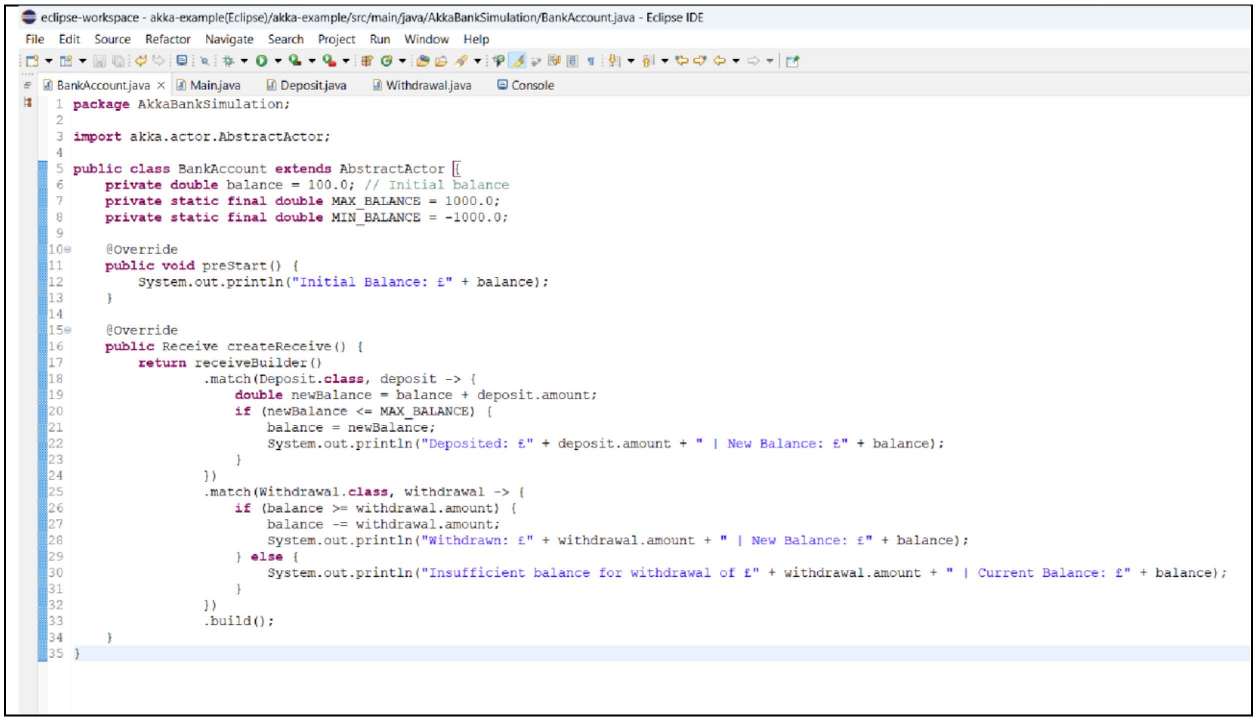
Using the Akka Actor framework to create a simulation of a bank account.

The bank account simulation is based on the Akka Actor Framework which is employed to represent concurrent transactions of a bank account. BankAccount, the function of which is to handle the account's balance and transaction processing, is the actor that performs this task. The system, once it has started running, shows the initial balance. It specifies the rules for deposits and withdrawals, so that the balance does not go out of the preset range. Concurrency transactions are created and sent to the BankAccount actor asynchronously. Every transaction is contained in a message and processed separately by the actor, thus, it is possible to handle the transactions in a timely and at the same time efficiently manner. The actor system supervises the lifecycle of actors and is also in charge of message passing between them. The system, after processing all transactions, outputs its final termination. With the help of Akka's actor model, the simulation is made to be concurrency, isolation, and fault tolerance, thus, it is the suitable for the development of scalable and robust banking systems.

Below classes with Screenshots explain the purpose of each class in the Akka bank account simulation, thus, providing a precise understanding of the classes and their roles in the simulation.

BankAccount.java:



eclipse-workspace - akka-example(Eclipse)/akka-example/src/main/java/AkkaBankSimulation/BankAccount.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
BankAccount.java
Main.java
Deposit.java
Withdrawal.java
Console

```java
package AkkaBankSimulation;

import akka.actor.AbstractActor;

public class BankAccount extends AbstractActor {
    private double balance = 100.0; // Initial balance
    private static final double MAX_BALANCE = 1000.0;
    private static final double MIN_BALANCE = -1000.0;

    @Override
    public void preStart() {
        System.out.println("Initial Balance: £" + balance);
    }

    @Override
    public Receive createReceive() {
        return receiveBuilder()
                .match(Deposit.class, deposit -> {
                    double newBalance = balance + deposit.amount;
                    if (newBalance <= MAX_BALANCE) {
                        balance = newBalance;
                        System.out.println("Deposited: £" + deposit.amount + " | New Balance: £" + balance);
                    }
                })
                .match(Withdrawal.class, withdrawal -> {
                    if (balance >= withdrawal.amount) {
                        balance -= withdrawal.amount;
                        System.out.println("Withdrawn: £" + withdrawal.amount + " | New Balance: £" + balance);
                    } else {
                        System.out.println("Insufficient balance for withdrawal of £" + withdrawal.amount + " | Current Balance: £" + balance);
                    }
                })
                .build();
    }
}
```

Description: This class represents a bank account in the simulation. It is an Akka actor that manages depositing and withdrawing money from the account.

Methods:

preStart(): This method is called when the actor is started. It prints the initial balance of the bank account.

createReceive(): This method defines the behavior of the actor. It specifies how the actor should respond to different types of messages.

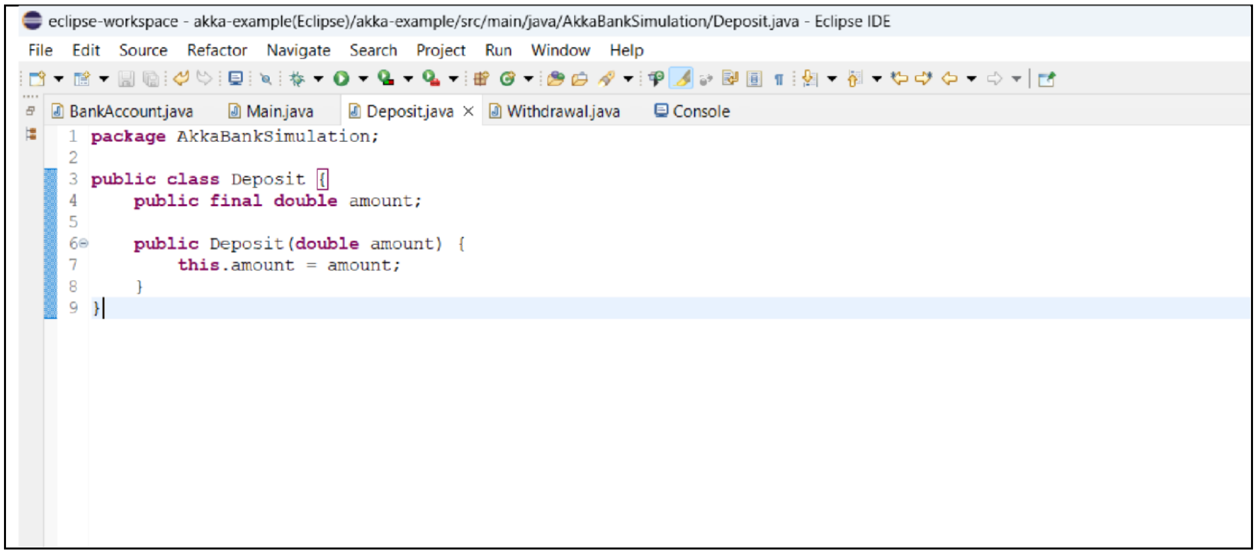
Attributes:

balance: Represents the current balance of the bank account.

MAX_BALANCE: Represents the maximum allowed balance for the account.

MIN_BALANCE: Represents the minimum allowed balance for the account.

Deposit.java:


eclipse-workspace - akka-example(Eclipse)/akka-example/src/main/java/AkkaBankSimulation/Deposit.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
BankAccount.java
Main.java
Deposit.java
Withdrawal.java
Console

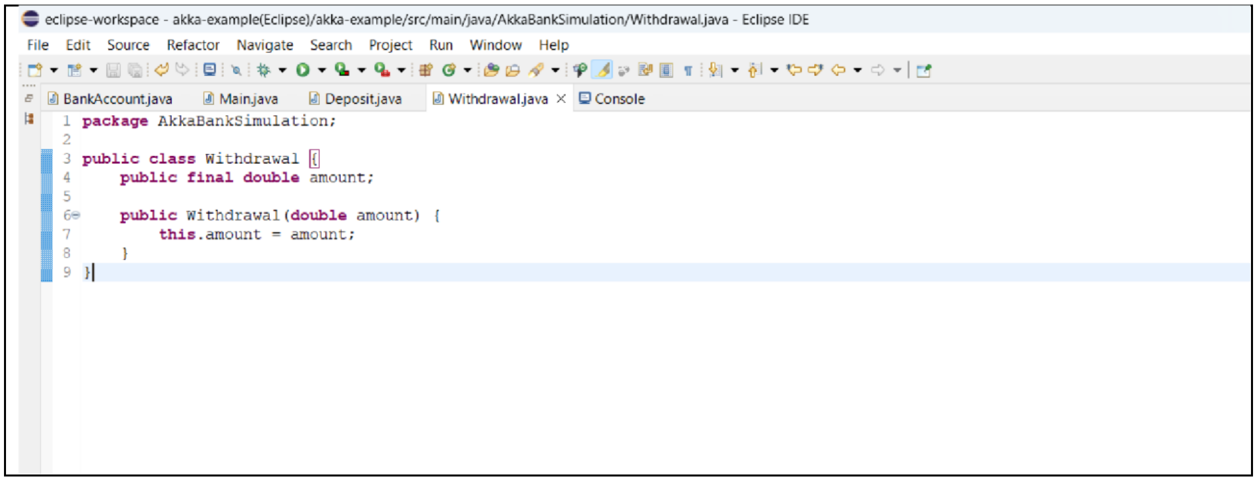
```java
package AkkaBankSimulation;

public class Deposit {
    public final double amount;

    public Deposit(double amount) {
        this.amount = amount;
    }
}
```

Description: This class represents a deposit message. It is used to send a deposit amount to the bank account actor.

Attributes:

amount: Represents the amount of money to be deposited.

Withdrawal.java:



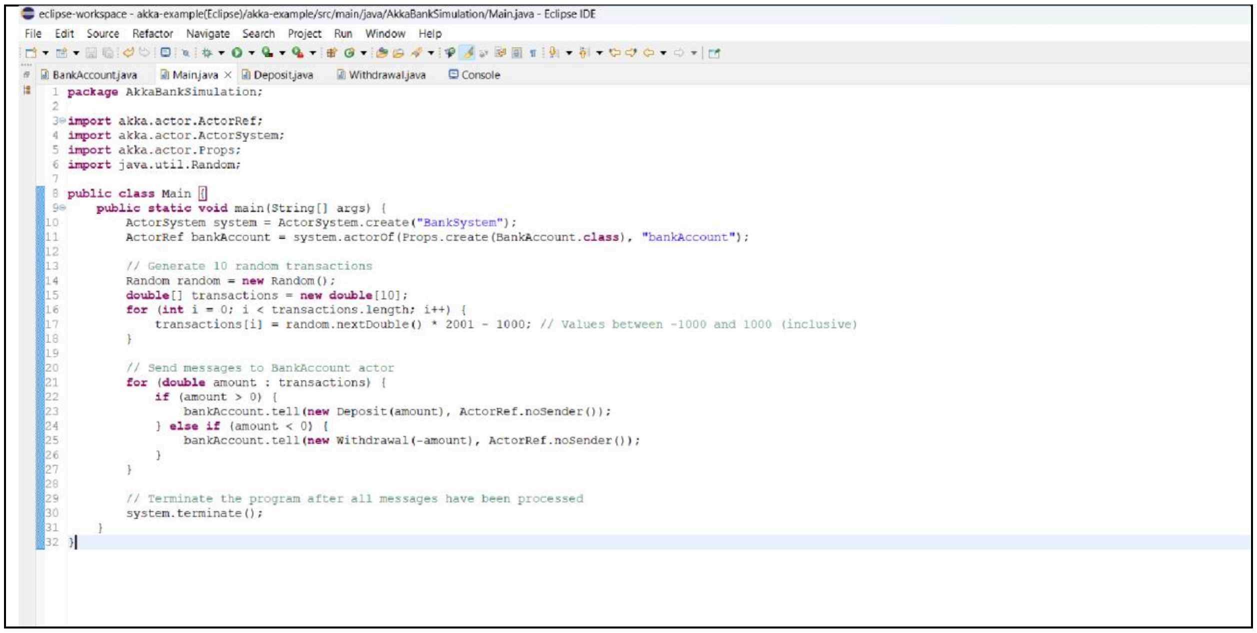
```java
package AkkaBankSimulation;

public class Withdrawal {
    public final double amount;

    public Withdrawal(double amount) {
        this.amount = amount;
    }
}
```

Description: This class signifies a message of withdrawal. Sending money to the bank account actor is what it's used for.

Attributes:

amount The amount that is to be withdrawn is represented by the term "amount."

Main.java:



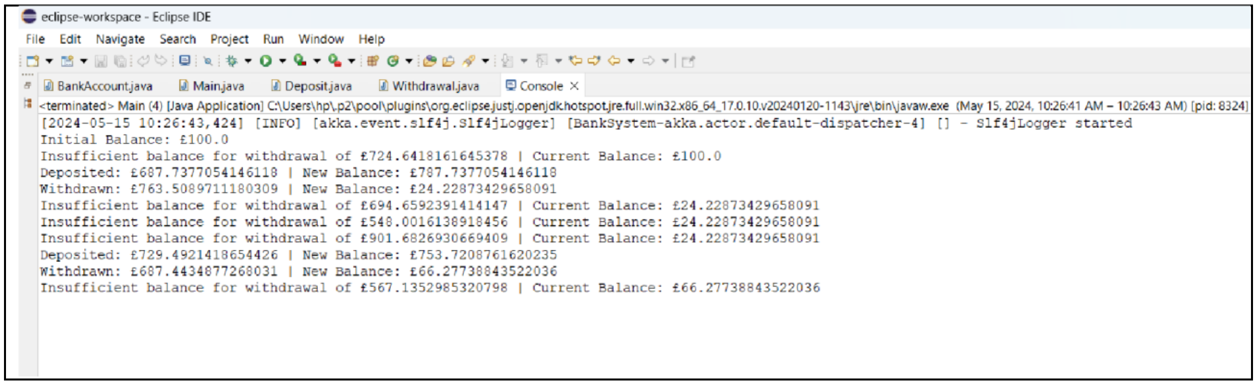
```java
package AkkaBankSimulation;

import akka.actor.ActorRef;
import akka.actor.ActorSystem;
import akka.actor.Props;
import java.util.Random;

public class Main {
    public static void main(String[] args) {
        ActorSystem system = ActorSystem.create("BankSystem");
        ActorRef bankAccount = system.actorOf(Props.create(BankAccount.class), "bankAccount");

        // Generate 10 random transactions
        Random random = new Random();
        double[] transactions = new double[10];
        for (int i = 0; i < transactions.length; i++) {
            transactions[i] = random.nextDouble() * 2001 - 1000; // Values between -1000 and 1000 (inclusive)
        }

        // Send messages to BankAccount actor
        for (double amount : transactions) {
            if (amount > 0) {
                bankAccount.tell(new Deposit(amount), ActorRef.noSender());
            } else if (amount < 0) {
                bankAccount.tell(new Withdrawal(-amount), ActorRef.noSender());
            }
        }

        // Terminate the program after all messages have been processed
        system.terminate();
    }
}
```

Description: This class contains the main method of simulation. It creates the actor system, creates the bank account actor, generates random transactions, and sends messages to the bank account actor to perform these transactions.

Methods:

main(String[] args): The entry point of the program. It initializes the actor system, creates the bank account actor, generates random transactions, sends messages to the bank account actor, and terminates the actor system after all messages have been processed.

Output:


eclipse-workspace - Eclipse IDE
File Edit Navigate Search Project Run Window Help
BankAccount.java
Main.java
Deposit.java
Withdrawal.java
Console

```
<terminated> Main (4) [Java Application] C:\Users\hp\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.10.v20240120-1143\jre\bin\javaw.exe (May 15, 2024, 10:26:41 AM – 10:26:43 AM) [pid: 8324]
[2024-05-15 10:26:43,424] [INFO] [akka.event.slf4j.Slf4jLogger] [BankSystem-akka.actor.default-dispatcher-4] [] - Slf4jLogger started
Initial Balance: £100.0
Insufficient balance for withdrawal of £724.6418161645378 | Current Balance: £100.0
Deposited: £687.7377054146118 | New Balance: £787.7377054146118
Withdrawn: £763.5089711180309 | New Balance: £24.22873429658091
Insufficient balance for withdrawal of £694.6592391414147 | Current Balance: £24.22873429658091
Insufficient balance for withdrawal of £548.0016138918456 | Current Balance: £24.22873429658091
Insufficient balance for withdrawal of £901.6826930669409 | Current Balance: £24.22873429658091
Deposited: £729.4921418654426 | New Balance: £753.7208761620235
Withdrawn: £687.4434877268031 | New Balance: £66.27738843522036
Insufficient balance for withdrawal of £567.1352985320798 | Current Balance: £66.27738843522036
```

Overall flow Description of the Akka Bank Account Simulation:

The program initializes a bank system with an actor representing a bank account, generates random transactions of deposits or withdrawals, sends these transactions as messages to the bank account actor, which processes them by updating its balance and printing relevant messages, then terminates the actor system after processing all messages, with output displayed on the console indicating transaction details and balance status.