Class Diagram for the ATM System

Learn to create a class diagram for the ATM design using the bottom-up approach.

We'll cover the following

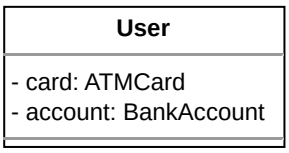
In this lesson, we'll design the classes and then identify the relationship between classes according to the requirements for the ATM design problem.

Components of the ATM system

As mentioned earlier, we'll design the class diagram for the ATM using a bottom-up approach.

User

The `User` class represents a user with an ATM card and a bank account.



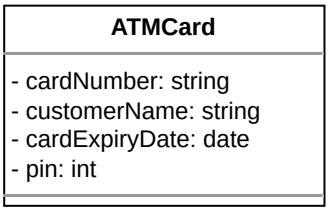
The User class

R1: ATM Design

Each user has a single account at the bank that they can access by inserting their card into the ATM.

ATM card

The `ATMCard` class is identified by the card number, customer name, expiration date, and the user's PIN.



The ATMCard class

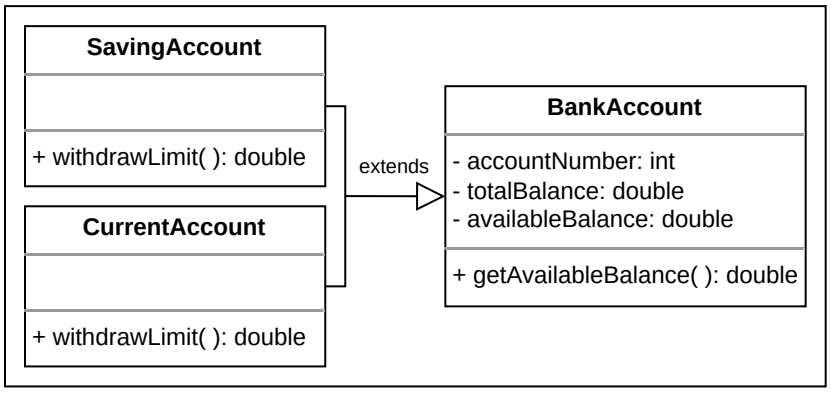
R4: ATM Design

R4: All transactions are possible after the successful authentication of the ATM card.

Bank account

`BankAccount` is a parent class with two types: `SavingAccount` and `CurrentAccount`. These classes are derived from the `BankAccount` class. This class stores the account number, total balance, and the user's available balance.

- `SavingAccount`: This derived class represents a saving account with a withdrawal limit.
- `CurrentAccount`: This derived class represents a current/checking account with a withdrawal limit.



BankAccount and its derived classes

R1 and R5: ATM Design

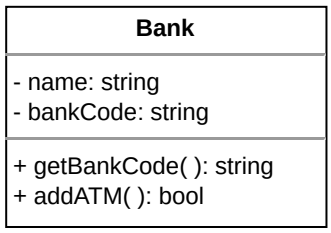
R1: Each user has a single account at the bank that they can access by inserting their card into the ATM.

R5: The user can have two types of accounts—current and savings—and can perform the following operations on the ATM:

- Balance inquiry
- Cash withdrawal
- Funds/money transfer

Bank

The `Bank` class represents a bank with a name and a bank code. A bank may or may not have an ATM.



The Bank class

R1: ATM Design

R1: Each user has a single account at the bank that they can access by inserting their card into the ATM.

Card reader, cash dispenser, keypad, screen, and printer

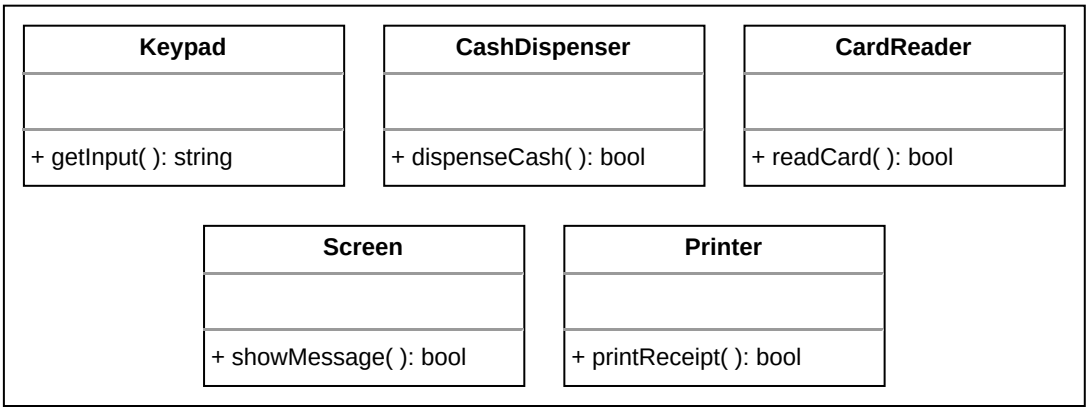
`CardReader`: This class accepts or rejects a card.

`CashDispenser`: This class provides the required amount specified by the user in cash.

`Keypad`: This class allows the user to enter the PIN.

`Screen`: This class represents a screen that displays information upon insertion of the card.

`Printer`: This class represents a printer that prints the transaction/withdrawal receipts for the user.



The class diagram of the Keypad, CashDispenser, CardReader, Screen and Printer classes

R2: ATM Design

R2: The main components of the ATM system that facilitate interactions between the user and the machine are listed below:

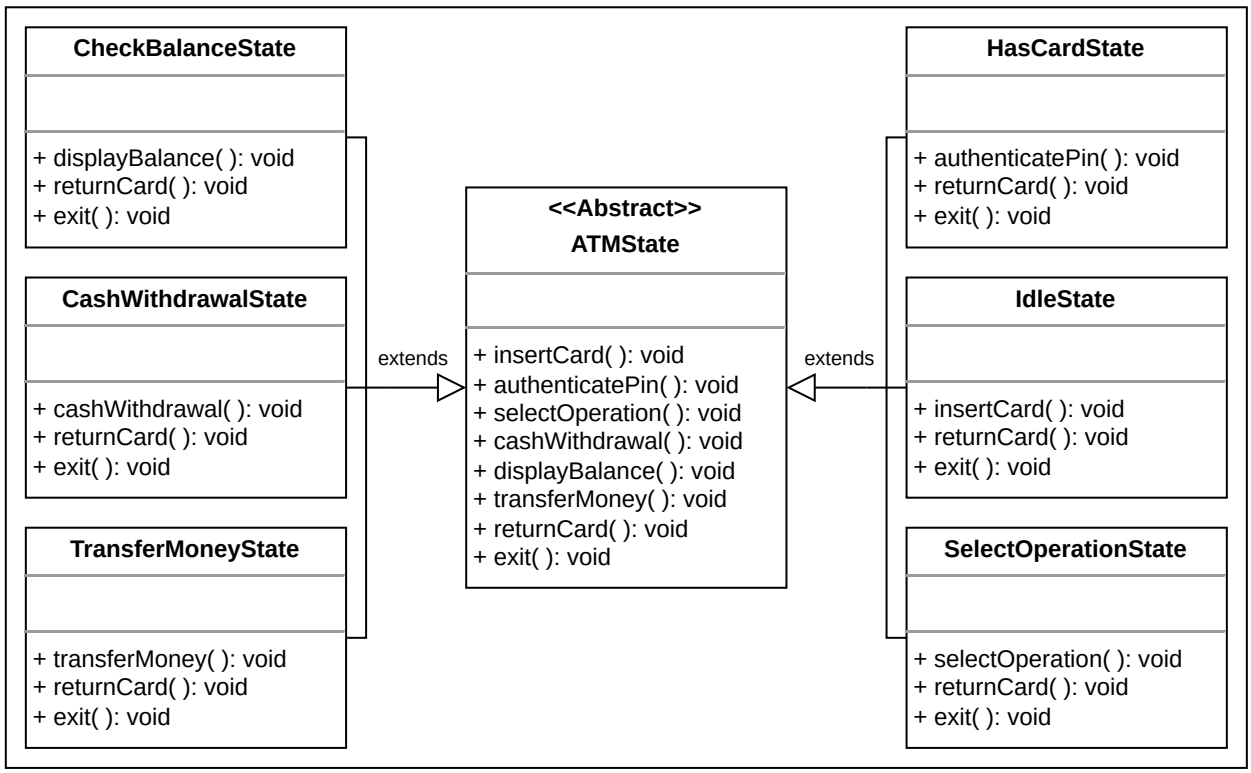
- **Card reader:** To read the user's ATM card
- **Keypad:** To enter information such as the user's PIN
- **Screen:** To display messages to the user, such as prompts or error messages
- **Cash dispenser:** To dispense cash to the user
- **Printer:** To print receipts for the user
- **Network infrastructure:** To connect with the bank's computer system in order to access account information and complete transactions

ATM state

`ATMState` is an abstract class with six types: `CheckBalanaceState`, `CashWithdrawalState`, `TransferMoneyState`, `HasCardState`, `IdleState`, and `SelectOperationState`. These classes are derived from the `ATMState` class. This class decides the state of the ATM system and several states including the return card and exit of the ATM system.

- `CheckBalanceState`: This class represents the state that allows users to check their account balance.

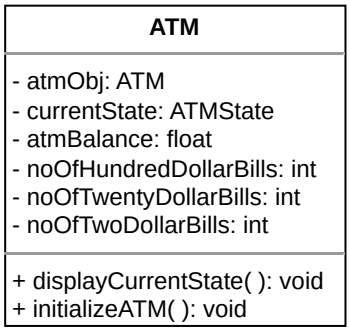
- `CashWithdrawalState`: This class represents the state that allows users to withdraw cash.
- `TransferMoneyState`: This class represents the state that allows users to transfer money.
- `HasCardState`: This class represents the state that checks whether or not the user has a valid card and authenticates the card's PIN.
- `IdleState`: This class represents the state where the ATM system is idle and is not performing any functions.
- `SelectOperationState`: This class represents the state that allows users to select an operation for the ATM to perform.



ATMState and its derived classes

ATM

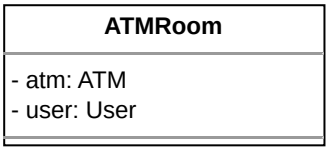
An `ATM` class can either have an idle state or can be performing an operation. It has a limited number of hundred, twenty, and two dollar bills.



The ATM class

ATM room

An `ATMRoom` class has an ATM and may or may not have a user.

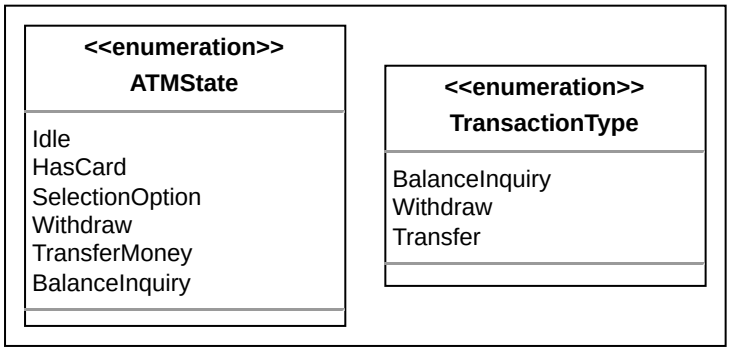


The ATMRoom class

Enumerations and custom data types

The following provides an overview of the enumerations and custom data types used in this problem.

- `ATMState`: This enumeration keeps track of the following states of an ATM:
  - Idle
  - Card inserted by the user
  - Option selected
  - Cash withdrawal
  - Transfer money
  - Display the account balance
- `TransactionType`: This enumeration represents the following transactions:
  - Balance inquiry
  - Cash withdrawal
  - Funds/money transfer



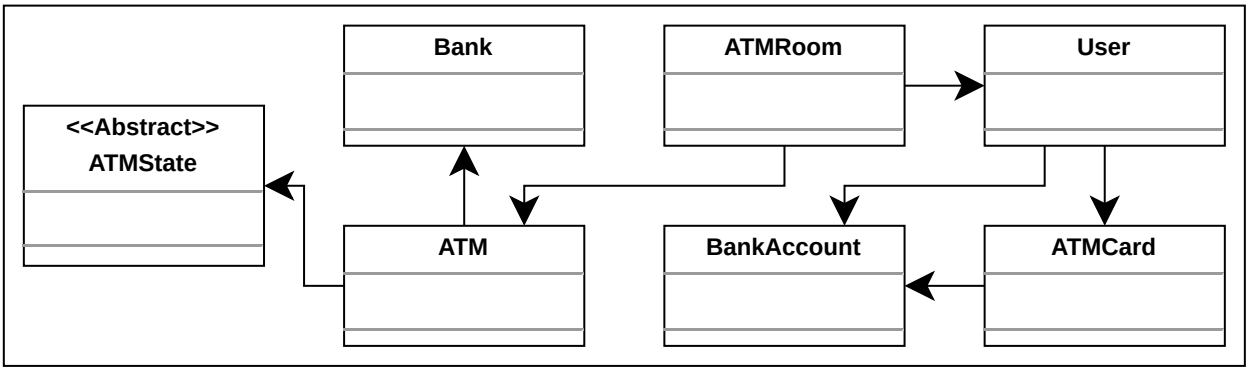
Enums in the ATM design problem

Relationship between the classes

Association

The class diagram has the following association relationships:

- The `ATMRoom` class has a one-way association with `User` and `ATM`.
- The `User` class has a one-way association with `ATMCard` and `BankAccount`.
- The `ATMCard` class has a one-way association with `BankAccount`.
- The `ATM` class has a one-way association with `Bank` and `ATMState`.

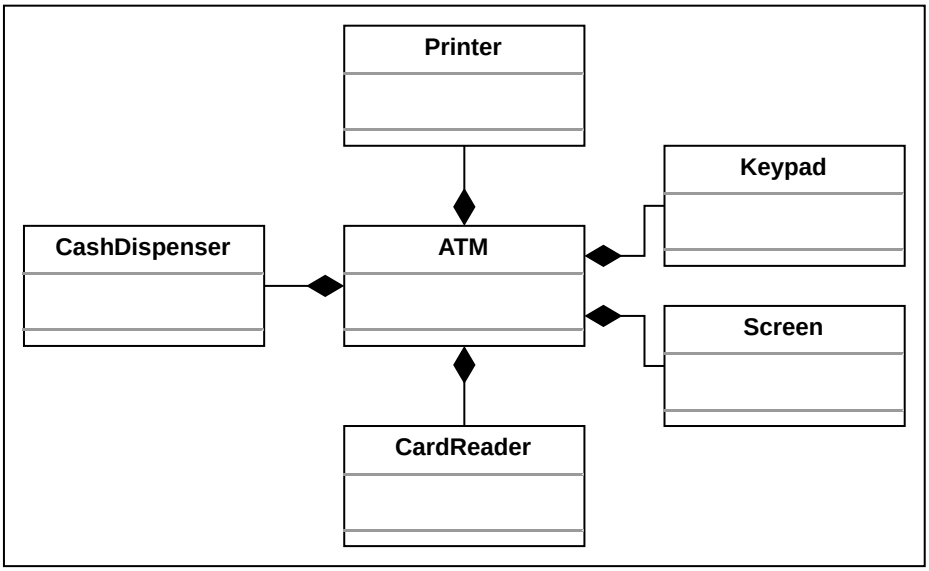


The association relationship between classes

Composition

The class diagram has the following composition relationships.

- The `ATM` class is composed of `Printer`, `Keypad`, `Screen`, `CardReader`, and `CashDispenser`.



The composition relationship between classes

Inheritance

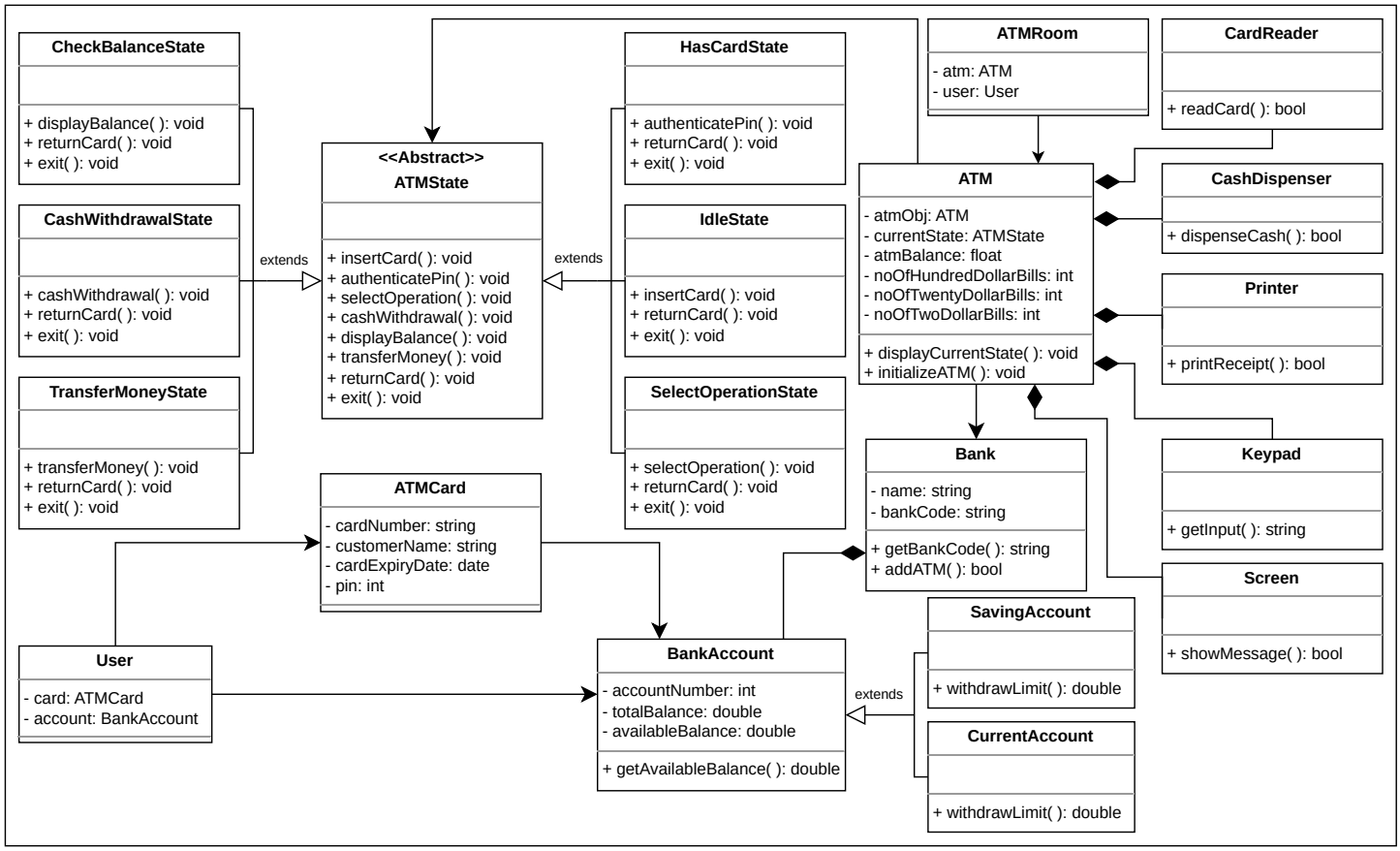
The following classes show an inheritance relationship:

- Both, `SavingAccount` and `CurrentAccount`, extend the `BankAccount` class.
- The `CheckBalanceState`, `CashWithdrawalState`, `TransferMoneyState`, `HasCardState`, `IdleState`, and `SelectOperationState` classes extend the abstract class, `ATMState`.

> **Note:** We have already discussed the inheritance relationship between classes in the component section above one by one.

Class diagram for the ATM System

Here's the complete class diagram for our ATM design:



The class diagram of the ATM design problem

Design pattern

The following design patterns have been used in the class diagram:

- **The Singleton design pattern:** This pattern ensures the existence of a single instance of the ATM at a given moment that can be accessed by multiple users, due to the shared nature of the ATM components.
- **The State design pattern:** This pattern enables the ATM to alter its behavior based on the internal changes in the machine. This way, an ATM can transition from one state to another, like switching from an idle state to displaying an account balance or money withdrawal state, and as soon as all the operations have been performed, it can switch back to the initial idle state.

The following design patterns can also be used to design ATM:

- The Composite design pattern can be used to combine different components of the ATM along with their functionalities.

- The Builder design pattern allows the same processes for a complex object to have different representations. In the ATM system, it can help separate different kinds of transactions like withdrawals, deposits, etc.

Additional requirements

There is a chance that the interviewer might ask about the working of the cash withdrawal process. How can it be implemented in our ATM system? This addition is a bit challenging since we need a system that can withdraw the correct combinations of hundred, twenty, and two dollar bills, respectively, according to the amount specified by the user. The system also needs to work sequentially until the required amount is met.

We will use the Chain of Responsibility design pattern to tackle this addition to our system. This design pattern will ensure the correct division of the dollar bills in the ATM by creating a chain of handlers that forward the requests based on the situation until all the requirements are met.
We have created the following classes to implement the Chain of Responsibility design pattern:

- `CashWithdrawProcessor`: This is associated with the `CashWithdrawalState` class. This abstract class is extended by `HundredDollarWithdrawProcessor`, `TwentyDollarWithdrawProcessor`, and `TwoDollarWithdrawProcessor`.
- `HundredDollarWithdrawProcessor`: This class is derived from `CashWithdrawProcessor` and is responsible for withdrawing hundred-dollar bills based on the requirement.
- `TwentyDollarWithdrawProcessor`: This class is derived from `CashWithdrawProcessor` and is responsible for withdrawing twenty-dollar bills based on the requirement.
- `TwoDollarWithdrawProcessor`: This class is derived from `CashWithdrawProcessor` and is responsible for withdrawing two-dollar bills based on the requirement.

> **Valid Amount:** If the amount entered by the user has a modulus equal to zero with any of the specified bills that the ATM can withdraw, then the amount is considered valid for the transaction. If the amount is invalid, then the transaction will not be processed.

For example, a user wants to withdraw $548. The `HundredDollarWithdrawProcessor` class will start the cash withdrawal using the `cashWithdrawal()` method by taking out five bills of hundred dollars. Now that we have $48 to withdraw for the user which is less than a hundred dollars, the `TwentyDollarWithdrawProcessor` class will start withdrawing dollar bills. This class will take out two bills of twenty dollars with $8 remaining. Since two dollars is less than twenty, the `cashWithdrawal()` method of the `TwoDollarWithdrawProcessor` will take out four bills of $2 for the user. The withdrawal, in this case, is successful.