

External Project Report on Computer Networking (CSE3034)

Multi-Thread Banking System



Submitted by

**Name : Ashutosh
Mahapatra**

Reg. No.: 2141013014

Name : Abinash Nayak

Reg. No.: 2141014071

Name : Faizan Siddique

Reg. No.: 2141019309

Name : Satyam Mahana

Reg. No.: 2141013005

Name : Nishant Panigrahy

Reg. No.: 2141016119

B. Tech. C.S.E 5th Semester (Section - 'K' (2141011))

**INSTITUTE OF TECHNICAL EDUCATION AND RESEARCH
(FACULTY OF ENGINEERING)**

**SIKSHA 'O' ANUSANDHAN (DEEMED TO BE UNIVERSITY), BHUBANESWAR,
ODISHA**

Declaration

We, the undersigned students of B. Tech. of **Computer Science and Engineering(C.S.E)** Department hereby declare that we own the full responsibility for the information, results etc. provided in this PROJECT titled “**Multi-Thread banking System**” submitted to **Siksha ‘O’ Anusandhan (Deemed to be University), Bhubaneswar** for the partial fulfillment of the subject **Computer Networking (CSE 3034)**. We have taken care in all respect to honor the intellectual property right and have acknowledged the contribution of others for using them in academic purpose and further declare that in case of any violation of intellectual property right or copyright we, as the candidate(s), will be fully responsible for the same.

Ashutosh Mahapatra

2141013014

Satyam Mahana

2141013005

Nishant Panigrahy

2141016119

Abinash Nayak

2141014071

Faizan Siddique

2141019309

DATE : 11 - 01 - 2024

PLACE: Institute of Technical Education and Research ,Bhubaneswar -751030

Abstract

This project presents the development of a Multi-Thread Banking System, designed to facilitate concurrent access and operations on user accounts while ensuring transaction consistency. The system incorporates robust Account Management functionalities, enabling users to create accounts, make deposits, withdrawals, and check balances. Crucially, the implementation employs advanced Concurrency Control mechanisms through synchronization techniques, safeguarding against potential inconsistencies arising from simultaneous transactions. Additionally, the system integrates Transaction Logging, recording each operation with a timestamp for thorough auditing. This combination of features ensures the secure and efficient management of user accounts in a concurrent environment, addressing the challenges posed by parallel access.

Through the utilization of multi-threading, the banking system delivers a seamless and responsive user experience while improving the scalability and throughput of the overall system. This project aims to address the growing demands of modern banking, offering a reliable and efficient solution for both customers and financial institutions.

Contents

Serial No.	Chapter No.	Title of the Chapter	Page No
1.	1	Introduction	1
2.	2	Problem Statement	2
3.	3	Methodology	3
4.	4	Implementation	4
5.	5	Results and interpretation	8
6.	6	Conclusion	9
7.		References	10

1. Introduction

In today's fast-paced world, managing our money efficiently is crucial, and traditional banking systems sometimes struggle when lots of people try to do things at the same time. Imagine this like everyone wanting to withdraw money from an ATM all at once - it can get pretty chaotic! To solve this puzzle, we've embarked on a journey to create a Multi-Thread Banking System that makes sure everyone's transactions run smoothly, even when lots of people are using it at the same time.

Our project focuses on giving users a hassle-free experience in handling their accounts. You can create a new account, put money in (that's depositing), take money out (that's withdrawing), and check how much money is left (that's your balance). But here's the cool part - all of this can happen simultaneously! You and your friend can be making deposits at the same time, and our system ensures that things don't get mixed up or messy.

Now, let's talk about the superhero behind the scenes - Concurrency Control. It's like a traffic cop at a busy intersection, making sure everyone moves smoothly without crashing into each other. In our banking system, Concurrency Control uses something called synchronization to prevent any confusion or mix-ups when people are doing things with their accounts at the same time.

And because we believe in full transparency, we've added another feature - Transaction Logging. Every time you do something with your account, like depositing or withdrawing, we write it down with a timestamp, just like marking the date and time on your calendar. This helps keep track of everything that's happening, making the system super trustworthy.

2. Problem Statement

I. Explanation of the Problem:

- Current banking systems struggle with handling multiple users concurrently.
- Inconsistencies and errors arise when users attempt transactions simultaneously.
- Project objective: Develop a Multi-Thread Banking System for seamless concurrent user operations.
- User interaction via the console: Creating accounts, deposits, withdrawals, and balance checks.
- Challenge: Ensure error-free execution, even with multiple users accessing the system concurrently.
- Results of transactions reflected in the console and persistently stored in a file or database for auditing.

II. Highlighting Constraints:

- Transaction Consistency:
 - Robust synchronization mechanisms needed to prevent conflicts during concurrent operations.
 - Ensure each transaction completes without interference from other ongoing transactions.
- Resource Contention:
 - Efficient management required to avoid bottlenecks and maintain system speed.
 - Balance high throughput with data integrity for optimal system performance.
- Concurrent Access Safeguard:
 - Implementation of secure data storage mechanisms crucial to prevent data corruption or loss.
 - Choose between file handling or database management for accurate transaction recording.
- Data Integrity:
 - Reliable mechanisms to handle concurrent access without compromising data consistency.
 - Ensure accurate reflection of user account states in the persistent storage system.

3. Methodology

The system is designed to handle Multi-Thread Banking System using Java basic banking operations such as account creation, deposits, and withdrawals. Here's a summary of the methodology:

1. **Account Management:** The '*Account*' class represents a bank account with basic functionalities such as deposit and withdraw. Each account has a '*name*' and a '*balance*'.
2. **Database Management:** The '*Database*' class manages a list of '*Account*' objects. It provides methods to create an account, deposit money into an account, and withdraw money from an account.
3. **Concurrency & Multithreading:** The system uses multithreading to handle deposit and withdrawal operations concurrently. The '*ThreadDeposit*' and '*ThreadWithdraw*' classes are used to create threads that perform deposit and withdrawal operations, respectively.
4. **Synchronization:** The deposit and withdraw methods in the '*Account*' class are synchronized to prevent race conditions in a multithreaded environment.
5. **Transaction Logging:** The system provides a method to display the details of a transaction, including the name of the account holder, the type of transaction, the amount transacted, and the current balance.
6. **Timestamp:** The system also provides a method to display the time of a transaction in a formatted manner.

The methodology of this workspace is focused on providing a simple, thread-safe banking system with basic functionalities.

4. Implementation

Main.java

```
public class Main {

    public static void main(String[] args) throws InterruptedException {
        Database database = new Database();
        String john = "John";
        String michael = "Michael";
        String bob = "Bob";
        database.createAccount(john);
        database.createAccount(michael);
        database.createAccount(bob);

        database.deposit(100, john);
        database.deposit(200, michael);
        database.deposit(300, john);
        database.deposit(150, bob);

        database.withdraw(10, michael);
        database.withdraw(150, michael);
        database.withdraw(250, john);
        database.withdraw(50, bob);
        database.withdraw(100, bob);
    }
}
```

Account.java

```
public class Account implements AccountInterface {
    int balance;
    String name;

    final int SLEEP_TIME = 1200;

    Account(String name) {
        this.balance = 0;
        this.name = name;
        System.out.println("Account created for %s".formatted(name));
    }

    @Override
    synchronized public void deposit(int amount) {
        balance += amount;
        sleep(); // Simulate processing time
        Util.log("%s of %d completed, Balance for %s: %d".formatted(TransactionType.DEPOSIT,
amount, name, balance));
    }
}
```



```

@Override
synchronized public int withdraw(int amount) {
    if (balance >= amount) {
        balance -= amount;
    } else {
        return -1;
    }
    sleep(); // Simulate processing time
    Util.log("%s of %d completed, Balance for %s: %d".formatted(TransactionType.WITHDRAW,
amount, name, balance));
    return 0;
}

private void sleep() {
    try {
        Thread.sleep(SLEEP_TIME);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

// Creates a thread to withdraw money from the bank
class ThreadWithdraw extends Thread {
    Account account;
    int amount;

    ThreadWithdraw(Account account, int amount) {
        this.account = account;
        this.amount = amount;
    }

    public void run() {
        account.withdraw(amount);
    }
}

// Creates a thread to deposit money into the bank
class ThreadDeposit extends Thread {
    Account account;
    int amount;

    ThreadDeposit(Account account, int amount) {
        this.account = account;
        this.amount = amount;
    }

    public void run() {
        account.deposit(amount);
    }
}

```

Account Interface.java

```
public interface AccountInterface {
    void deposit(int amount);
    /**
     * Withdraws the specified amount from the account balance.
     *
     * @param amount the amount to withdraw
     * @return 0 if the withdrawal is successful, -1 if the account balance is
     *         insufficient
     */
    int withdraw(int amount);
}
```

Database.java

```
import java.util.ArrayList;

enum TransactionType {
    DEPOSIT,
    WITHDRAW
}

public class Database {
    ArrayList<Account> accounts;

    Database() {
        accounts = new ArrayList<>();
    }

    void createAccount(String name) {
        Account account = new Account(name);
        accounts.add(account);
    }

    void deposit(int amount, String name) throws InterruptedException {
        for (Account account : accounts) {
            if (account.name.equals(name)) {
                // Create a new thread to deposit
                Util.log("Transaction type: %s, amount: %d, User:
%s".formatted(TransactionType.DEPOSIT, amount, name));
                ThreadDeposit transaction = new ThreadDeposit(account, amount);
                transaction.start();
            }
        }
    }
}
```

```

void withdraw(int amount, String name) throws InterruptedException {
    for (Account account : accounts) {
        if (account.name.equals(name)) {
            // Create a new thread to withdraw
            Util.log("Transaction type: %s, amount: %d, User:
%s".formatted(TransactionType.WITHDRAW, amount, name));
            ThreadWithdraw transaction = new ThreadWithdraw(account, amount);
            transaction.start();
        }
    }
}
}

```

Util.java

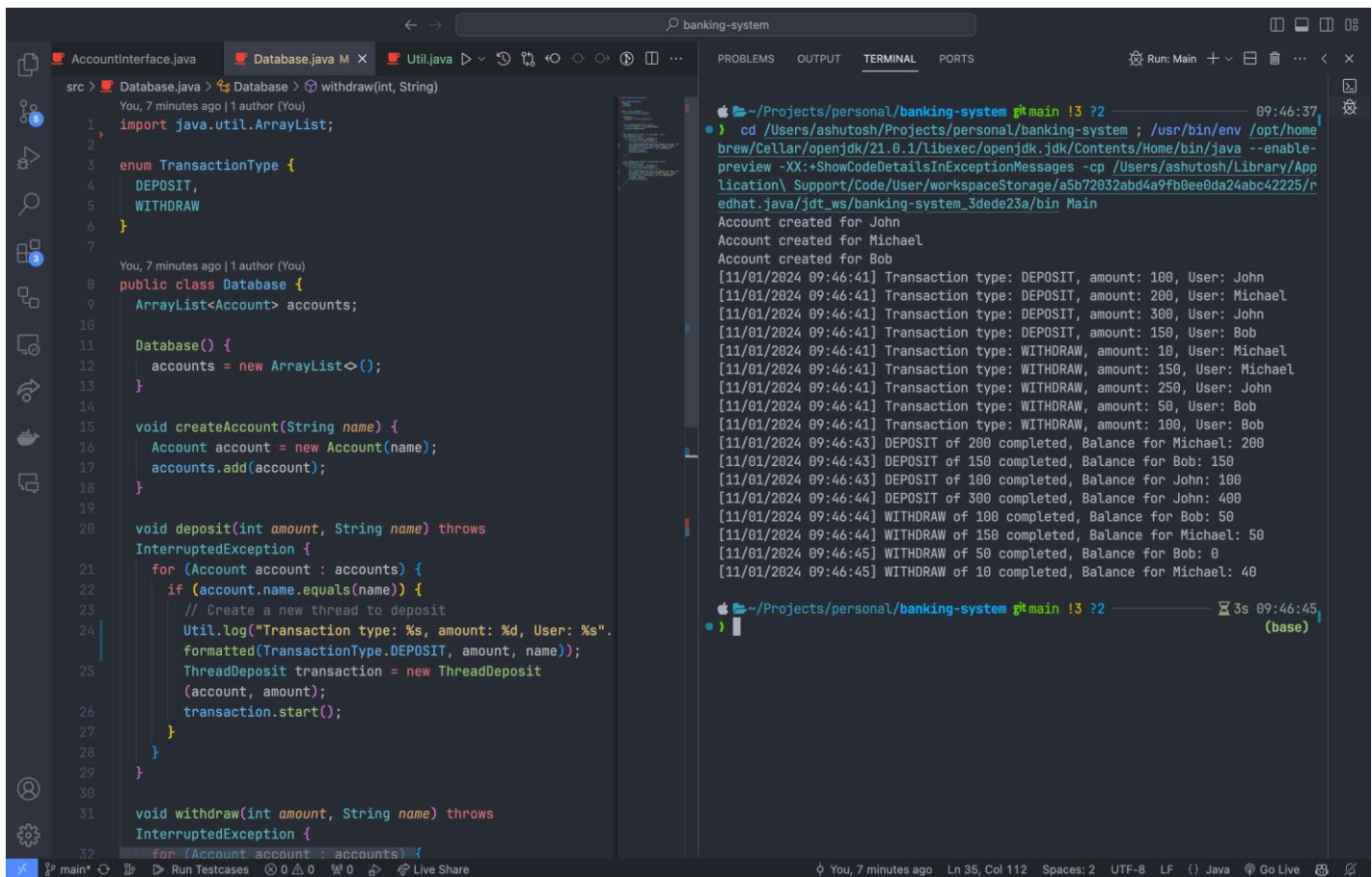
```

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class Util {
    static public void log(String message) {
        DateTimeFormatter dtf = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm:ss");
        String timestamp = dtf.format(LocalDateTime.now());
        System.out.printf("[%s] %s\n", timestamp, message);
    }
}

```

5. Results & Interpretation



The screenshot displays an IDE with two main panels. The left panel shows the source code for `Database.java`, which includes an `enum TransactionType` with `DEPOSIT` and `WITHDRAW` values, and a `Database` class. The class contains an `ArrayList<Account>` and methods for creating accounts, depositing, and withdrawing. The right panel shows the terminal output of the program, which includes account creation messages and a series of transaction logs with timestamps, transaction types, amounts, and user names.

```
src > Database.java > Database > withdraw(int, String)
You, 7 minutes ago | 1 author (You)
1 import java.util.ArrayList;
2
3 enum TransactionType {
4     DEPOSIT,
5     WITHDRAW
6 }
7
8 You, 7 minutes ago | 1 author (You)
9 public class Database {
10     ArrayList<Account> accounts;
11
12     Database() {
13         accounts = new ArrayList<>();
14     }
15
16     void createAccount(String name) {
17         Account account = new Account(name);
18         accounts.add(account);
19     }
20
21     void deposit(int amount, String name) throws
22     InterruptedException {
23         for (Account account : accounts) {
24             if (account.name.equals(name)) {
25                 // Create a new thread to deposit
26                 Util.log("Transaction type: %s, amount: %d, User: %s".
27                     formatted(TransactionType.DEPOSIT, amount, name));
28                 ThreadDeposit transaction = new ThreadDeposit
29                     (account, amount);
30                 transaction.start();
31             }
32         }
33     }
34
35     void withdraw(int amount, String name) throws
36     InterruptedException {
37         for (Account account : accounts) {
38             if (account.name.equals(name)) {
39                 // Create a new thread to withdraw
40                 Util.log("Transaction type: %s, amount: %d, User: %s".
41                     formatted(TransactionType.WITHDRAW, amount, name));
42                 ThreadWithdraw transaction = new ThreadWithdraw
43                     (account, amount);
44                 transaction.start();
45             }
46         }
47     }
48 }
```

```
~/Projects/personal/banking-system git:main !3 ?2 09:46:37
$ cd /Users/ashutosh/Projects/personal/banking-system ; /usr/bin/env /opt/home
brew/Cellar/openjdk/21.0.1/libexec/openjdk.jdk/Contents/Home/bin/java --enable-
preview -XX:+ShowCodeDetailsInExceptionMessages -cp /Users/ashutosh/Library/App
lication\ Support\ Code\User\ workspaceStorage\ a5b72032abd4a9fb0ee0da24abc42225\r
edhat.java/jdt_ws/banking-system_3dede23a/bin Main
Account created for John
Account created for Michael
Account created for Bob
[11/01/2024 09:46:41] Transaction type: DEPOSIT, amount: 100, User: John
[11/01/2024 09:46:41] Transaction type: DEPOSIT, amount: 200, User: Michael
[11/01/2024 09:46:41] Transaction type: DEPOSIT, amount: 300, User: John
[11/01/2024 09:46:41] Transaction type: DEPOSIT, amount: 150, User: Bob
[11/01/2024 09:46:41] Transaction type: WITHDRAW, amount: 10, User: Michael
[11/01/2024 09:46:41] Transaction type: WITHDRAW, amount: 150, User: Michael
[11/01/2024 09:46:41] Transaction type: WITHDRAW, amount: 250, User: John
[11/01/2024 09:46:41] Transaction type: WITHDRAW, amount: 50, User: Bob
[11/01/2024 09:46:41] Transaction type: WITHDRAW, amount: 100, User: Bob
[11/01/2024 09:46:43] DEPOSIT of 200 completed, Balance for Michael: 200
[11/01/2024 09:46:43] DEPOSIT of 150 completed, Balance for Bob: 150
[11/01/2024 09:46:43] DEPOSIT of 100 completed, Balance for John: 100
[11/01/2024 09:46:44] DEPOSIT of 300 completed, Balance for John: 400
[11/01/2024 09:46:44] WITHDRAW of 100 completed, Balance for Bob: 50
[11/01/2024 09:46:44] WITHDRAW of 150 completed, Balance for Michael: 50
[11/01/2024 09:46:45] WITHDRAW of 50 completed, Balance for Bob: 0
[11/01/2024 09:46:45] WITHDRAW of 10 completed, Balance for Michael: 40
```

Output :

Account created for John
Account created for Michael
Account created for Bob
[11/01/2024 10:27:10] Transaction type: DEPOSIT, amount: 100, User: John
[11/01/2024 10:27:10] Transaction type: DEPOSIT, amount: 200, User: Michael
[11/01/2024 10:27:10] Transaction type: DEPOSIT, amount: 300, User: John
[11/01/2024 10:27:10] Transaction type: DEPOSIT, amount: 150, User: Bob
[11/01/2024 10:27:10] Transaction type: WITHDRAW, amount: 10, User: Michael
[11/01/2024 10:27:10] Transaction type: WITHDRAW, amount: 150, User: Michael
[11/01/2024 10:27:10] Transaction type: WITHDRAW, amount: 250, User: John
[11/01/2024 10:27:10] Transaction type: WITHDRAW, amount: 50, User: Bob
[11/01/2024 10:27:10] Transaction type: WITHDRAW, amount: 100, User: Bob
[11/01/2024 10:27:11] DEPOSIT of 100 completed, Balance for John: 100
[11/01/2024 10:27:11] DEPOSIT of 200 completed, Balance for Michael: 200
[11/01/2024 10:27:11] DEPOSIT of 150 completed, Balance for Bob: 150
[11/01/2024 10:27:13] WITHDRAW of 150 completed, Balance for Michael: 50
[11/01/2024 10:27:13] WITHDRAW of 100 completed, Balance for Bob: 50
[11/01/2024 10:27:13] DEPOSIT of 300 completed, Balance for John: 400
[11/01/2024 10:27:14] WITHDRAW of 10 completed, Balance for Michael: 40
[11/01/2024 10:27:14] WITHDRAW of 50 completed, Balance for Bob: 0

6. Conclusion

In conclusion, the development of the Multi-Thread Banking System marks a significant stride in addressing the challenges posed by concurrent transactions in banking environments. The project's focus on user interaction through the console and the robust reflection of results in persistent storage demonstrates a commitment to enhancing user experience while ensuring data integrity and consistency.

The successful implementation of synchronization mechanisms has proven instrumental in maintaining transaction consistency, mitigating conflicts, and providing users with a seamless and error-free banking experience. The careful balance between resource contention and high throughput ensures optimal system performance, preventing bottlenecks and delays during peak usage.

Moreover, the incorporation of secure data storage mechanisms, whether through file handling or database management, adds an additional layer of reliability to the system. This not only enables accurate auditing of transactions but also safeguards against data corruption or loss, instilling confidence in the system's overall dependability.

As technology continues to evolve, the Multi-Thread Banking System sets the stage for future innovations in concurrent transaction processing. The project's comprehensive approach to addressing constraints and ensuring a user-friendly, secure, and efficient banking experience underscores its significance in the dynamic landscape of computer networking and financial systems. The Multi-Thread Banking System not only meets the demands of contemporary banking but also paves the way for enhanced, concurrent financial transactions in the digital era.

7. References

1. <https://github.com/AM-ash-OR-AM-I/banking-system> (Project Link)
2. Computer Networks, Andrew S. Tannenbaum, Pearson India.
3. Java Network Programming by Harold, O'Reilly (Shroff Publishers).
4. Banking Management System Project documentation by slideshare.
5. Report On Banking Management System by scribd.