



CSE23CL105 - Object Oriented Programming With Java Lab

ATM SYSTEM WITH MULTI-THREADING

PRAVIN M (E0224029)

B.TECH CYS & IOT – 1ST YEAR (BATCH –IV)

KALE GAGAN (E0224027)

B.TECH CYS & IOT – 1ST YEAR (BATCH –IV)

T SUBASELVAN(E0324063)

B.TECH AIDA– 1ST YEAR (BATCH –IV)

INTRODUCTION

This project showcases a Concurrent ATM System simulation developed in Java, modeling simultaneous user withdrawal requests on a single shared bank account. It demonstrates core concurrency concepts including multithreading, thread synchronization to ensure data integrity, exception handling, and transaction logging.

Brief Overview

System developed in Java language.

Purpose

Model thread synchronization
for safe concurrent withdrawals
from a shared account.

Applications

Banks , ATM,
Financial Institutions

PROJECT OBJECTIVES

- This project aims to create a functional simulation of a concurrent ATM system using Java.
- The primary focus is demonstrating safe management of simultaneous withdrawal requests targeting a single shared bank account, achieved through thread synchronization.
- The system also handles interactive setup for simulation parameters, manages errors like insufficient funds via exception handling, provides console logging to observe concurrent operations, and includes transaction recording with optional statement file generation, illustrating key Java concurrency techniques.

 **Concurrency**

 **Synchronization**

 **Simulation**

 **Exception
Handling**

TECHNIQUES USED

- **Object-Oriented Programming (OOP):** Using classes (BankAccount, WithdrawalTask, etc.) and objects to structure the code.
- **Multithreading:** Employing Thread and Runnable to simulate concurrent users.
- **Synchronization:** Using the synchronized keyword for controlling access to shared resources (BankAccount.withdraw).
- **Exception Handling:** Using try-catch-finally blocks, custom exceptions (InsufficientFundsException), and handling standard exceptions (InterruptedException, InputMismatchException, IOException).
- **Java Collections Framework:** Utilizing List, ArrayList, and Collections.synchronizedList for managing threads and transaction logs.
- **Console Input/Output (I/O):** Using java.util.Scanner for input and System.out/System.err for output (via logging helpers).
- **File Input/Output (I/O):** Using java.io.FileWriter and java.io.PrintWriter for creating the statement file.
- **Date & Time API (java.time):** Using LocalDateTime and DateTimeFormatter for timestamps.

RESULTS

```
===== SETUP STARTING =====
```

```
Interactive ATM Simulation Setup
```

```
Current Time: 2025-04-16 22:50:49
```

```
-----  
Enter the initial balance for the account: $2500
```

```
Account created with balance: $2500.00
```

```
===== ACCOUNT CREATED =====
```

```
How many users want to withdraw simultaneously? 3
```

```
-----  
Enter details for each user:
```

```
--- User #1 Input ---
```

```
Enter name for User #1: Pravin
```

```
Enter amount for Pravin to withdraw: $650
```

```
--- User #2 Input ---
```

```
Enter name for User #2: gagan
```

```
Enter amount for gagan to withdraw: $961
```

```
--- User #3 Input ---
```

```
Enter name for User #3: suba
```

```
Enter amount for suba to withdraw: $1250
```

```
===== SETUP COMPLETE =====
```

- This output shows the interactive setup phase of the ATM simulation where the program collects initial parameters from the user via the console.
- Specifically, it records a starting balance of \$2500 and gathers the details for three concurrent withdrawal attempts by users Pravin (\$650), gagan (\$961), and suba (\$1250)

```
===== SIMULATION STARTING =====
```

```
[22:51:11.802] [MainThread] Starting 3 concurrent withdrawal threads...
[22:51:11.810] [MainThread] Waiting for worker threads to complete...
[22:51:11.812] [suba] >>> Task starting. Attempting Withdraw $1250.00...
[22:51:11.815] [Pravin] >>> Task starting. Attempting Withdraw $650.00...
[22:51:11.815] [gagan] >>> Task starting. Attempting Withdraw $961.00...
[22:51:11.982] [suba] --- Transaction Summary ---
[22:51:11.983] [suba] Action: WITHDRAW $1250.00
[22:51:11.984] [suba] Status: SUCCESS
[22:51:11.985] [suba] Balance Before: $2500.00
[22:51:11.985] [suba] Balance After: $1250.00
[22:51:11.986] [suba] <<< Task finished.
[22:51:12.132] [gagan] --- Transaction Summary ---
[22:51:12.135] [Pravin] --- Transaction Summary ---
[22:51:12.136] [gagan] Action: WITHDRAW $961.00
[22:51:12.137] [Pravin] Action: WITHDRAW $650.00
[22:51:12.138] [Pravin] Status: FAILED (Insufficient Funds)
[22:51:12.137] [gagan] Status: SUCCESS
[22:51:12.138] [Pravin] Balance Before: $289.00
[22:51:12.139] [gagan] Balance Before: $1250.00
[22:51:12.140] [gagan] Balance After: $289.00
[22:51:12.140] [Pravin] Balance After: $289.00
[22:51:12.141] [gagan] <<< Task finished.
[22:51:12.143] [Pravin] <<< Task finished.
[22:51:12.145] [MainThread] All worker threads have completed their tasks.
```

```
===== SIMULATION FINISHED =====
```

```
===== FINAL RESULTS =====
```

```
[22:51:12.146] [MainThread] Final balance in the account: $289.00
[22:51:12.147] [MainThread] Total transactions recorded: 3
```

```
=====
```

```
Generate transaction statement file? (yes/no): yes
```

```
Generating statement file (multi-line format): ATM_Statement_20250416_225131.txt ...
Statement file generated successfully.
```

- This output shows the execution phase of the ATM simulation **where three concurrent threads (Suba, Pravin, gagan) attempt their withdrawals**, displaying the interleaved results using the summary block format.
- It demonstrates successful withdrawals by suba and gagan, Pravin's failure due to **insufficient funds**, the final account balance (\$289), and the subsequent **generation of the transaction statement file** as requested by the user.

```
===== SETUP STARTING =====
```

```
Interactive ATM Simulation Setup
```

```
Current Time: 2025-04-18 23:20:56
```

```
Enter the initial balance for the account: $3000
```

```
Account created with balance: $3000.00
```

```
===== ACCOUNT CREATED =====
```

```
How many users want to withdraw simultaneously? 3
```

```
Enter details for each user:
```

```
--- User #1 Input ---
```

```
Enter name for User #1: 251
```

```
Enter amount for 251 to withdraw: $five
```

```
Invalid input. Please enter a number.
```

```
Enter amount for 251 to withdraw: $012
```

```
--- User #2 Input ---
```

```
Enter name for User #2: curan
```

```
Enter amount for curan to withdraw: $twelve
```

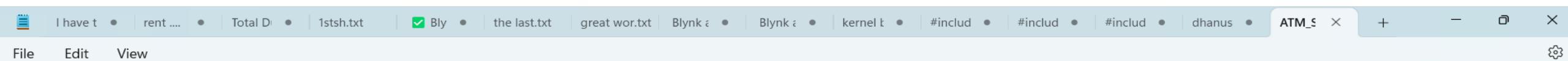
```
Invalid input. Please enter a number.
```

```
Enter amount for curan to withdraw: $
```

Exception Handling

- This screenshot shows the ATM simulation's interactive setup phase where input validation is actively working. After accepting an initial balance of \$3000 and the number of users (3)
- The program correctly identifies and **rejects non-numeric withdrawal amounts like "five" and "twelve"**. It prompts the user again after each invalid entry, waiting for a valid number before proceeding to the next step.

Generated Statement File Using Java File Handling



ATM Transaction Statement

Generated on: 2025-04-16 22:51:31

Initial Balance Recorded: \$2500.00

Timestamp: 2025-04-16 22:51:11.827

User: suba

Action: WITHDRAW_SUCCESS

Amount: \$1250.00

Balance: \$2500.00 -> \$1250.00

Status: Transaction successful.

Timestamp: 2025-04-16 22:51:11.982

User: gagan

Action: WITHDRAW_SUCCESS

Amount: \$961.00

Balance: \$1250.00 -> \$289.00

Status: Transaction successful.

Timestamp: 2025-04-16 22:51:12.132

User: Pravin

Action: WITHDRAW_FAILED_FUNDS

REFERENCES

1. Head First Java by Kathy Sierra and Bert Bates
2. Java: A Beginner's Guide by Herbert Schildt
3. <https://www.geeksforgeeks.org/synchronization-in-java/>
4. <https://www.geeksforgeeks.org/synchronization-in-java/>
5. https://www.w3schools.com/java/java_files.asp
6. <https://www.geeksforgeeks.org/exceptions-in-java/>

*Thank
you!*