

Distributed and Cloud System Computing (5CS022)

AKKA Framework (Task 1)

Student Id: 2332244

Student Name: Naomi Thing

Group: L5CG4

Module Leader: Mr. Deepshon Shrestha

Tutor: Mr. Deepshon Shrestha

Submission Date: 18th May, 2024.

Table of Contents

1.	Main.java	1
2.	BankAccount.java	2
3.	Withdraw.java	3
4.	Deposit.java	3
5.	Output	4

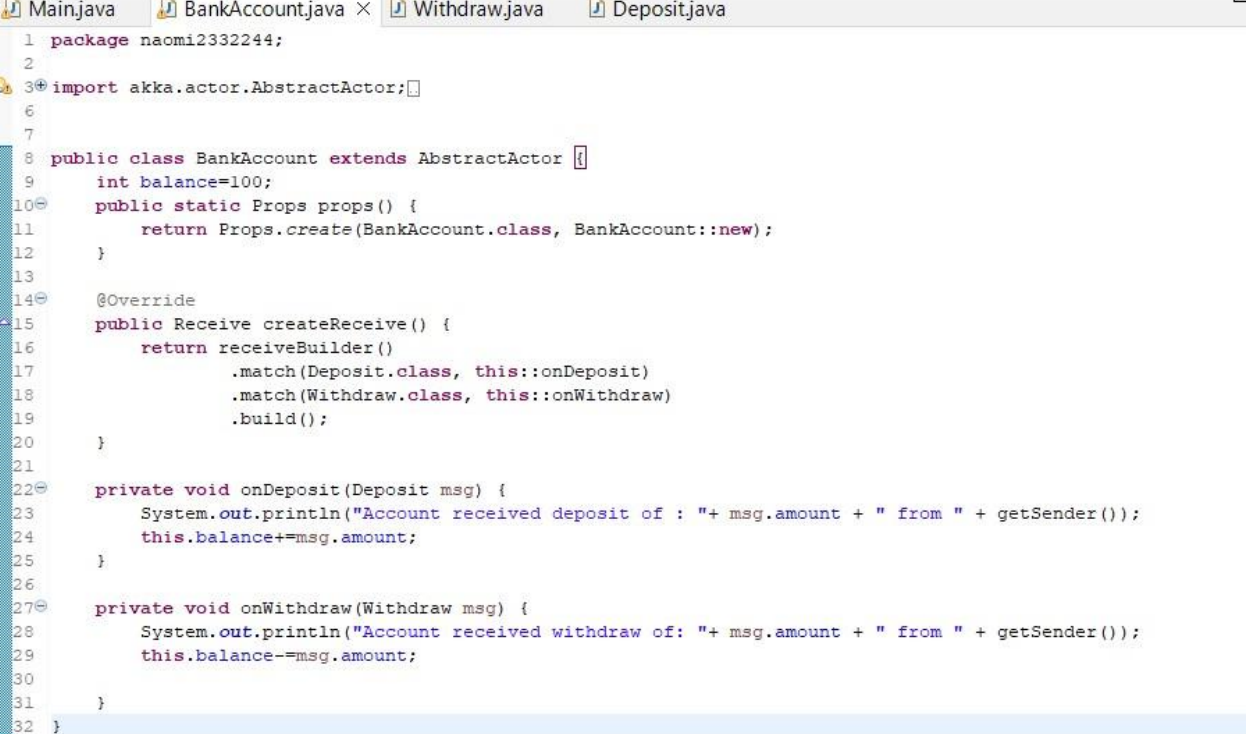
1. Main.java



```
1 package naomi2332244;
2
3 import akka.actor.typed.Behavior;
4
13
14 class Main {
15
16     public static void main(String[] args) {
17
18         ActorSystem system = ActorSystem.create();
19         ActorRef bankAccRef = system.actorOf(Props.create(BankAccount.class));
20
21         for (int i = 0; i < 10; i++) {
22             int amount = (int) (Math.random() * 2001) - 1000; // Random value between -1000 and 1000
23             if (amount > 0) {
24                 // Deposit
25                 bankAccRef.tell(new Deposit(amount), ActorRef.noSender());
26             } else {
27                 // Withdrawal
28                 bankAccRef.tell(new Withdraw(-amount), ActorRef.noSender());
29             }
30         }
31
32         // Terminate the system after processing all transactions
33         system.terminate();
34     }
35 }
36
37
```

The code below generates a simple financial simulation using the Akka framework. It shows an `ActorSystem` and creates a `BankAccount` actor to handle transactions. Within the main function, a loop generates ten random transaction amounts ranging from -1000 to 1000 before sending a `Deposit` or `Withdraw` message to the `BankAccount` actor based on the amount's sign. Positive sums start deposits, while negative sums initiate withdrawals. After processing all transactions, the actor system is shut down to show how Akka actors manage concurrency and asynchronous message handling in a distributed system.

2. BankAccount.java

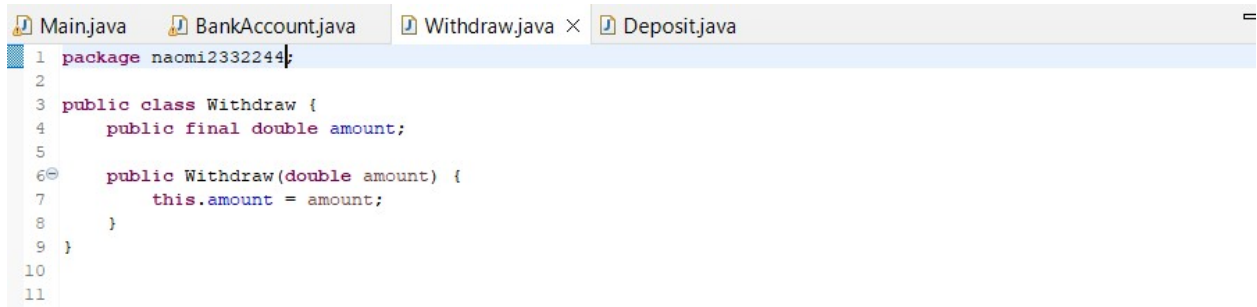


```
1 package naomi2332244;
2
3 import akka.actor.AbstractActor;
4
5
6
7
8 public class BankAccount extends AbstractActor {
9     int balance=100;
10    public static Props props() {
11        return Props.create(BankAccount.class, BankAccount::new);
12    }
13
14    @Override
15    public Receive createReceive() {
16        return receiveBuilder()
17            .match(Deposit.class, this::onDeposit)
18            .match(Withdraw.class, this::onWithdraw)
19            .build();
20    }
21
22    private void onDeposit(Deposit msg) {
23        System.out.println("Account received deposit of : " + msg.amount + " from " + getSender());
24        this.balance+=msg.amount;
25    }
26
27    private void onWithdraw(Withdraw msg) {
28        System.out.println("Account received withdraw of: " + msg.amount + " from " + getSender());
29        this.balance-=msg.amount;
30    }
31 }
32 }
```

The Akka framework's `BankAccount` actor class manages deposits and withdrawals. It supports an account balance that is initially set to 100 and processes `Deposit` and `Withdraw` messages by adjusting this number accordingly. The static `props` method configures a `BankAccount` actor.

The `createReceive` method describes the message processing system, which sends deposits to the `onDeposit` function and withdrawals to the `onWithdraw` method. Both approaches generate transaction data and update the balance, demonstrating Akka's ability to manage state and concurrency in a distributed system.

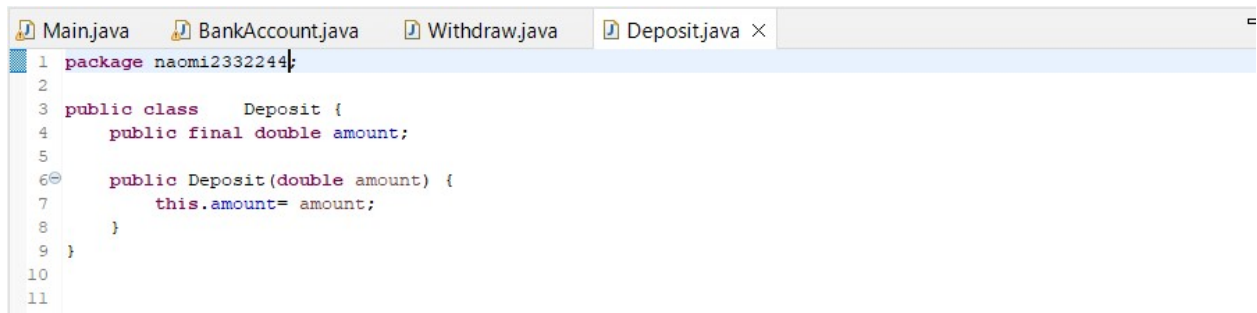
3. Withdraw.java



```
1 package naomi2332244;
2
3 public class Withdraw {
4     public final double amount;
5
6     public Withdraw(double amount) {
7         this.amount = amount;
8     }
9 }
10
11
```

The `Withdraw` class is a withdrawal transaction in a banking simulation. It is a component of the `naomi2332244` package and has a single public final field, `amount`, which stores the withdrawal amount in doubles. The class has a constructor that sets this `amount` field to the supplied value, guaranteeing that the withdrawal amount is set whenever a new instance of the `Withdraw` class is generated. This class is used in the Akka actor system to hold withdrawal transaction data sent to the `BankAccount` actor.

4. Deposit.java

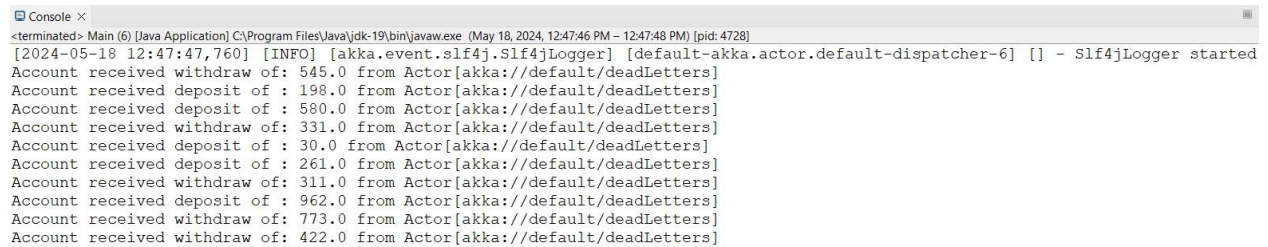


```
1 package naomi2332244;
2
3 public class Deposit {
4     public final double amount;
5
6     public Deposit(double amount) {
7         this.amount = amount;
8     }
9 }
10
11
```

The `Deposit` class is used in the banking model to stand for a deposit. It is a part of `naomi2332244` package and has a single public final field, `amount`, which stores the deposit amount as a double value. The class has a constructor that initializes this `amount` property with the given value, ensuring that the deposit amount is set when a `Deposit` instance is created. This

class is used in the Akka actor system to encapsulate deposit transaction information supplied to the 'BankAccount' actor.

5. Output



```
<terminated> Main (6) [Java Application] C:\Program Files\Java\jdk-19\bin\javaw.exe (May 18, 2024, 12:47:46 PM - 12:47:48 PM) [pid: 4728]
[2024-05-18 12:47:47,760] [INFO] [akka.event.slf4j.Slf4jLogger] [default-akka.actor.default-dispatcher-6] [] - Slf4jLogger started
Account received withdraw of: 545.0 from Actor[akka://default/deadLetters]
Account received deposit of : 198.0 from Actor[akka://default/deadLetters]
Account received deposit of : 580.0 from Actor[akka://default/deadLetters]
Account received withdraw of: 331.0 from Actor[akka://default/deadLetters]
Account received deposit of : 30.0 from Actor[akka://default/deadLetters]
Account received deposit of : 261.0 from Actor[akka://default/deadLetters]
Account received withdraw of: 311.0 from Actor[akka://default/deadLetters]
Account received deposit of : 962.0 from Actor[akka://default/deadLetters]
Account received withdraw of: 773.0 from Actor[akka://default/deadLetters]
Account received withdraw of: 422.0 from Actor[akka://default/deadLetters]
```