10. Design ATM

Step 1: Outline Use Cases and Constraints

Use Cases

An automated teller machine (ATM) is an electronic telecommunications instrument that provides clients of a financial institution with access to financial transactions in a public space without the need for a cashier or bank teller. ATMs are necessary as not all bank branches are open every day of the week, and some

customers may not be in a position to visit a bank each time they want to withdraw or deposit money.

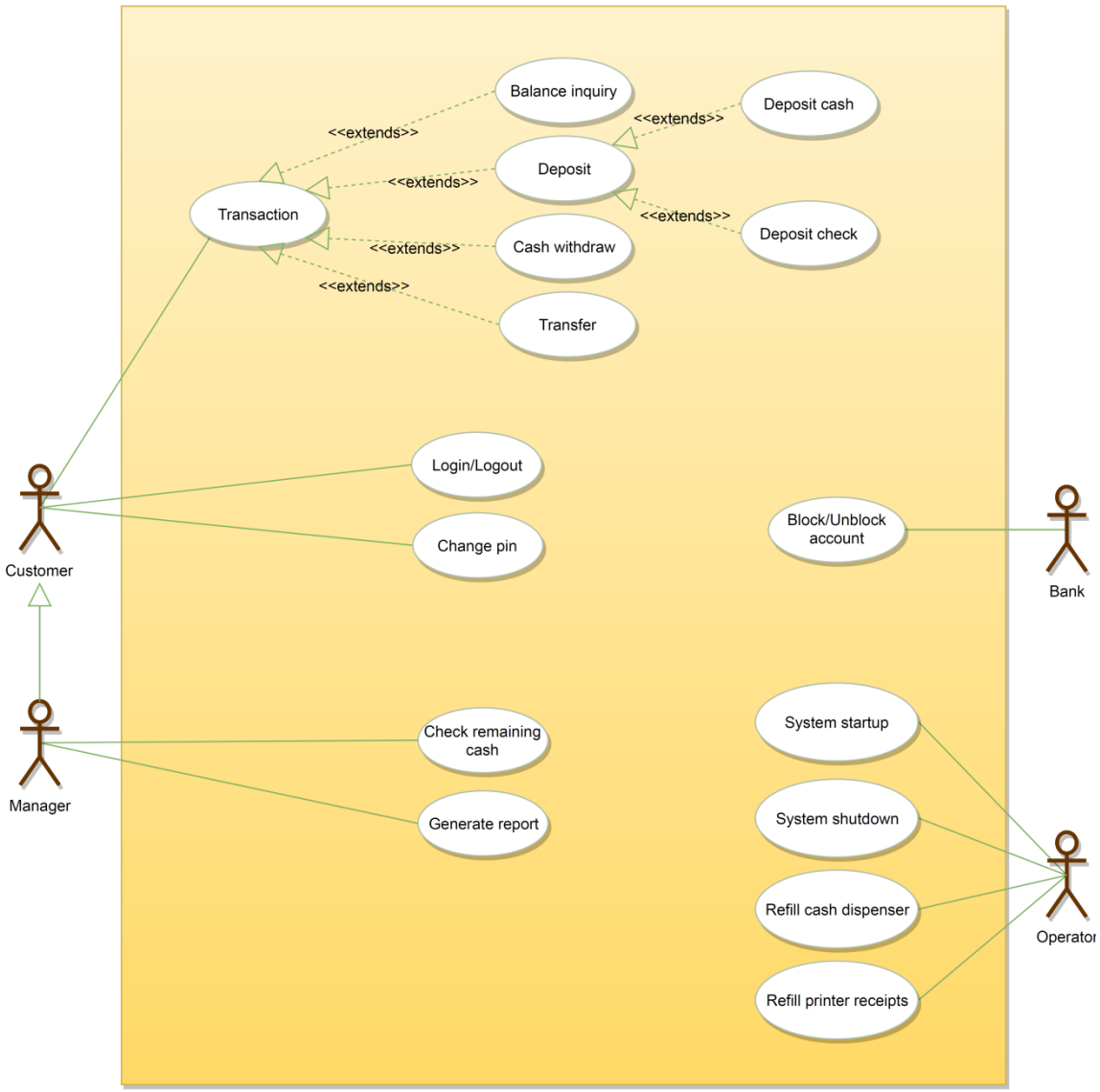
Use Cases:

1. **Balance inquiry** - Users can check their account balance.
2. **Deposit cash** - Users can deposit cash into their account.
3. **Deposit check** - Users can deposit checks.
4. **Withdraw cash** - Users can withdraw money from their checking account.
5. **Transfer funds** - Users can transfer funds to another account.

Out of Scope:

- Online banking services.
- Mobile app integration.
- Loan and credit card services.

Here is the use case diagram of our ATM system:

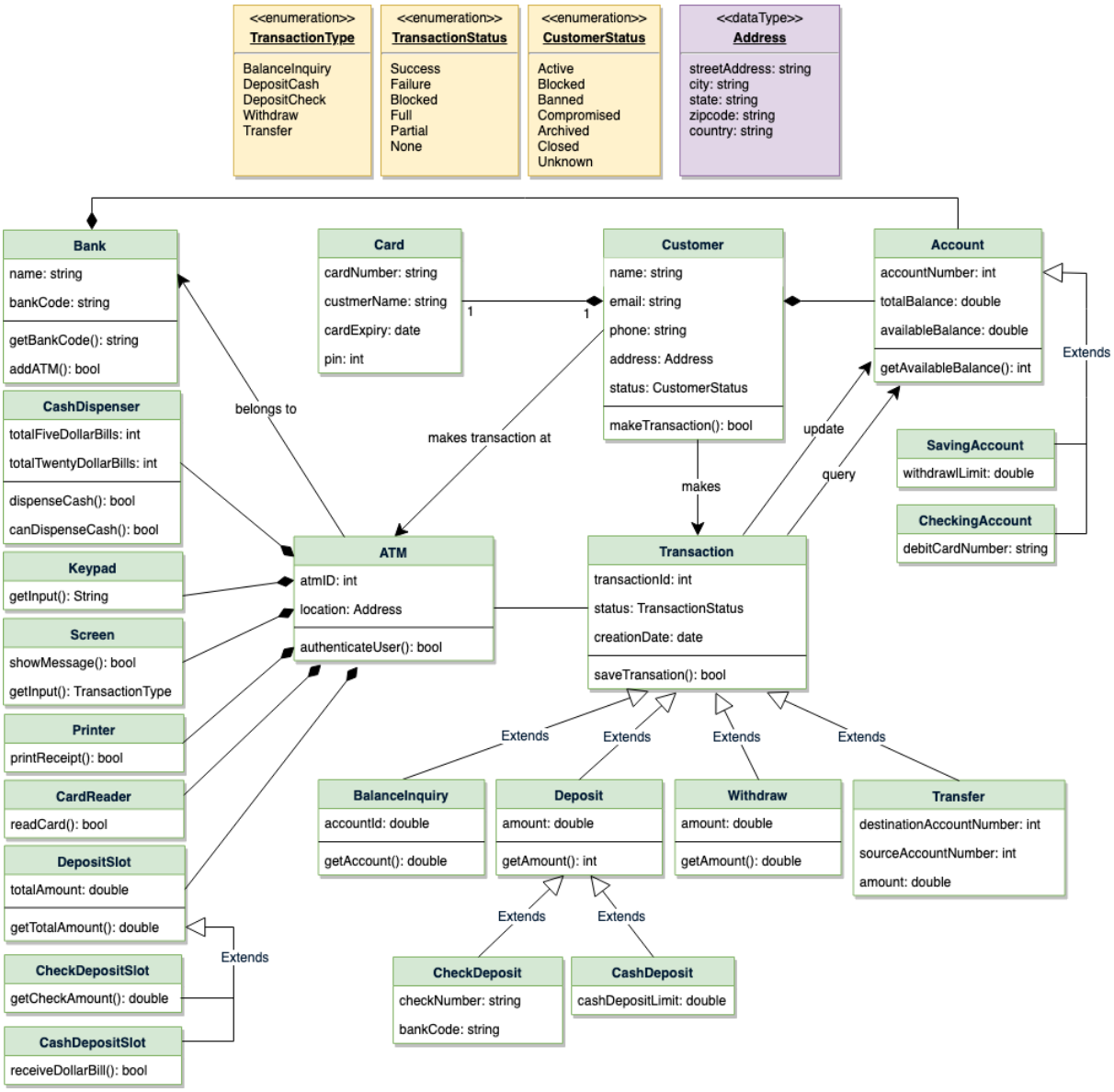


Use Case Diagram for ATM

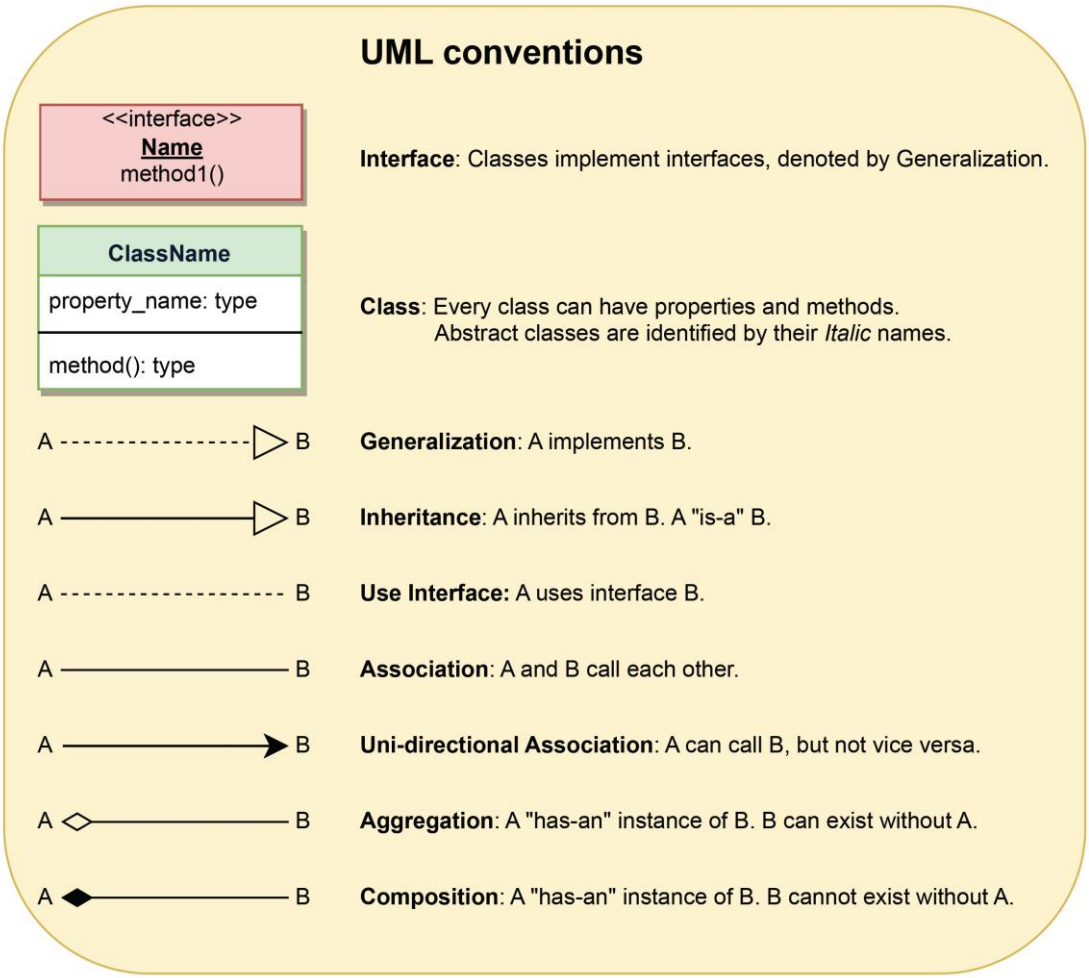
Constraints and Assumptions

- The ATM system should operate 24/7.
- The user can have two types of accounts: Checking and Savings.
- The ATM must authenticate users using an ATM card and PIN.
- Deposits (especially checks) may require manual verification before being added to the account.

- The ATM operator manages the machine, refills cash, and resolves issues.
- The ATM must be connected to a bank network for real-time transactions.



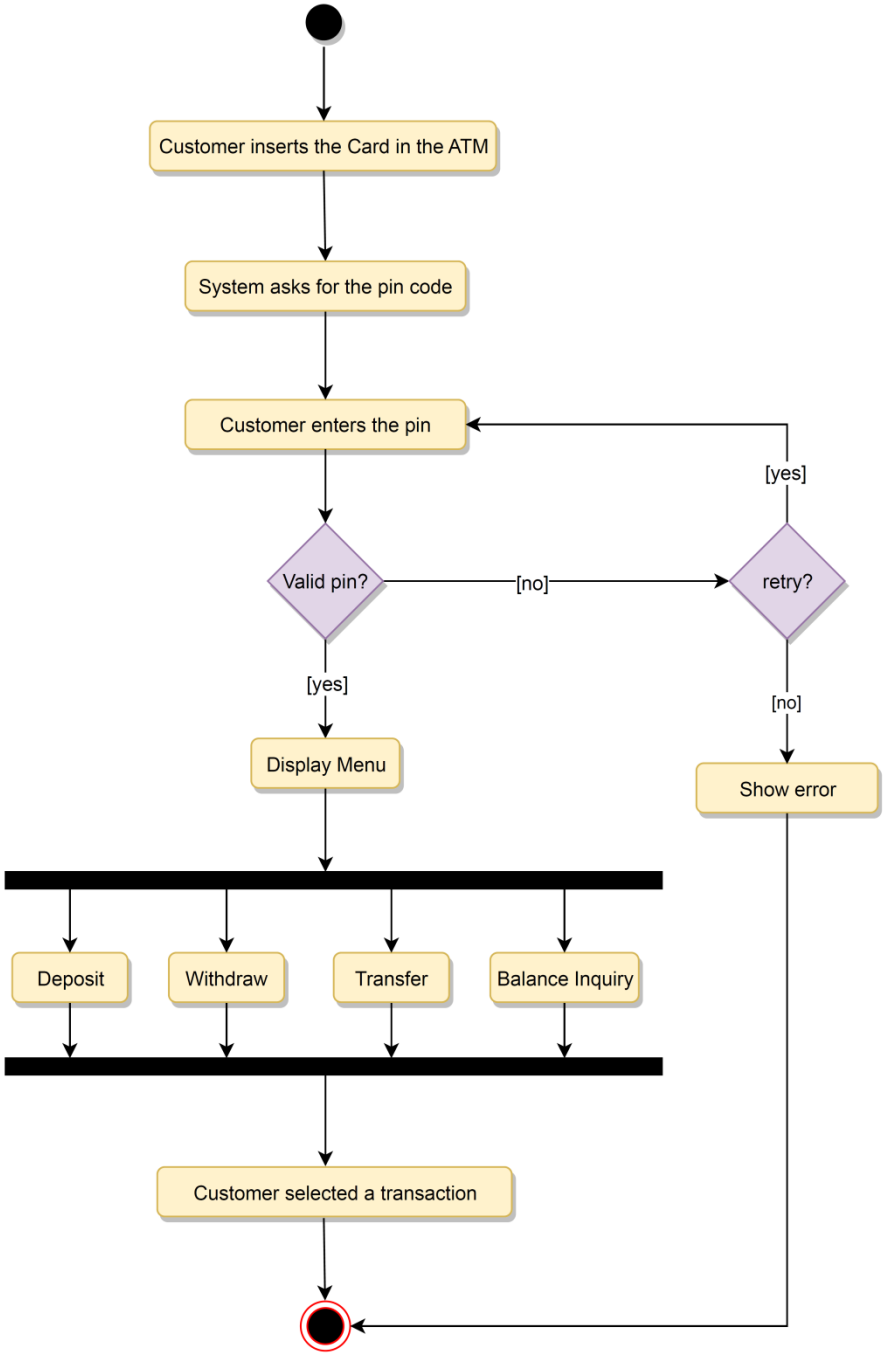
Class Diagram for ATM


UML conventions
<<interface>>
Name
method1()
Interface: Classes implement interfaces, denoted by Generalization.
ClassName
property_name: type
method(): type
Class: Every class can have properties and methods.
Abstract classes are identified by their Italic names.
A
B
Generalization: A implements B.
Inheritance: A inherits from B. A "is-a" B.
Use Interface: A uses interface B.
Association: A and B call each other.
Uni-directional Association: A can call B, but not vice versa.
Aggregation: A "has-an" instance of B. B can exist without A.
Composition: A "has-an" instance of B. B cannot exist without A.

UML for ATM

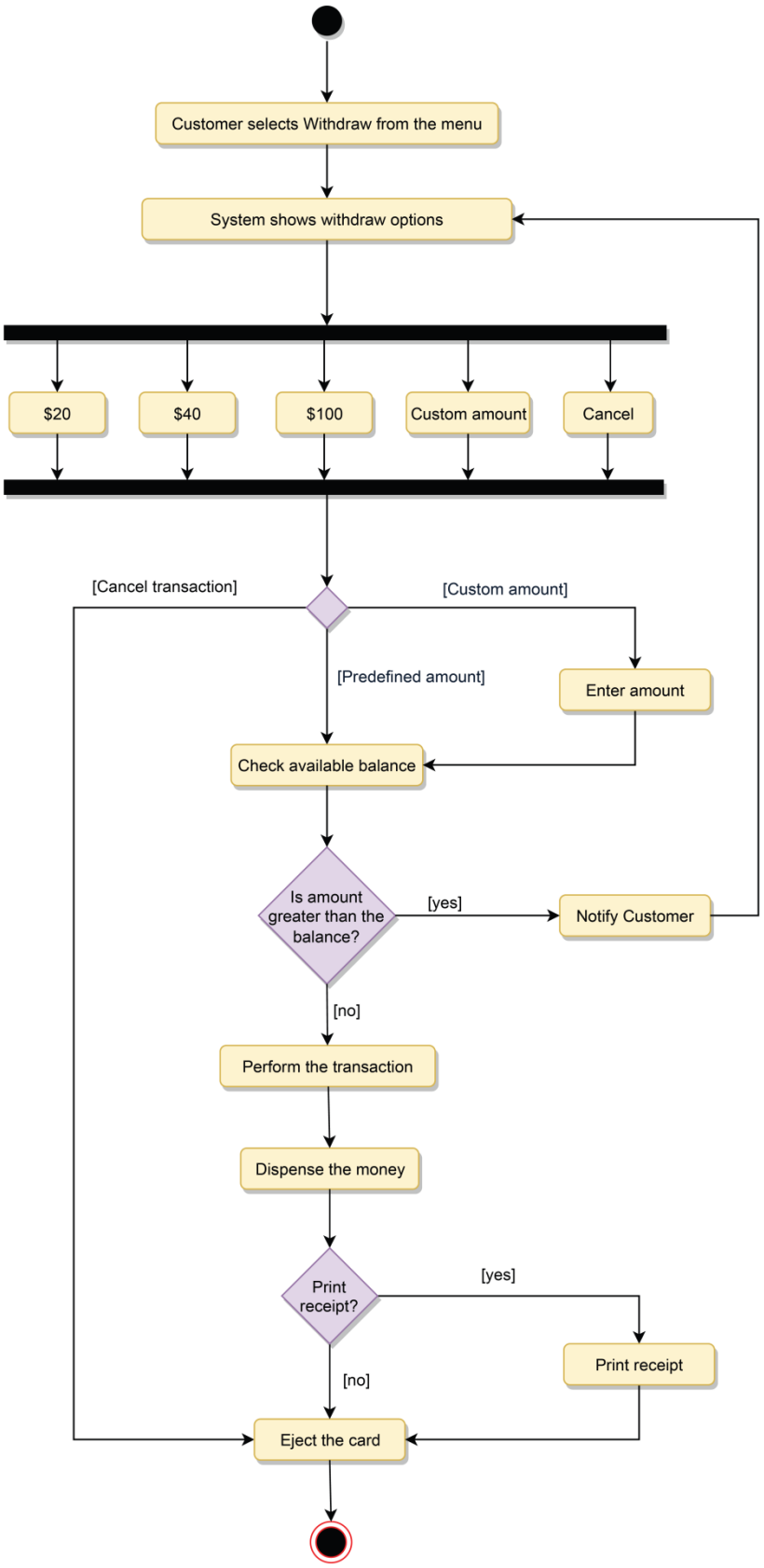
Activity Diagrams

Customer authentication: Following is the activity diagram for a customer authenticating themselves to perform an ATM transaction:

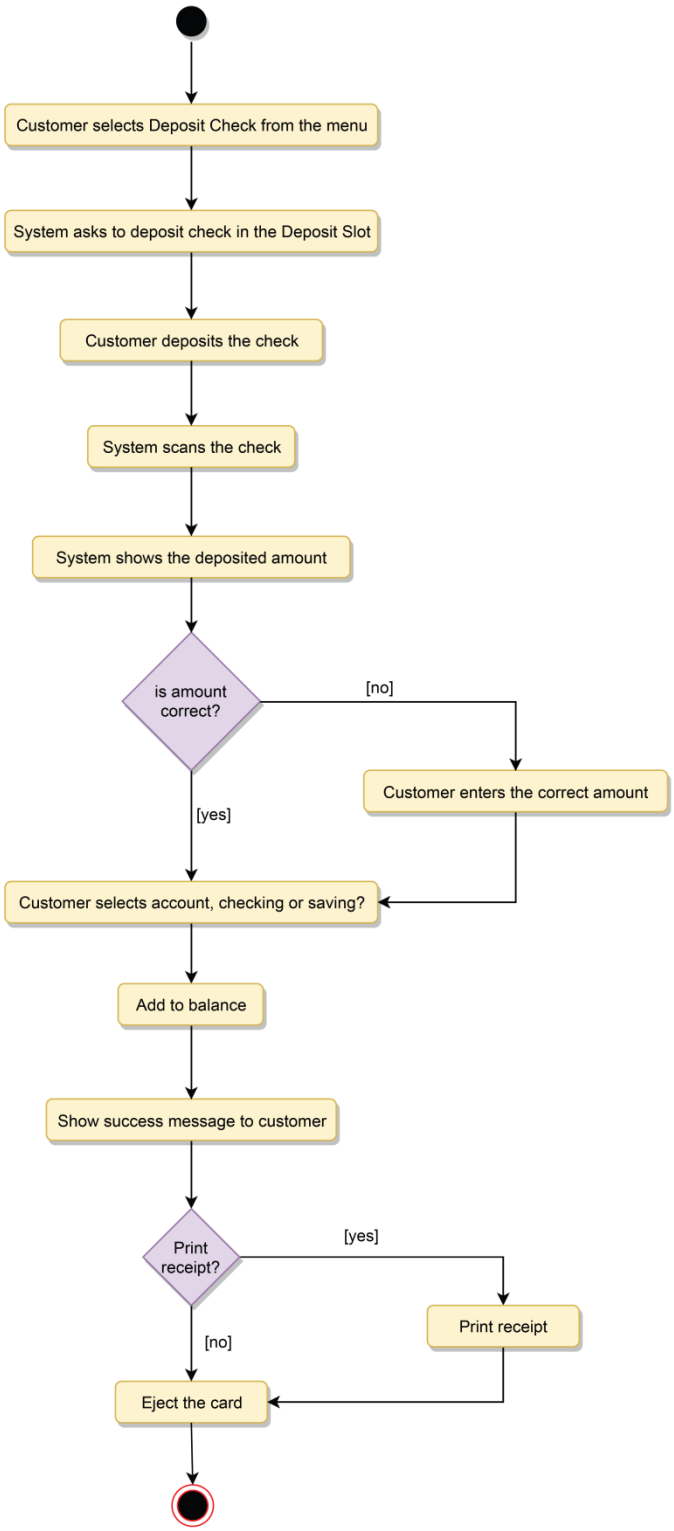


Activity Diagram for ATM Customer Authentication

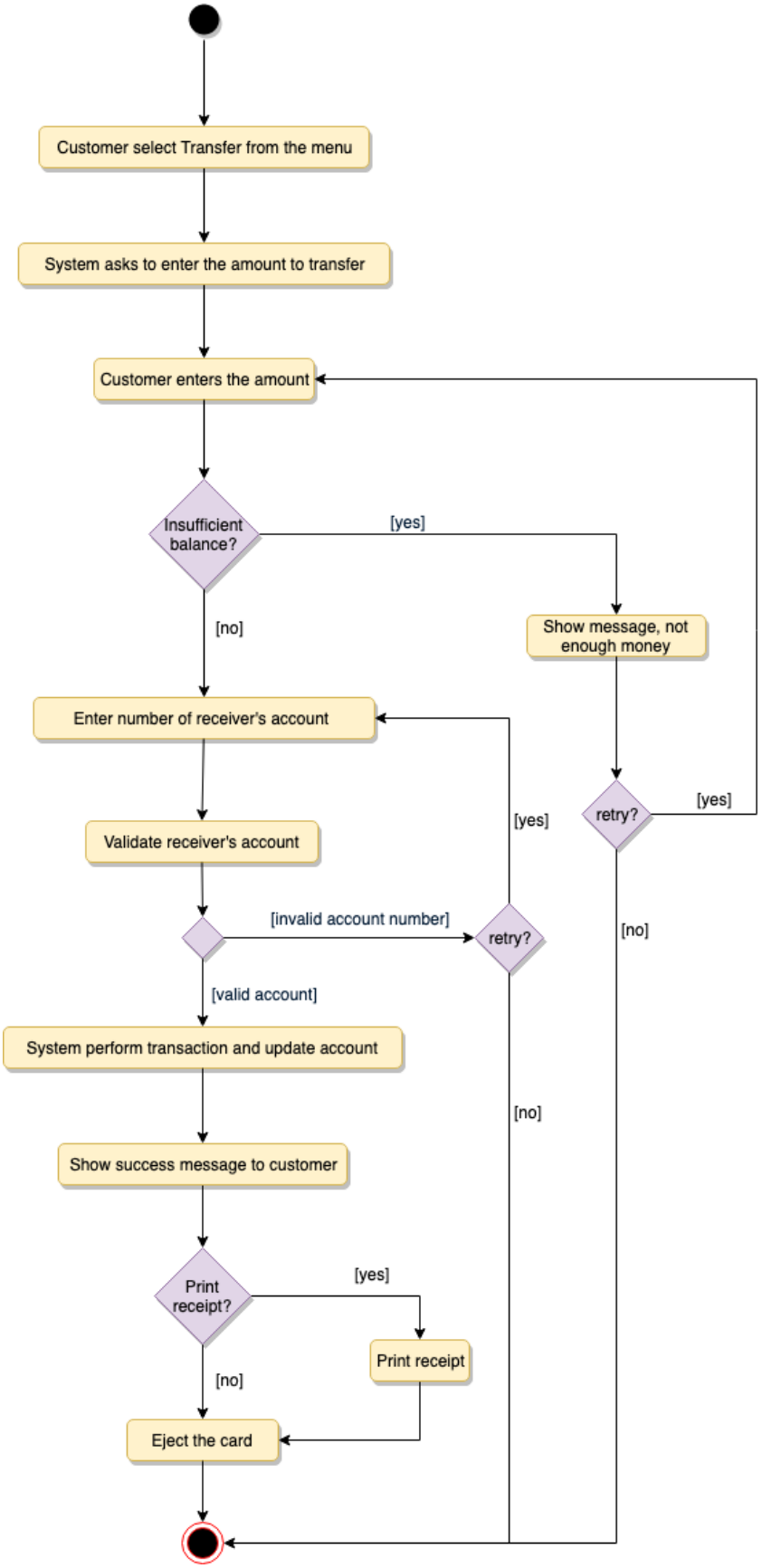
Cash withdraw: Following is the activity diagram for a user withdrawing cash:



Deposit check: Following is the activity diagram for the customer depositing a check:

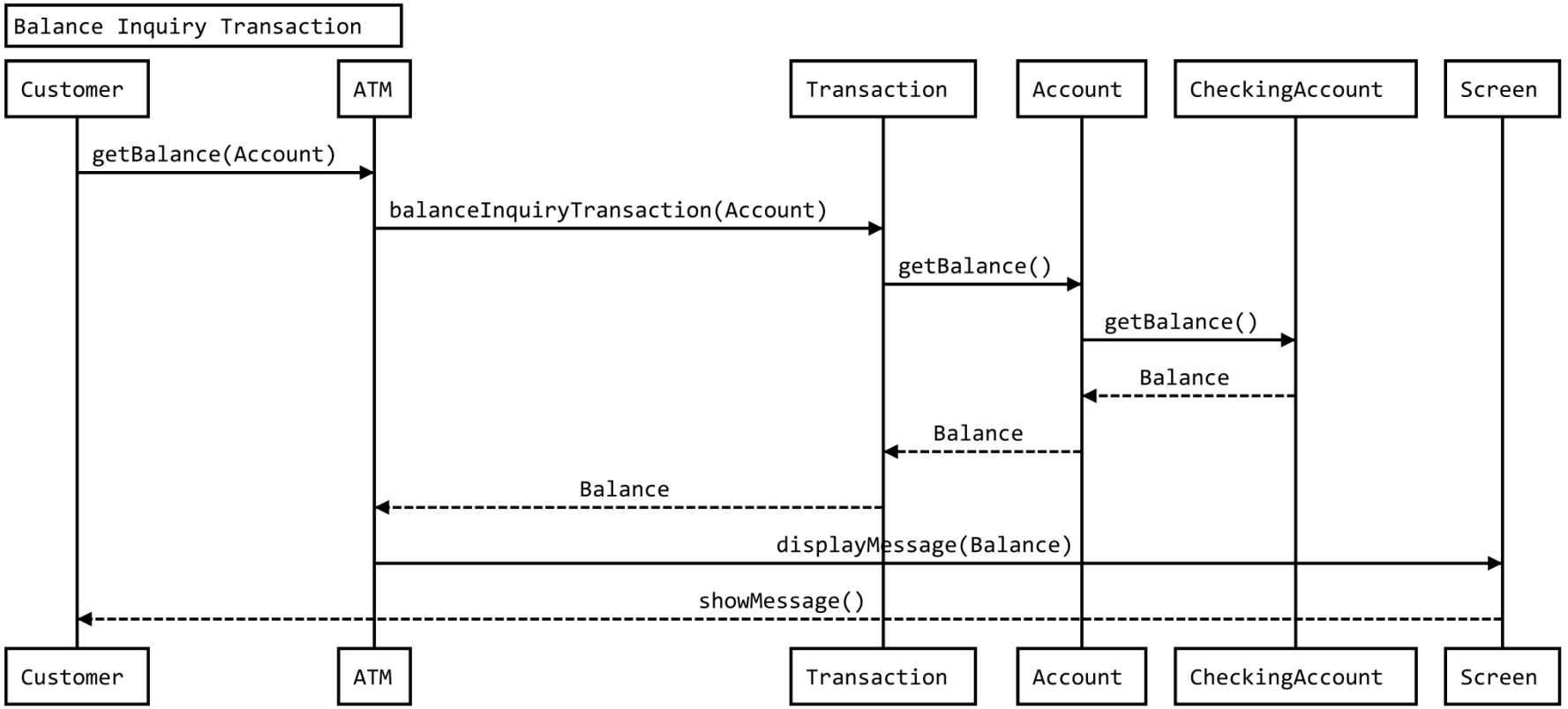


Fund transfer: Following is the activity diagram for a user transferring funds to another account:



Sequence Diagram

Here is the sequence diagram for balance inquiry transaction:



Sequence Diagram for ATM

Step 2: High-Level Design

Main Components

1. ATM Machine
2. User Authentication System (PIN-based authentication)
3. Transaction Processing System
4. Bank Database (stores customer accounts and balances)
5. Hardware Components
    - Card Reader
    - Keypad
    - Screen
    - Cash Dispenser
    - Deposit Slot
    - Printer
    - Communication Network Infrastructure

Step 3: Design Core Components

Class Diagram

The following core components are included in the system:

1. **ATM** - The main ATM system.
2. **Bank** - Represents the bank that owns the ATM.
3. **Customer** - Represents a customer using the ATM.
4. **Account** - Represents user accounts (Checking and Savings).
5. **Card** - Represents ATM cards.
6. **Transaction** - Represents different transaction types.
7. **Hardware Components** - Includes Card Reader, Screen, Keypad, Printer, etc.

Java Code for Core Components

```
// Enum representing different types of transactions (BALANCE_INQUIRY, DEPOSIT_CASH, DEPOSIT_CHECK, WITHDRAW, TRANSFER)
enum TransactionType {
    BALANCE_INQUIRY, DEPOSIT_CASH, DEPOSIT_CHECK, WITHDRAW, TRANSFER
}

// Class representing a Customer with name, email, card, and account information
class Customer {
    private String name;
    private String email;
    private Card card;
    private Account account;

    public Customer(String name, String email, Card card, Account account) {
        this.name = name;
        this.email = email;
        this.card = card;
        this.account = account;
    }
}

// Class representing a Card with card number, customer name, expiry date, and pin
class Card {
    private String cardNumber;
    private String customerName;
    private String expiryDate;
    private int pin;

    public Card(String cardNumber, String customerName, String expiryDate, int pin) {
        this.cardNumber = cardNumber;
        this.customerName = customerName;
        this.expiryDate = expiryDate;
        this.pin = pin;
    }
}

// Abstract class representing a generic Account with account number and total balance
abstract class Account {
    protected String accountNumber;
    protected double totalBalance;

    public Account(String accountNumber, double totalBalance) {
        this.accountNumber = accountNumber;
        this.totalBalance = totalBalance;
    }
}
```

```
// Class representing a Checking Account, inheriting from the Account class
class CheckingAccount extends Account {
    public CheckingAccount(String accountNumber, double totalBalance) {
        super(accountNumber, totalBalance);
    }
}

// Class representing a Saving Account, inheriting from the Account class with an added withdraw limit
class SavingAccount extends Account {
    private double withdrawLimit;

    public SavingAccount(String accountNumber, double totalBalance, double withdrawLimit) {
        super(accountNumber, totalBalance);
        this.withdrawLimit = withdrawLimit;
    }
}

// Class representing an ATM with various components such as cash dispenser, keypad, screen, and printer
class ATM {
    private String atmID;
    private String location;
    private CashDispenser cashDispenser;
    private Keypad keypad;
    private Screen screen;
    private Printer printer;

    public ATM(String atmID, String location) {
        this.atmID = atmID;
        this.location = location;
        this.cashDispenser = new CashDispenser();
        this.keypad = new Keypad();
        this.screen = new Screen();
        this.printer = new Printer();
    }
}

// Class representing the Cash Dispenser with available cash and the ability to dispense cash
class CashDispenser {
    private int availableCash;

    public void dispenseCash(int amount) {
        if (availableCash >= amount) {
            availableCash -= amount;
            System.out.println("Dispensed: " + amount);
        } else {
            System.out.println("Insufficient cash in ATM.");
        }
    }
}
```

```
// Class representing the Keypad, used for receiving user input (e.g., PIN entry)
class Keypad {
    public int getInput() {
        return 1234; // Simulating user input
    }
}

// Class representing the Screen, which displays messages to the user
class Screen {
    public void showMessage(String message) {
        System.out.println(message);
    }
}

// Class representing the Printer, which prints transaction receipts
class Printer {
    public void printReceipt(String transactionDetails) {
        System.out.println("Printing receipt: " + transactionDetails);
    }
}
```

Step 4: Scale the Design

Scaling Considerations

1. **Concurrency**: Multiple ATMs should be able to handle transactions simultaneously.
2. **Failover Mechanism**: In case of network failure, the ATM should retry transactions.
3. **Security**: Implement encryption for data transmission and PIN verification.
4. **Load Balancing**: Distributed ATM servers should handle requests efficiently.
5. **Logging & Monitoring**: Keep track of transactions, errors, and security breaches.

Future Enhancements

- Support for mobile payments and QR code-based authentication.
- AI-powered fraud detection to monitor unusual transactions.
- Cloud-based transaction management for better scalability.

This document provides a structured approach to designing an ATM system while ensuring scalability, security, and efficiency. The Java code for core components demonstrates the key functionalities needed for ATM transactions.