**Use Case Diagram**

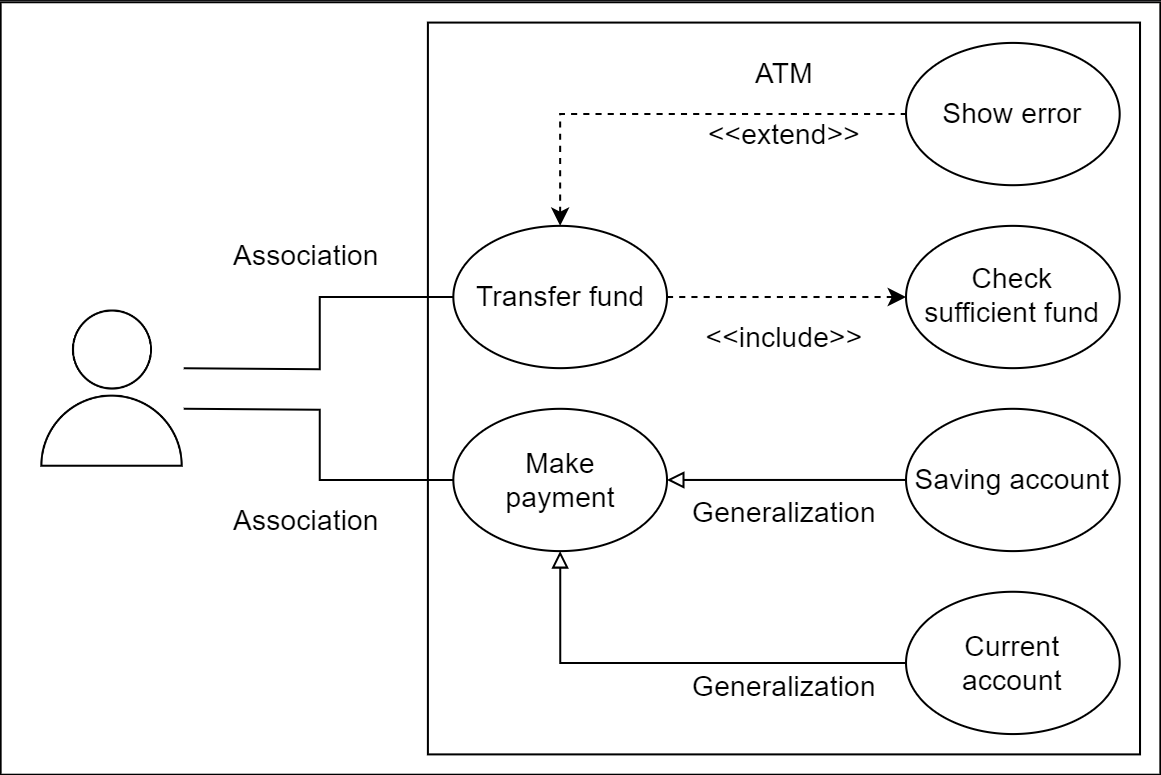
1. **Relationships in use case diagrams**

There are four different types of relationships in a use case diagram:

1. **Association:** This shows the relationship between and among actor(s) and use case(s). It represents how an actor can perform certain functions. It is denoted by a solid line without arrows. All the actors in a use case diagram must have at least one association with any use case. More than one actor can be associated with the same use case, and a single actor can be associated with more than one use case.
2. **Generalization:**This relationship is also known as inheritance. In inheritance, we have parent and children classes. Similarly, in a use case diagram, we have parent and child use cases. The child use case has generalization with the parent use case. Each child inherits the behavior of its parent. It is denoted by a solid line with an arrow on only one side (toward the parent use case).
3. **Include:**We use this to show the relationship between two use cases. It shows that one use case includes the behavior of another use case. The included use case will execute only after the execution of the base use case. We can also say that the base use case requires an included use case in order to be completed. It is represented by a dashed line with an arrow on only one side (toward the included use case), and we write <<include>> above the line.
4. **Extend:**We use this to show the relationship between two use cases. It shows that one use case extends the behaviors of another use case. The extended use case does not execute every time. It always depends on certain conditions. It is used to extend the functionality of the base use case. It is represented by a dashed line with an arrow on only one side (toward the base use case), and we write <<extend>> above the line.

In the example below, we have a small ATM (automated teller machine) transaction system where customers can transfer funds and make payments. To validate the funds, the transfer system has to check if a sufficient amount of funds is available. Otherwise, an error message will be displayed. To make a payment, a customer has two choices. It can either pay via a current account or a savings account.

The use case diagram of an ATM



**Class Diagram**

1. **Popular notations in the class diagram**

The following are some essential notations of the class diagram:

* Class notation
* Interface, abstract class, and enumeration
* Access modifiers

1. **Class notation**

A class is represented by a rectangle with three sections. The first section holds the class **name**, the second one lists the **attributes**, and the third one shows the **methods** (operations). The following is the depiction of a `**Movie`** class with its attributes and methods.

Notation of a class in a class diagram

1. **Interface, abstract class, and enumeration**

We can declare a class as abstract using **abstract** keywords. The class name will be printed in italic. We can use the interface, annotation, and enum keywords too. The illustration below shows how to depict these notations in a class diagram.

Class diagram notation for abstract classes, interfaces, enumerations, and annotations

1. **Access modifiers**

You may use character symbols to specify the visibility of the associated object when defining methods or attributes. The most widely used access modifiers are as follows:

**Public:**A public member can be seen anywhere in the system. It is represented by a + symbol.

**Private:** Members can only be accessible from within the class. It is inaccessible from outside the class. It is represented by a - symbol.

**Protected**: Members are only accessible within the class and derived classes. It is represented by the # symbol.

The following images show how to use the access modifiers in the class diagram: