Week

Class Diagram

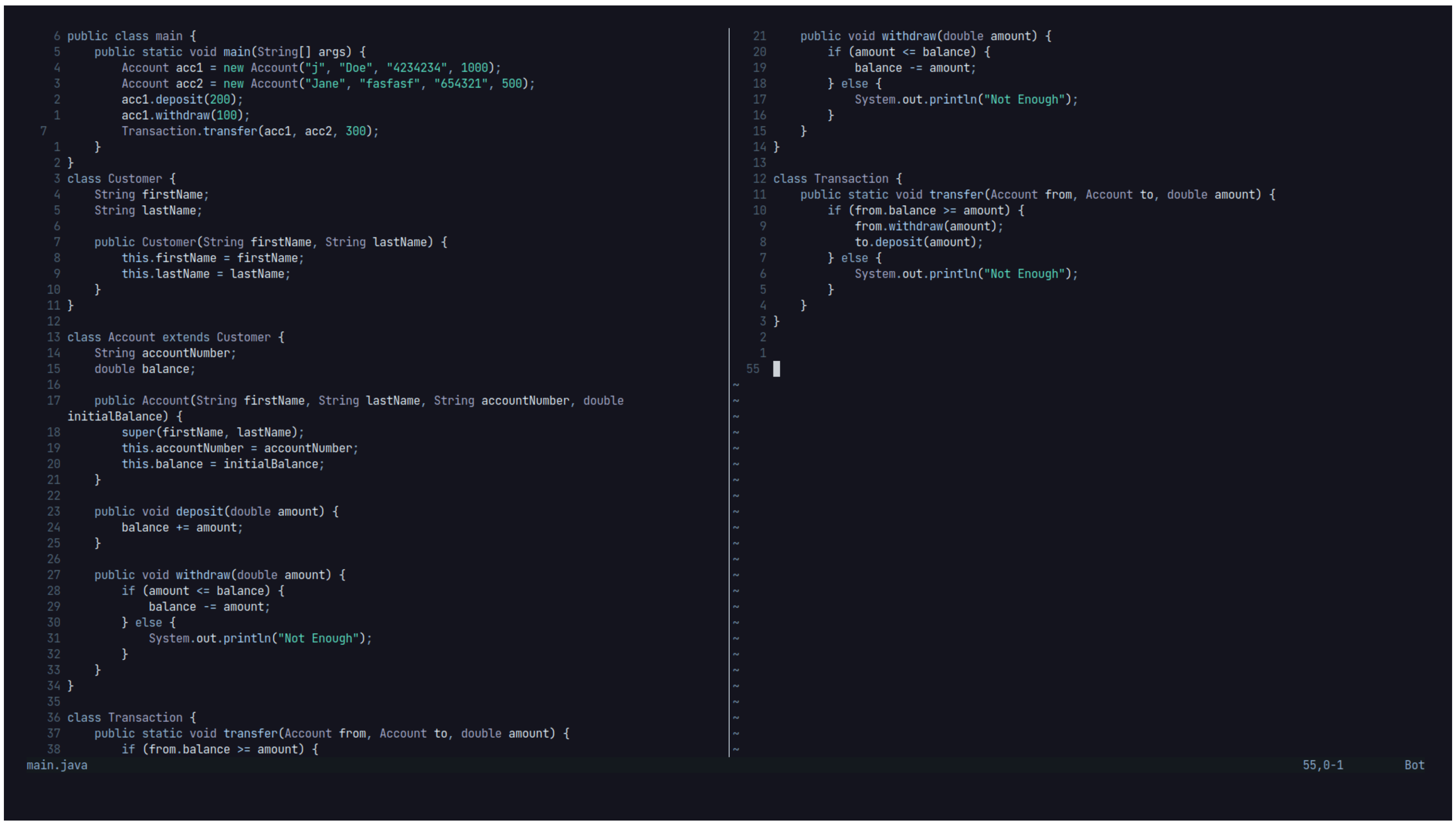
- Write a **code first** and convert the code into the class diagram.

  1.Create a **Customer class:** will hold basic customer information such as the first name and last name.

  2.Create a **Account class:**will hold information such as the account number, account balance and will extend the customer class to allow customer details to be used within the constructor. This class will also hold functions such as deposit and withdraw.

  3.Create a **Transaction class:** will hold a method called transfer to allow accounts to transfer money between them.

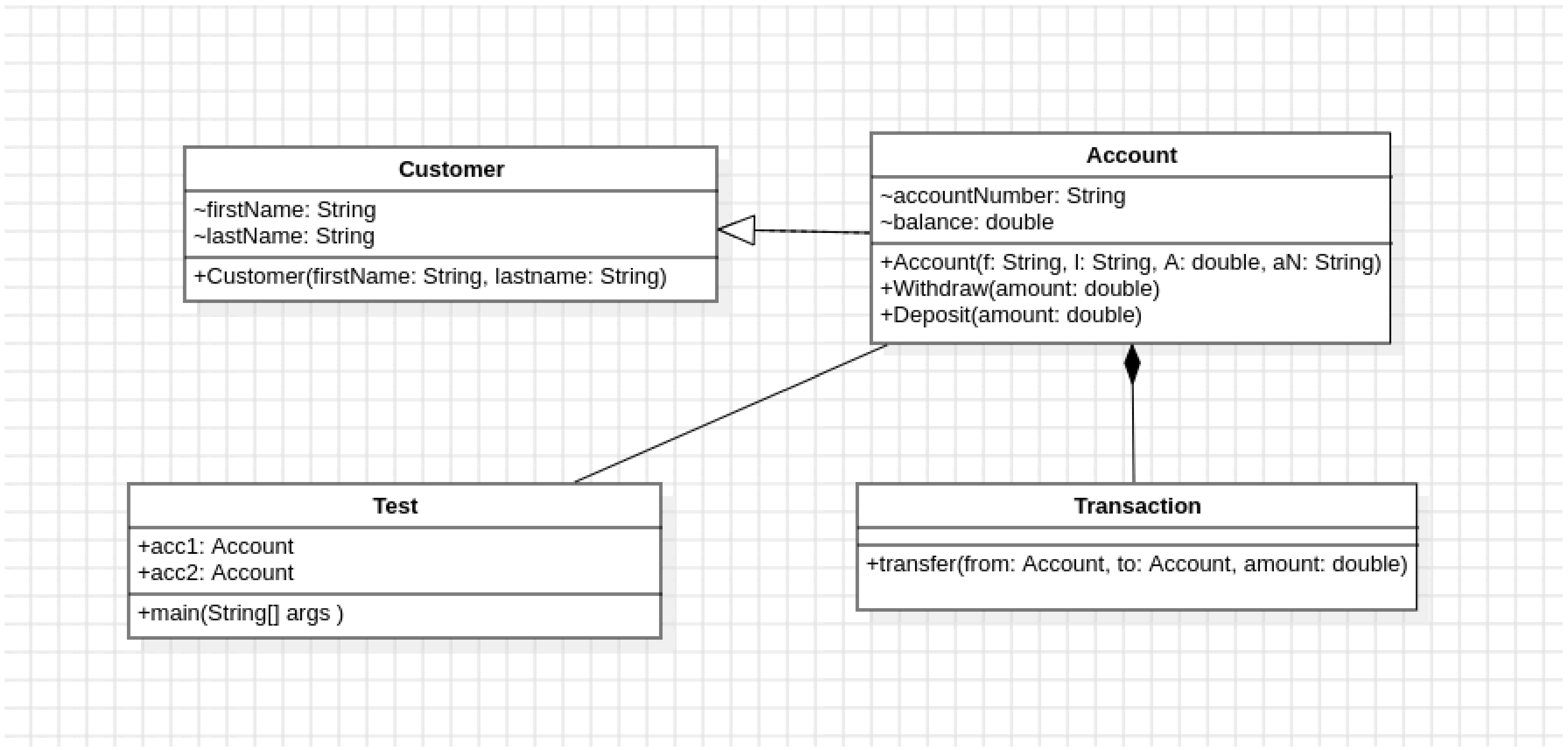
  4.Create a **Test class:**will hold the objects of the above classes and perform the operation needed.



```java
public class main {
    public static void main(String[] args) {
        Account acc1 = new Account("j", "Doe", "4234234", 1000);
        Account acc2 = new Account("Jane", "fasfasf", "654321", 500);
        acc1.deposit(200);
        acc1.withdraw(100);
        Transaction.transfer(acc1, acc2, 300);
    }
}
class Customer {
    String firstName;
    String lastName;

    public Customer(String firstName, String lastName) {
        this.firstName = firstName;
        this.lastName = lastName;
    }
}

class Account extends Customer {
    String accountNumber;
    double balance;

    public Account(String firstName, String lastName, String accountNumber, double
initialBalance) {
        super(firstName, lastName);
        this.accountNumber = accountNumber;
        this.balance = initialBalance;
    }

    public void deposit(double amount) {
        balance += amount;
    }

    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("Not Enough");
        }
    }
}

class Transaction {
    public static void transfer(Account from, Account to, double amount) {
        if (from.balance >= amount) {
```

```java
    public void withdraw(double amount) {
        if (amount <= balance) {
            balance -= amount;
        } else {
            System.out.println("Not Enough");
        }
    }
}

class Transaction {
    public static void transfer(Account from, Account to, double amount) {
        if (from.balance >= amount) {
            from.withdraw(amount);
            to.deposit(amount);
        } else {
            System.out.println("Not Enough");
        }
    }
}


```

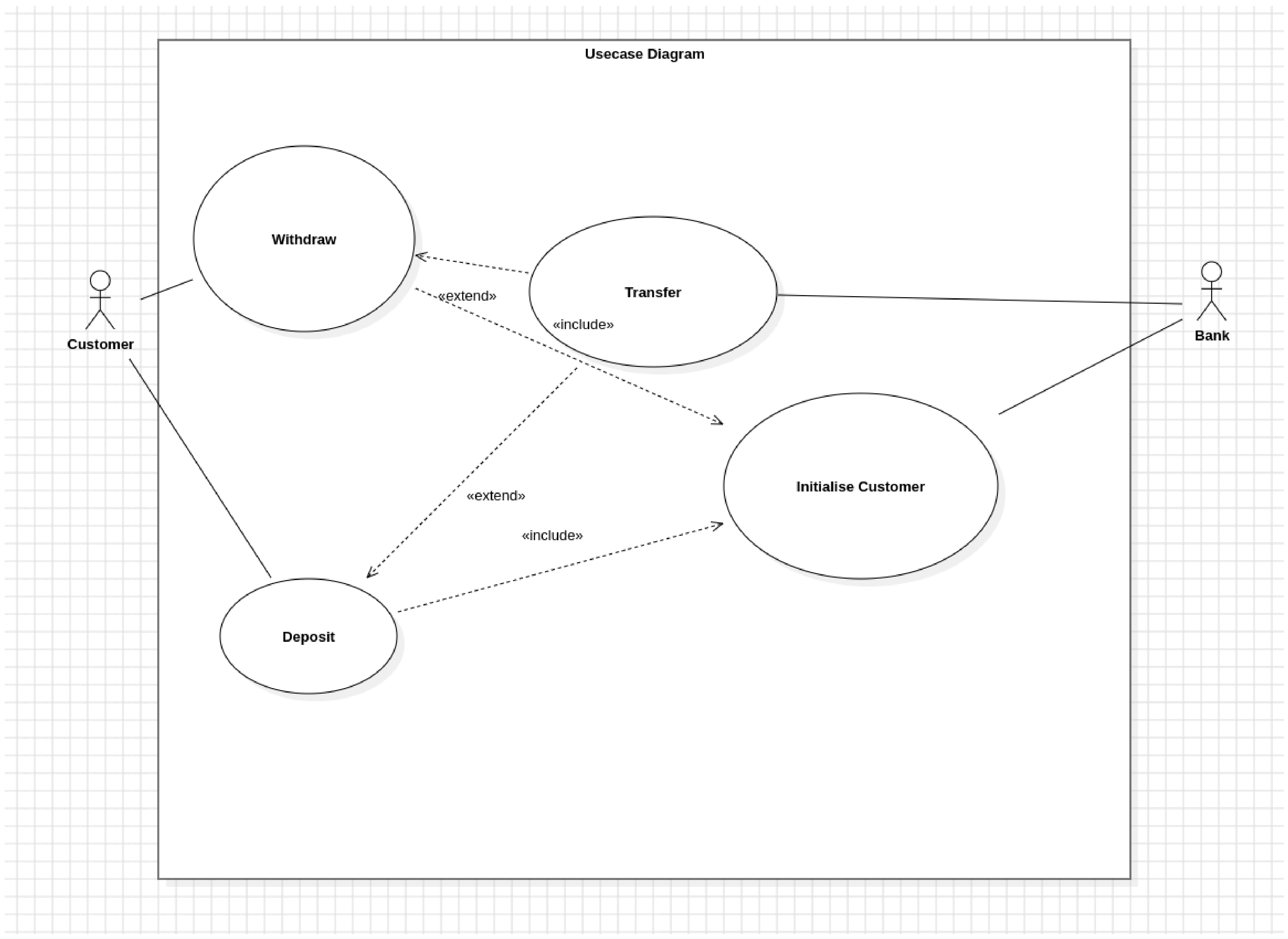
```
main.java                                                                55,0-1        Bot
```


Customer
~firstName: String
~lastName: String
+Customer(firstName: String, lastname: String)
Account
~accountNumber: String
~balance: double
+Account(f: String, l: String, A: double, aN: String)
+Withdraw(amount: double)
+Deposit(amount: double)
Test
+acc1: Account
+acc2: Account
+main(String[] args)
Transaction
+transfer(from: Account, to: Account, amount: double)

UseCase Diagram

1. What are the primary actors described by the code above?
   Customer Bank
2. What are the primary use cases (actions) described by the code above.
   Withdraw, Deposit , Transfer , Initialise Customer,
3. Assocations in the code above.

4. Relations(**include and exclude**) if needed.
5. Perform steps 1-4 and convert it to **useCase diagram.**



Sequence Diagram

1. What are the **objects** involved in the banking system above?
   Customer , Account , Transfer
2. Observe and **find the sequence** of the code for sequence Diagram.

Test Initialises Customer Instances, Customer deposits money , withdraws money, transfers money.

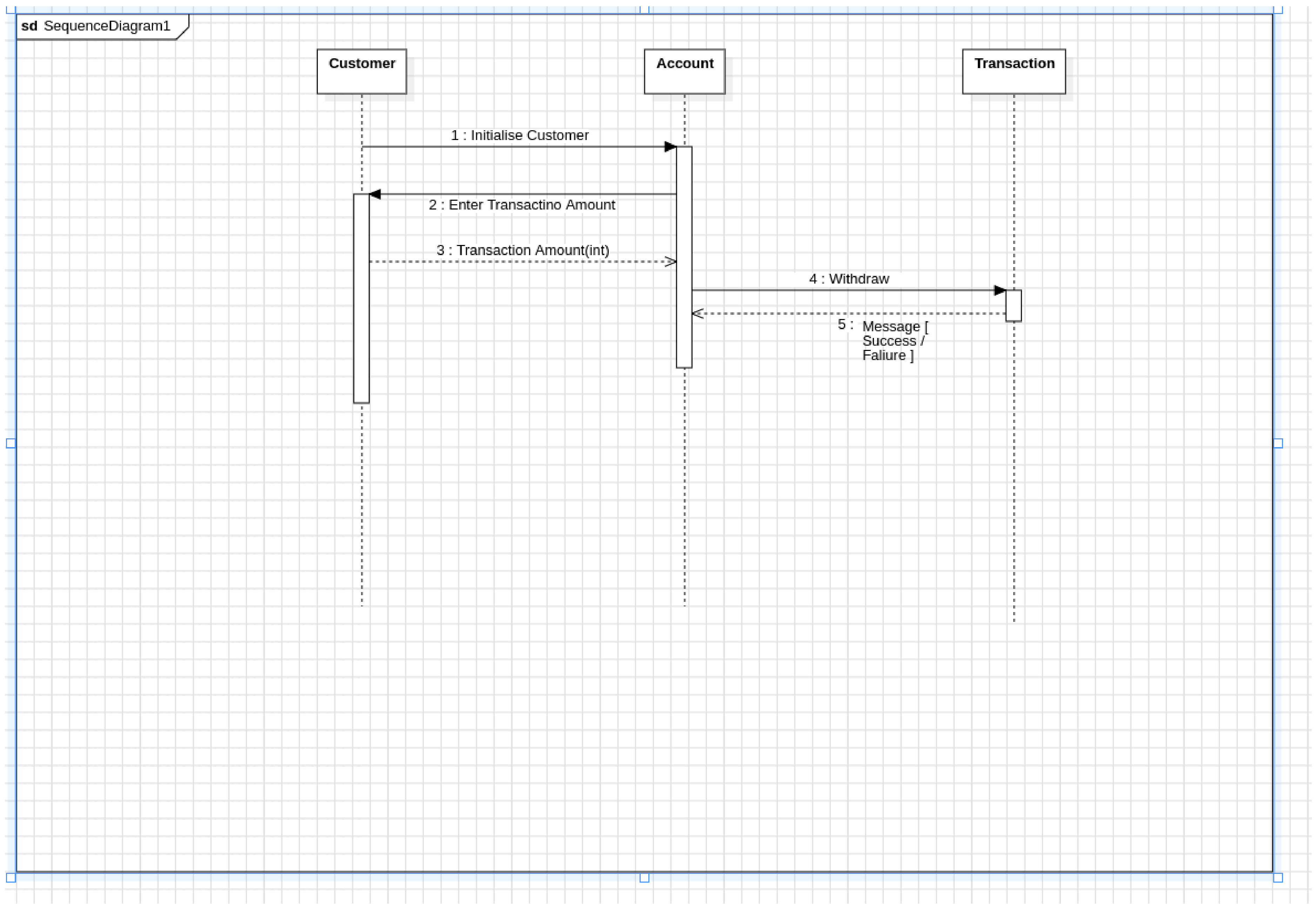
3. Are there any **synchronous and asynchronous** messages?

All of them are synchronous

4. What are the possible responses?

Success or failure

5. Perform steps mentioned above and convert it to **sequence diagram?**


sd SequenceDiagram1
Customer
Account
Transaction
1 : Initialise Customer
2 : Enter Transactino Amount
3 : Transaction Amount(int)
4 : Withdraw
5 : Message [Success / Faliure]