customer
Login
Bad PIN
<<include>>
<<extend>>
Transaction
Withdraw
Print Receipt
Check Balance
Deposit
Bank