|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **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 **09.04.01/12 Интеллектуальный анализ больших**

**данных в системах поддержки принятия решений**

**Отчет**

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

Вариант 6

**Название:** Потоки.

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-23М |  |  | В.А. Гордеев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | П.В. Степанов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2023

**ВВЕДЕНИЕ**

**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.
  2. Реализовать многопоточное приложение “Магазин”. Вся цепочка: производитель-магазин-покупатель. Пока производитель не поставит на склад продукт, покупатель не может его забрать. Реализовать приход товара от производителя в магазин случайным числом. В том случае, если товара в магазине не хватает– вывести сообщение.

**ОСНОВНАЯ ЧАСТЬ**

**Задание 1.1**

Реализовано многопоточное приложение “Банк”. Имеется банковский счет. Сделано синхронное пополнение и снятие денежных средств на счет/со счета со случайной суммой. При каждой операции (пополнения или снятие) выводится текущий баланс счета. В том случае, если денежных средств недостаточно – выводится сообщение.

Код программы приведен в листинге 1. Результат выполнения программы представлен на рисунке 1.

Листинг 1 – Код класса Main

import java.util.concurrent.Semaphore;  
  
// Реализовать многопоточное приложение “Магазин”.  
// Вся цепочка: производитель-магазин-покупатель.  
// Пока производитель не поставит на склад продукт, покупатель не может его забрать.  
// Реализовать приход товара от производителя в магазин случайным числом.  
// В том случае, если товара в магазине не хватает – вывести сообщение.  
class Main {  
 public static void main(String[] args) {  
 var shop = new Shop(10);  
  
 // Создаем поток производителя  
 var producerThread = new Thread(() -> {  
 while (true) {  
 var amount = (int) (Math.random() \* 5) + 1;  
 shop.put(amount);  
 try {  
 Thread.sleep(500);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 // Создаем поток покупателя  
 var consumerThread = new Thread(() -> {  
 while (true) {  
 var amount = (int) (Math.random() \* 3) + 1;  
 shop.get(amount);  
 try {  
 Thread.sleep(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
 producerThread.start();  
 consumerThread.start();  
 }  
}  
  
class Shop {  
 private int stock;  
 // Для синхронизации доступа производителя к ресурсу  
 private final Semaphore producerSemaphore;  
 // Для синхронизации доступа покупателя к ресурсу  
 private final Semaphore consumerSemaphore;  
  
 public Shop(int stock) {  
 this.stock = stock;  
 producerSemaphore = new Semaphore(1);  
 consumerSemaphore = new Semaphore(0);  
 }  
  
 public void put(int amount) {  
 try {  
 producerSemaphore.acquire();  
 stock += amount;  
 System.out.println("Producer put new " + amount + " items, stock: " + stock);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 } finally {  
 consumerSemaphore.release();  
 }  
 }  
  
 public void get(int amount) {  
 try {  
 consumerSemaphore.acquire();  
 if (stock < amount) {  
 System.out.println("Consumer wants " + amount + ", but there are " + stock + " items in stock");  
 } else {  
 stock -= amount;  
 System.out.println("Consumer bought " + amount + " items, stock: " + stock);  
 }  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 } finally {  
 producerSemaphore.release();  
 }  
 }  
}

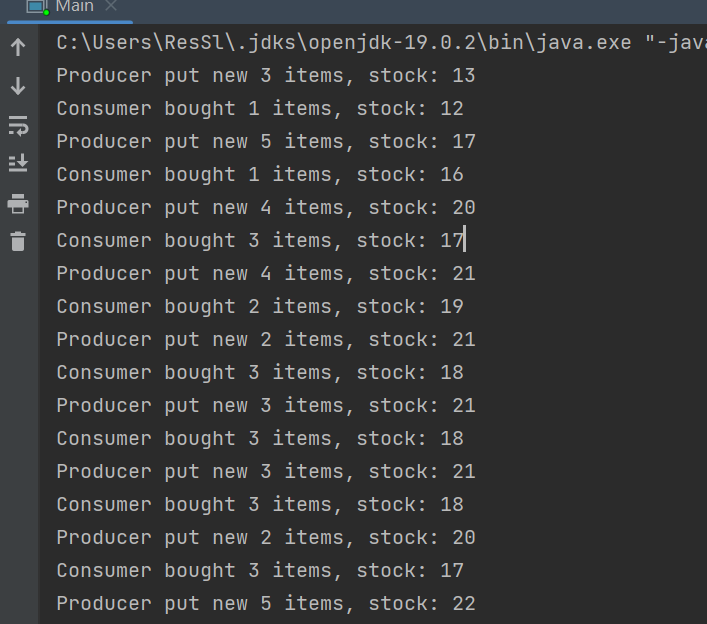


Рисунок 1 – Результат работы программы 1.1

**Задание 1.2**

Реализовано многопоточное приложение “Магазин”. Вся цепочка: производитель-магазин-покупатель. Пока производитель не поставит на склад продукт, покупатель не может его забрать. Реализован приход товара от производителя в магазин случайным числом. В том случае, если товара в магазине не хватает– выводится сообщение.

Код программы приведен в листинге 2. Результат выполнения программы представлен на рисунке 1.

Листинг 2 – Код класса Main

import java.util.Random;  
import java.util.concurrent.Semaphore;  
  
public class Main {  
 private static final Random random = new Random();  
  
 public static void main(String[] args) {  
 Bank bank = new Bank(1000);  
  
 Thread depositThread = new Thread(() -> {  
 while (true) {  
 try {  
 bank.deposit(random.nextInt(100));  
 Thread.sleep(1000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 Thread withdrawThread = new Thread(() -> {  
 while (true) {  
 try {  
 bank.withdraw(random.nextInt(100));  
 Thread.sleep(2000);  
 } catch (InterruptedException e) {  
 e.printStackTrace();  
 }  
 }  
 });  
  
 depositThread.start();  
 withdrawThread.start();  
 }  
}  
class Bank {  
 private int balance;  
 private final Semaphore withdrawSemaphore = new Semaphore(1);  
 private final Semaphore depositSemaphore = new Semaphore(1);  
  
 public Bank(int balance) {  
 this.balance = balance;  
 }  
  
 public void deposit(int amount) throws InterruptedException {  
 depositSemaphore.acquire();  
 balance += amount;  
 System.out.println("Depositing " + amount + " New balance: " + balance);  
 depositSemaphore.release();  
 }  
  
 public void withdraw(int amount) throws InterruptedException {  
 withdrawSemaphore.acquire();  
 if (balance < amount) {  
 System.out.println("Balance less then amount. Current balance: " + balance);  
 withdrawSemaphore.release();  
 return;  
 }  
 balance -= amount;  
 System.out.println("Withdrawing " + amount + " New balance: " + balance);  
 withdrawSemaphore.release();  
 }  
}

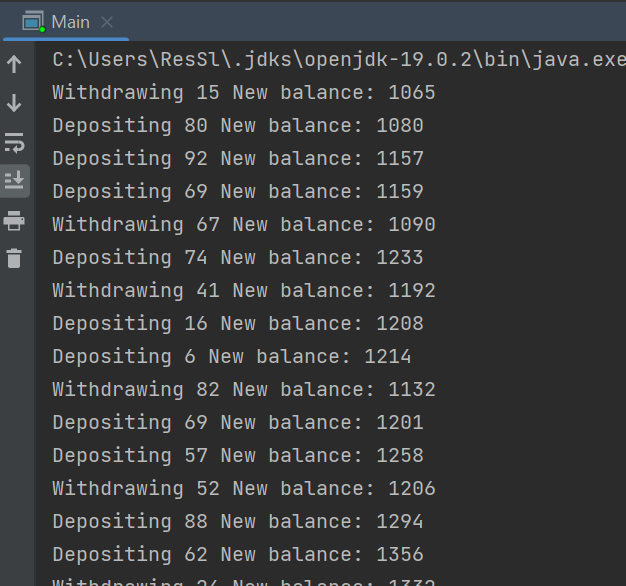


Рисунок 2 – Результат работы программы 1.2

**Вывод:** В результате выполнения лабораторной работы были получены практические навыки для работы с потоками в Java. Также реализованы многопоточные приложения.