|  |  |
| --- | --- |
|  | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **Компьютерные системы и сети (ИУ6)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.01 Информатика и вычислительная техника**

МАГИСТЕРСКАЯ ПРОГРАММА **09.04.01/07 Интеллектуальные системы анализа, обработки и интерпретации больших данных**

**Отчет**

|  |  |
| --- | --- |
| **по лабораторной работе №** | 8 |

**Название:** Потоки

**Дисциплина:** Языки программирования для работы с большими данными

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Студент | ИУ6-22М |  |  | С.В. Астахов | |
|  | (Группа) |  | (Подпись, дата) | | (И.О. Фамилия) |
|  |  |  |  | |  |
| Преподаватель |  |  |  | | П.В. Степанов |
|  |  |  | (Подпись, дата) | | (И.О. Фамилия) |

Москва, 2024

**Цель работы:** освоить базовые принципы работы с потоками на языке Java.

**Вариант: 1.**

**Задание 1:** Реализовать многопоточное приложение “Банк”. Имеется банковский счет. Сделать синхронным пополнение и снятие денежных средств на счет/со счет случайной суммой. При каждой операции (пополнения или снятие) вывести текущий баланс счета. В том случае, если денежных средств недостаточно – вывести сообщение.

Код решения приведен в листинге 1.

Листинг 1 — реализация решения

|  |
| --- |
| package org.example;  import java.io.\*;  import java.util.\*;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  import java.util.stream.Collectors;  import java.util.stream.IntStream;  public class Main {  public static void main(String[] args) throws IOException, InterruptedException {  System.out.println("=== Bank with threads work example ===\n\n");  BankThread thread1 = new BankThread(1, 1, "register", 0);  BankThread thread2 = new BankThread(2, 1, "put\_cash", 10);  BankThread thread3 = new BankThread(3, 1, "get\_cash", 10);  BankThread thread4 = new BankThread(4, 1, "put\_cash", 5);  BankThread thread5 = new BankThread(5, 1, "get\_cash", 3);  thread1.start();  Thread.sleep(1000);  thread2.start();  Thread.sleep(1000);  thread2.join(); // put everywhere for correct work  thread3.start();  Thread.sleep(1000);  thread4.start();  Thread.sleep(1000);  thread5.start();  }  }  // singletone  class Bank {  private static HashMap<Integer, Integer> accounts = new HashMap<>();  public static int getCash(int client\_id, int amt) throws BankException, InterruptedException {  synchronized (accounts) {  if (!accounts.containsKey(client\_id)) {  throw new BankException("client doesn't exist");  }  if (accounts.getOrDefault(client\_id, 0) < amt || amt <= 0) {  throw new BankException("incorrect balance or amount");  }  accounts.put(client\_id, accounts.getOrDefault(client\_id, 0) - amt);  Thread.sleep(3000);  return accounts.getOrDefault(client\_id, 0);  }  }  public synchronized static int putCash(int client\_id, int amt) throws BankException, InterruptedException {  synchronized (accounts) {  if (!accounts.containsKey(client\_id)) {  throw new BankException("client doesn't exist");  }  if (amt <= 0) {  throw new BankException("incorrect amount");  }  accounts.put(client\_id, accounts.getOrDefault(client\_id, 0) + amt);  Thread.sleep(3000);  return accounts.getOrDefault(client\_id, 0);  }  }  public synchronized static void registerClient(int client\_id) throws BankException, InterruptedException {  synchronized (accounts) {  if (accounts.containsKey(client\_id)) {  throw new BankException("client already exists");  }  Thread.sleep(3000);  accounts.put(client\_id, 0);  }  }  public static class BankException extends Exception {  BankException(String msg){  super(msg);  }  }  }  class BankThread extends Thread {  public int amt;  public int clientId;  public int id;  public String operation;  BankThread(int p\_id, int p\_client\_id, String p\_operation, int p\_amt) {  id = p\_id;  amt = p\_amt;  clientId = p\_client\_id;  operation = p\_operation;  }  @Override  public void run() {  System.out.printf("\n>>> <begin ts:%d> tx-%d client-%d operation-%s amt-%d\n", Calendar.getInstance().get(Calendar.SECOND) ,id, clientId, operation, amt);  switch (operation) {  case "register" -> {  try {  Bank.registerClient(clientId);  System.out.printf("> [%d] client registered\n", id);  } catch (Bank.BankException | InterruptedException e) {  System.out.printf("> [%d] Exception: %s\n", id, e);  }  }  case "get\_cash" -> {  try {  int balance = Bank.getCash(clientId, amt);  System.out.printf("> [%d] get cash, balance: %d\n", id, balance);  } catch (Bank.BankException | InterruptedException e) {  System.out.printf("> [%d] Exception: %s\n", id, e);  }  }  case "put\_cash" -> {  try {  int balance = Bank.putCash(clientId, amt);  System.out.printf("> [%d] put cash, balance: %d\n", id, balance);  } catch (Bank.BankException | InterruptedException e) {  System.out.printf("> [%d] Exception: %s\n", id, e);  }  }  default -> System.out.printf("> [%d] unknown operation: %s \n", id, operation);  }  System.out.printf(">>> <finish ts:%d> tx-%d\n", Calendar.getInstance().get(Calendar.SECOND), id);  }  } |

Результат исполнения программы приведен в листинге 2.

Листинг 2 — результат исполнения программы

|  |
| --- |
| === Bank with threads work example ===  >>> <begin ts:25> tx-1 client-1 operation-register amt-0  >>> <begin ts:26> tx-2 client-1 operation-put\_cash amt-10  > [1] client registered  >>> <finish ts:28> tx-1  > [2] put cash, balance: 10  >>> <finish ts:31> tx-2  >>> <begin ts:31> tx-3 client-1 operation-get\_cash amt-10  >>> <begin ts:32> tx-4 client-1 operation-put\_cash amt-5  >>> <begin ts:33> tx-5 client-1 operation-get\_cash amt-3  > [3] get cash, balance: 0  > [5] Exception: org.example.Bank$BankException: incorrect balance or amount  >>> <finish ts:34> tx-3  >>> <finish ts:34> tx-5  > [4] put cash, balance: 5  >>> <finish ts:37> tx-4 |

**Задание 2:** Реализовать многопоточное приложение “Робот”. Надо написать робота, который умеет ходить. За движение каждой его ноги отвечает отдельный поток. Шаг выражается в выводе в консоль LEFT или RIGHT.

Код решения приведен в листинге 3.

Листинг 3 — реализация решения

|  |
| --- |
| package org.example;  import java.io.\*;  import java.util.\*;  import java.util.concurrent.atomic.AtomicBoolean;  import java.util.regex.Matcher;  import java.util.regex.Pattern;  import java.util.stream.Collectors;  import java.util.stream.IntStream;  public class Main {  public static void main(String[] args) throws IOException, InterruptedException {  System.out.println("=== Robot with threads work example ===\n\n");  Robot.walk();  }  }  // singletone  class Robot {  static AtomicBoolean leg = new AtomicBoolean(false);  public static void walk(){  RobotThread th1 = new RobotThread(true, "RIGHT");  RobotThread th2 = new RobotThread(false, "LEFT");  th1.start();  th2.start();  }  public static class RobotThread extends Thread {  private boolean leg;  private String leg\_str;  RobotThread(boolean leg\_fl, String p\_leg\_str) {  leg = leg\_fl;  leg\_str = p\_leg\_str;  }  @Override  public void run() {  while(true) {  int sleep\_ms = (int) ((Math.random() \* (3000 - 1000)) + 1000);  try {  Thread.sleep(sleep\_ms);  } catch (InterruptedException e) {  throw new RuntimeException(e);  }  if(leg == Robot.leg.get()) {  System.out.printf("%s <ts:%d>\n", leg\_str, Calendar.getInstance().get(Calendar.SECOND) );  Robot.leg.set(!Robot.leg.get());  }  }  }  }  } |

Результат исполнения программы приведен в листинге 4.

Листинг 4 — результат исполнения программы

|  |
| --- |
| === Robot with threads work example ===  LEFT <ts:58>  RIGHT <ts:58>  LEFT <ts:1>  RIGHT <ts:1>  LEFT <ts:4>  RIGHT <ts:4>  LEFT <ts:6>  RIGHT <ts:8>  LEFT <ts:10>  RIGHT <ts:11>  LEFT <ts:11>  RIGHT <ts:12>  LEFT <ts:13>  RIGHT <ts:14> |

**Вывод:** в ходе лабораторной работы были освоены базовые принципы работы с потоками на языке Java.