| МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ  КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ імені Тараса Шевченка  ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ  **Кафедра програмних систем і технологій**                Дисципліна  **« Кросплатформне програмування,»**      **Лабораторна робота № 6**  **на тему:**  **"Симуляція роботи банкомату"** | | | |
| --- | --- | --- | --- |
| **Виконав:** | Безруков Андрій Миколайович | **Перевірив**: | Васильєв Олексій Миколайович |
| Група | ІПЗ-33 | Дата перевірки |  |
| Форма навчання | денна | Оцінка |  |
| Спеціальність | 121 |
| 2024 | | | |

## **1. Постановка задачі**

## Створити Java‑консольний застосунок, який моделює роботу банкомату з підтримкою одночасного обслуговування кількох користувачів. Основні вимоги:

## Кожен користувач має унікальний рахунок з балансом.

## Одноразове зняття коштів обмежене сумою **1000**.

## Підтримка операцій:

## відкриття рахунку;

## закриття рахунку;

## зняття коштів;

## поповнення рахунку;

## Логування всіх операцій у файл atm\_log.txt.

## 

## **2. Опис реалізації та програмний код**

## Застосунок реалізовано з використанням:

## ConcurrentHashMap для безпечного зберігання рахунків при паралельному доступі.

## ReentrantLock для синхронізації операцій з балансом та логування.

## Клас Logger для запису операцій у лог-файл із мітками часу.

### **Розділ "Програмний код"**

## *package lab6;*

## 

## *import java.io.FileWriter;*

## *import java.io.IOException;*

## *import java.io.PrintWriter;*

## *import java.text.SimpleDateFormat;*

## *import java.util.Date;*

## *import java.util.Scanner;*

## *import java.util.concurrent.ConcurrentHashMap;*

## *import java.util.concurrent.locks.Lock;*

## *import java.util.concurrent.locks.ReentrantLock;*

## 

## *public class lab6 {*

## *private static final double WITHDRAWAL\_LIMIT = 1000.0;*

## *private static final ConcurrentHashMap<String, Account> accounts = new ConcurrentHashMap<>();*

## *private static final Logger logger = new Logger("atm\_log.txt");*

## *private static final Lock logLock = new ReentrantLock();*

## 

## *public static void main(String[] args) {*

## *Scanner scanner = new Scanner(System.in);*

## 

## *System.out.println("ATM Simulation Started");*

## 

## *while (true) {*

## *System.out.println("\n1. Create a new user session");*

## *System.out.println("0. Exit");*

## *System.out.print("Choose an option: ");*

## 

## *int choice;*

## *try {*

## *choice = Integer.parseInt(scanner.nextLine());*

## *} catch (NumberFormatException e) {*

## *System.out.println("Please enter a valid number");*

## *continue;*

## *}*

## 

## *if (choice == 0) {*

## *break;*

## *} else if (choice == 1) {*

## *System.out.print("Enter user ID: ");*

## *String userId = scanner.nextLine();*

## 

## *// Run the user session directly in the current thread*

## *runUserSession(userId, scanner);*

## *} else {*

## *System.out.println("Invalid option!");*

## *}*

## *}*

## 

## *logger.close();*

## *scanner.close();*

## *System.out.println("ATM Simulation Ended");*

## *}*

## 

## *private static void runUserSession(String userId, Scanner scanner) {*

## *System.out.println("\n--- User Session Started: " + userId + " ---");*

## 

## *boolean running = true;*

## *while (running) {*

## *System.out.println("\nUser: " + userId);*

## *System.out.println("1. Open Account");*

## *System.out.println("2. Close Account");*

## *System.out.println("3. Withdraw");*

## *System.out.println("4. Deposit");*

## *System.out.println("5. Check Balance");*

## *System.out.println("0. Exit");*

## *System.out.print("Choose an option: ");*

## 

## *int choice;*

## *try {*

## *choice = Integer.parseInt(scanner.nextLine());*

## *} catch (NumberFormatException e) {*

## *System.out.println("Please enter a valid number");*

## *continue;*

## *}*

## 

## *switch (choice) {*

## *case 0:*

## *running = false;*

## *break;*

## *case 1:*

## *openAccount(userId, scanner);*

## *break;*

## *case 2:*

## *closeAccount(userId, scanner);*

## *break;*

## *case 3:*

## *withdraw(userId, scanner);*

## *break;*

## *case 4:*

## *deposit(userId, scanner);*

## *break;*

## *case 5:*

## *checkBalance(userId, scanner);*

## *break;*

## *default:*

## *System.out.println("Invalid option!");*

## *}*

## *}*

## 

## *System.out.println("--- User Session Ended: " + userId + " ---");*

## *}*

## 

## *private static void openAccount(String userId, Scanner scanner) {*

## *System.out.print("Enter account number: ");*

## *String accountNumber = scanner.nextLine();*

## 

## *if (accounts.containsKey(accountNumber)) {*

## *System.out.println("Account already exists!");*

## *return;*

## *}*

## 

## *System.out.print("Enter initial deposit amount: ");*

## *double initialDeposit;*

## *try {*

## *initialDeposit = Double.parseDouble(scanner.nextLine());*

## *} catch (NumberFormatException e) {*

## *System.out.println("Please enter a valid amount");*

## *return;*

## *}*

## 

## *if (initialDeposit < 0) {*

## *System.out.println("Initial deposit cannot be negative!");*

## *return;*

## *}*

## 

## *Account newAccount = new Account(accountNumber, userId, initialDeposit);*

## *accounts.put(accountNumber, newAccount);*

## 

## *log(userId, "Account opened", accountNumber, initialDeposit);*

## *System.out.println("Account opened successfully!");*

## *}*

## 

## *private static void closeAccount(String userId, Scanner scanner) {*

## *System.out.print("Enter account number: ");*

## *String accountNumber = scanner.nextLine();*

## 

## *Account account = accounts.get(accountNumber);*

## *if (account == null) {*

## *System.out.println("Account does not exist!");*

## *return;*

## *}*

## 

## *if (!account.getOwnerId().equals(userId)) {*

## *System.out.println("This is not your account!");*

## *return;*

## *}*

## 

## *accounts.remove(accountNumber);*

## *log(userId, "Account closed", accountNumber, account.getBalance());*

## *System.out.println("Account closed successfully! Remaining balance: " + account.getBalance());*

## *}*

## 

## *private static void withdraw(String userId, Scanner scanner) {*

## *System.out.print("Enter account number: ");*

## *String accountNumber = scanner.nextLine();*

## 

## *Account account = accounts.get(accountNumber);*

## *if (account == null) {*

## *System.out.println("Account does not exist!");*

## *return;*

## *}*

## 

## *if (!account.getOwnerId().equals(userId)) {*

## *System.out.println("This is not your account!");*

## *return;*

## *}*

## 

## *System.out.print("Enter withdrawal amount: ");*

## *double amount;*

## *try {*

## *amount = Double.parseDouble(scanner.nextLine());*

## *} catch (NumberFormatException e) {*

## *System.out.println("Please enter a valid amount");*

## *return;*

## *}*

## 

## *if (amount <= 0) {*

## *System.out.println("Amount must be positive!");*

## *return;*

## *}*

## 

## *if (amount > WITHDRAWAL\_LIMIT) {*

## *System.out.println("Amount exceeds withdrawal limit of " + WITHDRAWAL\_LIMIT);*

## *return;*

## *}*

## 

## *if (amount > account.getBalance()) {*

## *System.out.println("Insufficient funds!");*

## *return;*

## *}*

## 

## *account.withdraw(amount);*

## *log(userId, "Withdrawal", accountNumber, amount);*

## *System.out.println("Withdrawal successful! New balance: " + account.getBalance());*

## *}*

## 

## *private static void deposit(String userId, Scanner scanner) {*

## *System.out.print("Enter account number: ");*

## *String accountNumber = scanner.nextLine();*

## 

## *Account account = accounts.get(accountNumber);*

## *if (account == null) {*

## *System.out.println("Account does not exist!");*

## *return;*

## *}*

## 

## *System.out.print("Enter deposit amount: ");*

## *double amount;*

## *try {*

## *amount = Double.parseDouble(scanner.nextLine());*

## *} catch (NumberFormatException e) {*

## *System.out.println("Please enter a valid amount");*

## *return;*

## *}*

## 

## *if (amount <= 0) {*

## *System.out.println("Amount must be positive!");*

## *return;*

## *}*

## 

## *account.deposit(amount);*

## *log(userId, "Deposit", accountNumber, amount);*

## *System.out.println("Deposit successful! New balance: " + account.getBalance());*

## *}*

## 

## *private static void checkBalance(String userId, Scanner scanner) {*

## *System.out.print("Enter account number: ");*

## *String accountNumber = scanner.nextLine();*

## 

## *Account account = accounts.get(accountNumber);*

## *if (account == null) {*

## *System.out.println("Account does not exist!");*

## *return;*

## *}*

## 

## *if (!account.getOwnerId().equals(userId)) {*

## *System.out.println("This is not your account!");*

## *return;*

## *}*

## 

## *System.out.println("Current balance: " + account.getBalance());*

## *}*

## 

## *private static void log(String userId, String operation, String accountNumber, double amount) {*

## *logLock.lock();*

## *try {*

## *logger.log(userId, operation, accountNumber, amount);*

## *} finally {*

## *logLock.unlock();*

## *}*

## *}*

## 

## *static class Account {*

## *private final String accountNumber;*

## *private final String ownerId;*

## *private double balance;*

## *private final Lock lock = new ReentrantLock();*

## 

## *public Account(String accountNumber, String ownerId, double initialBalance) {*

## *this.accountNumber = accountNumber;*

## *this.ownerId = ownerId;*

## *this.balance = initialBalance;*

## *}*

## 

## *public String getAccountNumber() {*

## *return accountNumber;*

## *}*

## 

## *public String getOwnerId() {*

## *return ownerId;*

## *}*

## 

## *public double getBalance() {*

## *lock.lock();*

## *try {*

## *return balance;*

## *} finally {*

## *lock.unlock();*

## *}*

## *}*

## 

## *public void withdraw(double amount) {*

## *lock.lock();*

## *try {*

## *if (amount <= balance) {*

## *balance -= amount;*

## *}*

## *} finally {*

## *lock.unlock();*

## *}*

## *}*

## 

## *public void deposit(double amount) {*

## *lock.lock();*

## *try {*

## *balance += amount;*

## *} finally {*

## *lock.unlock();*

## *}*

## *}*

## *}*

## 

## *static class Logger {*

## *private PrintWriter writer;*

## *private final SimpleDateFormat dateFormat = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");*

## 

## *public Logger(String filename) {*

## *try {*

## *writer = new PrintWriter(new FileWriter(filename, true));*

## *} catch (IOException e) {*

## *System.err.println("Error creating log file: " + e.getMessage());*

## *}*

## *}*

## 

## *public void log(String userId, String operation, String accountNumber, double amount) {*

## *if (writer != null) {*

## *String timestamp = dateFormat.format(new Date());*

## *writer.println(timestamp + " | User: " + userId + " | Operation: " + operation +*

## *" | Account: " + accountNumber + " | Amount: " + amount);*

## *writer.flush();*

## *}*

## *}*

## 

## *public void close() {*

## *if (writer != null) {*

## *writer.close();*

## *}*

## *}*

## *}*

## *}*

## 

## 

## **3. Результати тестування**

## Застосунок протестовано на:

## **Windows 10**

## **Arch Linux**

## Перевірено сценарії:

## одночасне відкриття декількох рахунків під різними користувачами;

## зняття та поповнення з урахуванням ліміту;

## закриття рахунків із залишком;

## коректне логування всіх операцій у файл atm\_log.txt.

## 

## **4. Висновки**

## Розроблено крос‑платформний консольний застосунок для симуляції банкомату з багатокористувацькою роботою та детальним логуванням.

## Програма забезпечує:

## безпечну роботу з балансом у багатопоточному середовищі;

## перевірку обмежень на зняття коштів;

## повний журнал операцій з таймстемпами.

## Тестування на Windows і Arch Linux підтвердило коректність і стабільність роботи. Перспективи: додати GUI-інтерфейс, розширити функціонал (переклади, звіти) та підключення до бази даних.