

UNIVERSITY PARTNER



Distributed and Cloud Systems Programming (5CS022)

Assignment 1

<Concurrent Bank Account Simulation Using Akka Actor Framework>

Student Id: 2331187

Name: Sujan Prasad Pandey

Group: L5CG14

Module Leader: Mr. Deepson Shrestha

Submitted On: 2024-05-20

Contents

1. Introduction	1
2. Implementation Details.....	2
2.1. BankAccount Actor	2
2.2. Withdrawal Message.....	3
2.3. Deposit Message	4
2.4. Main Class	5
3. Code Execution and Output	6
4. Summary of How the Code Works	7
5. Explanation of Key Concepts	8
6. Conclusion	9

1. Introduction

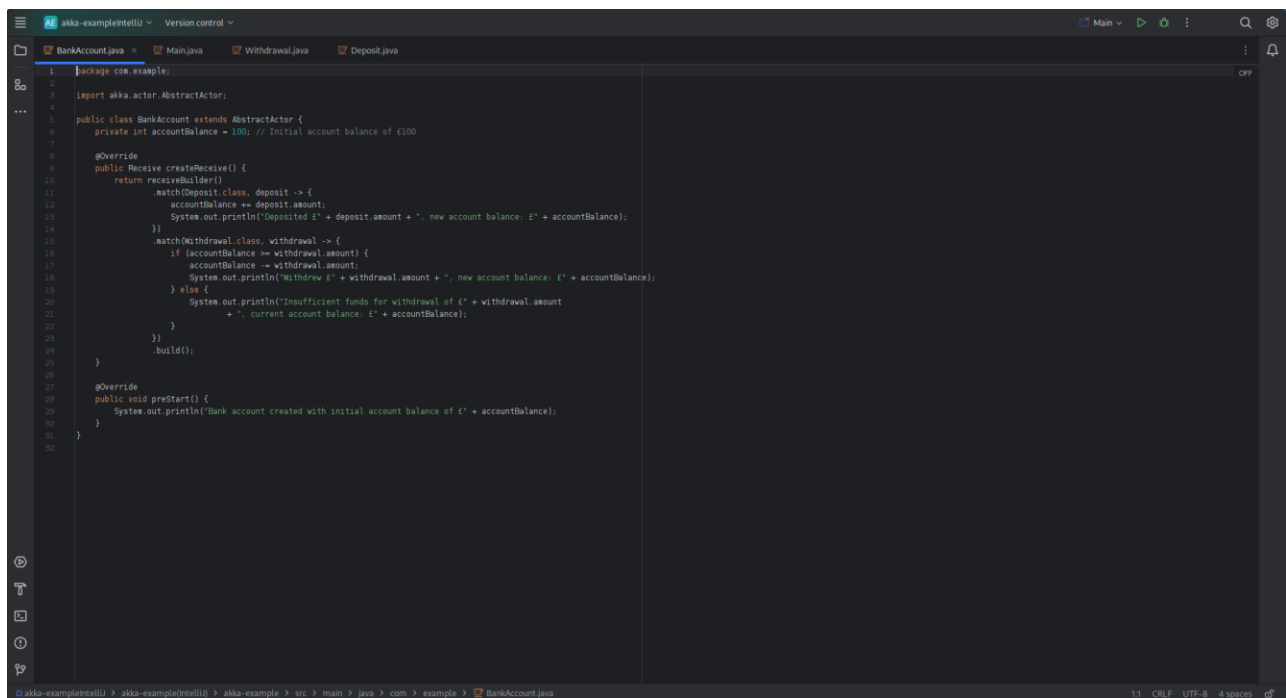
In the following, I applied the Akka Actor library to project the Bank Account, which serve multiple concurrent deposits and withdrawals at the same time. The aim of this simulation is to have it act as an illustration of how Akka does concurrency and state management. The step by step documentation provided here is a detailed guide explaining how to implement the tool. Along with the code snippets and screenshots included the documentation is meant to illustrate the process.

2. Implementation Details

2.1. BankAccount Actor

Our BankAccount class addresses this task. It is a class that manages your account balance. It handles credit/debit card transactions and deposits/withdrawals.

- `preStart()` Method: Starts the actor with balance £100 and announces this on the console.
- `createReceive()` Method: The Draw message is responsible for the activation of these messages.

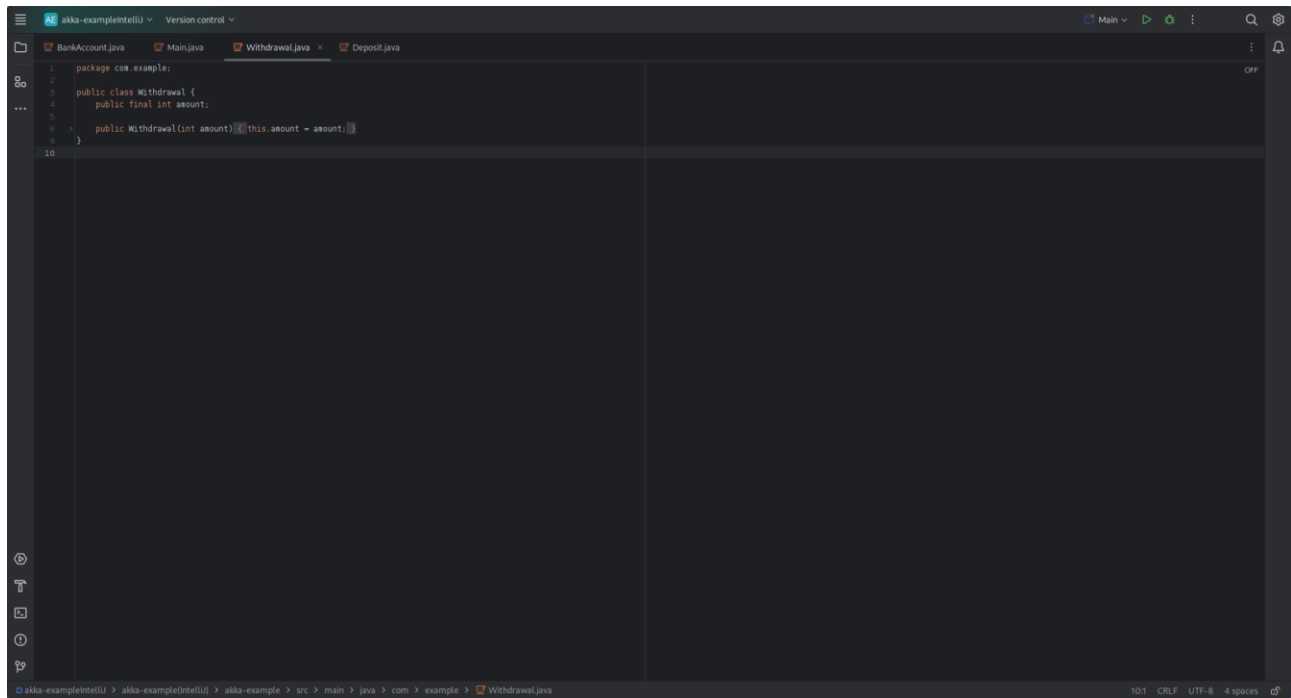


```
1 package com.example;
2
3 import akka.actor.AbstractActor;
4
5 public class BankAccount extends AbstractActor {
6     private int accountBalance = 100; // Initial account balance of £100
7
8     @Override
9     public Receive createReceive() {
10         return receiveBuilder()
11             .match(Deposit.class, deposit -> {
12                 accountBalance += deposit.amount;
13                 System.out.println("Deposited £" + deposit.amount + ", new account balance: £" + accountBalance);
14             })
15             .match(Withdrawal.class, withdrawal -> {
16                 if (accountBalance >= withdrawal.amount) {
17                     accountBalance -= withdrawal.amount;
18                     System.out.println("Withdrew £" + withdrawal.amount + ", new account balance: £" + accountBalance);
19                 } else {
20                     System.out.println("Insufficient funds for withdrawal of £" + withdrawal.amount
21                                     + ", current account balance: £" + accountBalance);
22                 }
23             })
24             .build();
25     }
26
27     @Override
28     public void preStart() {
29         System.out.println("Bank account created with initial account balance of £" + accountBalance);
30     }
31 }
32
```

2.2. Withdrawal Message

The class Withdrawal is a message to be sent to the BankAccount actor when a withdrawal has been made.

Constructor: Defines the withdrawal amount(s).

A screenshot of an IDE window titled 'akka-example IntelliJ'. The 'Withdrawal.java' file is open and shows the following code:

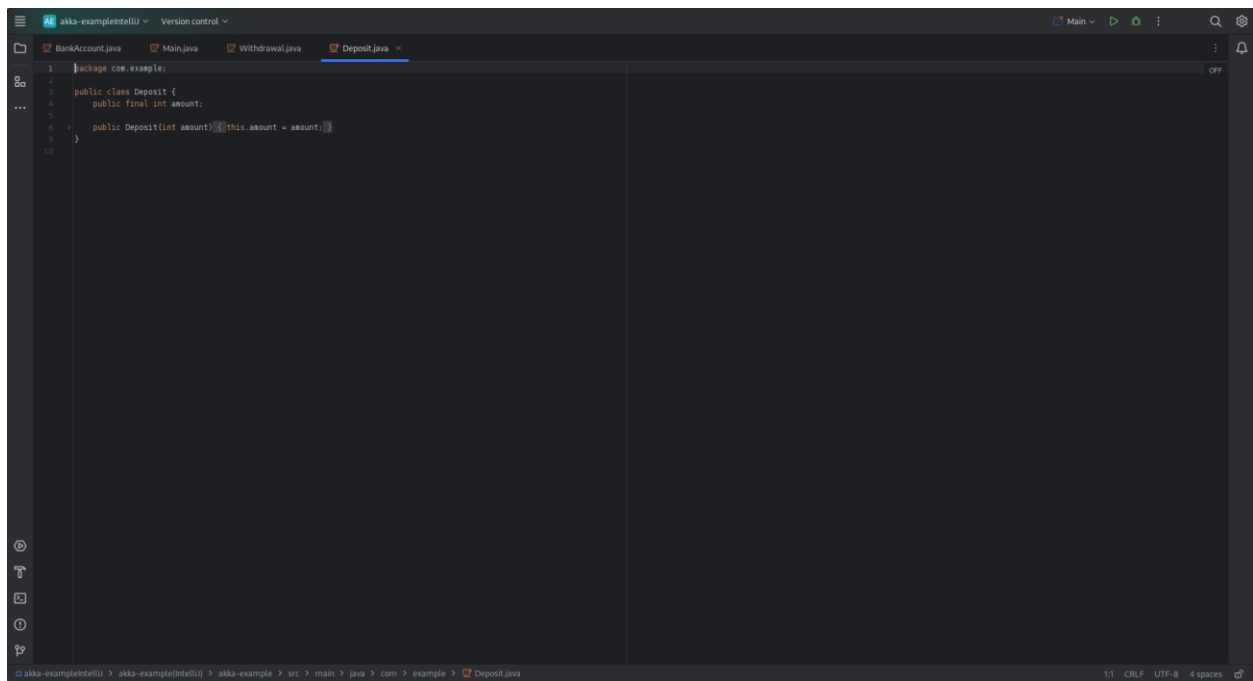
```
1 package com.example;
2
3 public class Withdrawal {
4     public final int amount;
5
6     public Withdrawal(int amount) { this.amount = amount; }
7 }
8
9
10
```

The IDE interface includes a file explorer on the left, a version control panel at the top, and a status bar at the bottom showing '10/1 CRLF UTF-8 4 spaces'.

2.3. Deposit Message

Deposit is the actor sent to the BankAccount to issue a deposit message.

Constructor: Calls the deposit method.(Initializes the deposit amount.)

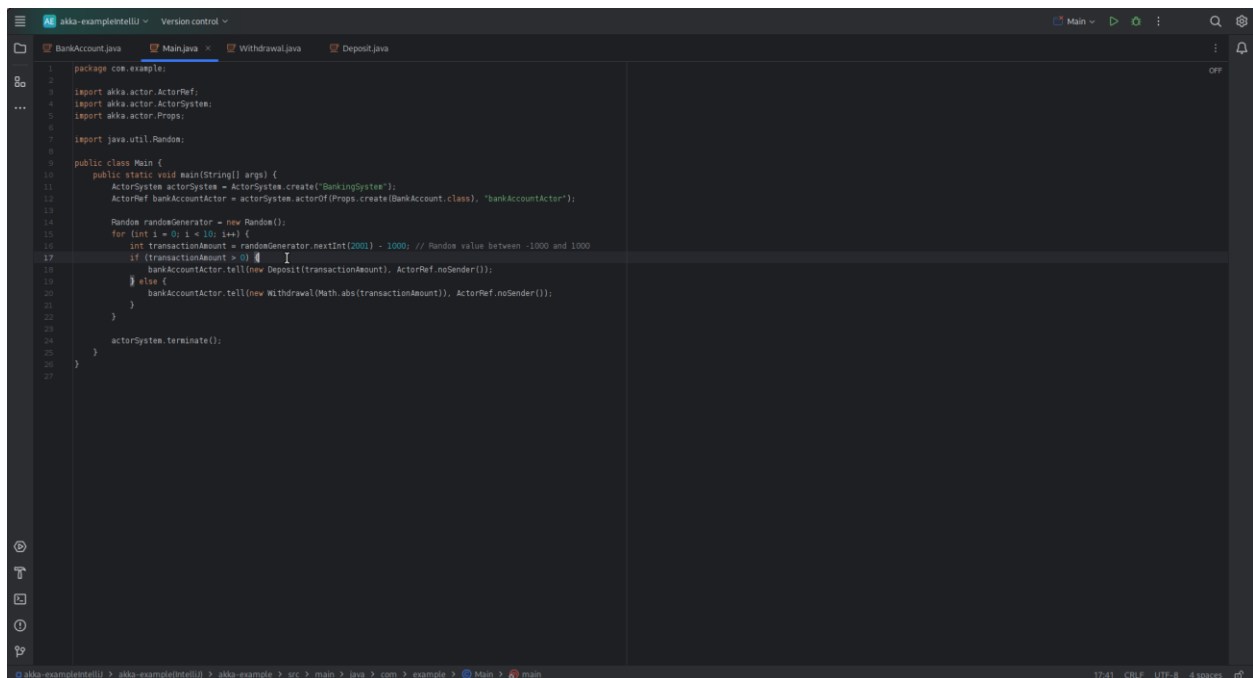


```
1 package com.example;
2
3 public class Deposit {
4     public final int amount;
5
6     public Deposit(int amount){this.amount = amount;}
7
8 }
9
10
```

2.4. Main Class

Main class is the one running the simulation, setting up BankAccount actor, and sending through myriads of transactions at random.

- ActorSystem Creation: Lauds the beginning of the Akka actor system. Listen to the given audio and then transcribe it.
- ActorRef Creation: Allocate an actor reference for use with the BankAccount actor.
- Random Transactions: Gives 10 randomly generated (deposits or withdrawals) transactions to the BankAccount.
- actorSystem. terminate(): Wrap with the execution of the actor system when all transactions are completed.



```
1 package com.example;
2
3 import akka.actor.ActorRef;
4 import akka.actor.ActorSystem;
5 import akka.actor.Props;
6
7 import java.util.Random;
8
9 public class Main {
10     public static void main(String[] args) {
11         ActorSystem actorSystem = ActorSystem.create("BankingSystem");
12         ActorRef bankAccountActor = actorSystem.actorOf(Props.create(BankAccount.class), "bankAccountActor");
13
14         Random randomGenerator = new Random();
15         for (int i = 0; i < 10; i++) {
16             int transactionAmount = randomGenerator.nextInt(2000) - 1000; // Random value between -1000 and 1000
17             if (transactionAmount > 0) {
18                 bankAccountActor.tell(new Deposit(transactionAmount), ActorRef.noSender());
19             } else {
20                 bankAccountActor.tell(new Withdrawal(Math.abs(transactionAmount)), ActorRef.noSender());
21             }
22         }
23         actorSystem.terminate();
24     }
25 }
26
27 }
```

3. Code Execution and Output

When the code is executed, the following output is generated:

```
[2024-05-19 08:34:01,674] [INFO] [akka.event.slf4j.Slf4jLogger] [BankingSystem-akka.actor.default-dispatcher-4] [] - Slf4jLogger started
Bank account created with initial account balance of £100
Insufficient funds for withdrawal of £888, current account balance: £100
Withdrew £10, new account balance: £90
Insufficient funds for withdrawal of £865, current account balance: £90
Insufficient funds for withdrawal of £228, current account balance: £90
Insufficient funds for withdrawal of £305, current account balance: £90
Insufficient funds for withdrawal of £275, current account balance: £90
Deposited £776, new account balance: £866
Deposited £88, new account balance: £954
Withdrew £338, new account balance: £616
Withdrew £373, new account balance: £243

Process finished with exit code 0
```


4. Summary of How the Code Works

This project undertakes development of a bank account that follows the Akka Architecture pattern to process deposits and withdrawals simultaneously. Here's a brief overview of how the code works: Here's a brief overview of how the code works:

- a) Initialization: The Main class creates an Actor system of Akka and act as a BankAccount actor with a starting balance of £100.
- b) Message Classes: Messages classes such as Deposit and Withdrawal define actions of (a) entering and (b) removal of money from account.
- c) Generating Transactions: The Main class generates the 10 transaction amounts, which are go - from -1000 to 1000. The positive figures are taken as refunds whilst the negative figures are considered as the refunds.
- d) Message Handling: Through a series of messages for each transaction, BankAccount actor is the beneficiary. The actor sends a balance update based on the message type, printing the new balance or preparing a message for the insufficient funds about the transaction exceeds the current balance.
- e) Termination: Once transactions are processed, executing actor system is available.

The gist is to establish a banking account that can safely process thousands of transactions concurrently, prompting authoritative updates and balances reports when it completes each transaction. It is going to be giving benefit of custom software development.

5. Explanation of Key Concepts

- **Actor Model:** Actor model is a computational model for working with concurrency. It bases clarity on the actors of the process as the primitive units of computation. This can be performed by any actor with the reception, processing, and sending of messages.
- **Akka Actor Framework:** Akka is a set of libraries and runtime engine for creating massively parallel, distributed and fault tolerant message passing application on the Java Virtual Machine. The model enables the process of doing the making of concurrent programs to be simplified as actors are used.
- **Concurrency:** The network simultaneity of running different software applications together or several parts of a program at the same time. Here, Akka addresses the concurrency aspect to ensure that bank accounts can be squared away as well as monies can be deposited smoothly without conflicts.
- **Actor Lifecycle:** The Akka actors can be created and shut down; within them messages are given to handlers. The `preStart` method is defined to be called during actor's start up operations.

6. Conclusion

This Documentation presents a complete summary of a bank account simulation which is based on the Akka Actor framework. The project is an example of managing state and concurrency using actors, processing non-atomic deposit or withdrawal operation, as well handling all possible scenarios like insufficient balance, etc. The code snippets and the explanations should well show the process of creating a real-life application with the framework.