|  |  |
| --- | --- |
| 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**

**Название:** Потоки

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

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

Москва, 2023

**Задание 1:** 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.

Решение задания представлено в листинге 1.

Листинг 1 – Программный код модуля Main.java для задания 1

import java.util.concurrent.ArrayBlockingQueue;

import java.util.concurrent.BlockingQueue;

class Producer implements Runnable {

private final BlockingQueue<Integer> queue;

public Producer(BlockingQueue<Integer> queue) {

this.queue = queue;

}

@Override

public void run() {

try {

while (true) {

int product = produceProduct();

queue.put(product);

System.out.println("Производитель поставил товар: " + product);

Thread.sleep(1000);

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

private int produceProduct() {

return (int) (Math.random() \* 10) + 1;

}

}

class Consumer implements Runnable {

private final BlockingQueue<Integer> queue;

public Consumer(BlockingQueue<Integer> queue) {

this.queue = queue;

}

@Override

public void run() {

try {

while (true) {

int product = queue.take();

System.out.println("Покупатель купил товар: " + product);

Thread.sleep(2000);

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

}

public class Main {

public static void main(String[] args) {

BlockingQueue<Integer> queue = new ArrayBlockingQueue<>(5);

Producer producer = new Producer(queue);

Consumer consumer = new Consumer(queue);

Thread producerThread = new Thread(producer);

Thread consumerThread = new Thread(consumer);

producerThread.start();

consumerThread.start();

}

}

**Задание 2:** 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.

Решение задания представлено в листинге 2.

Листинг 2 – Программный код модуля Main.java для задания 2

class BankAccount {

private int balance;

public BankAccount(int balance) {

this.balance = balance;

}

public synchronized void deposit(int amount) {

balance += amount;

System.out.println("Пополнение на сумму " + amount + " руб. Баланс: " + balance + " руб.");

}

public synchronized void withdraw(int amount) {

if (balance >= amount) {

balance -= amount;

System.out.println("Снятие на сумму " + amount + " руб. Баланс: " + balance + " руб.");

} else {

System.out.println("Недостаточно денежных средств на счете.");

}

}

}

class BankThread implements Runnable {

private final BankAccount account;

public BankThread(BankAccount account) {

this.account = account;

}

@Override

public void run() {

try {

while (true) {

int amount = generateRandomAmount();

if (Math.random() < 0.5) {

account.deposit(amount);

} else {

account.withdraw(amount);

}

Thread.sleep(1000);

}

} catch (InterruptedException e) {

Thread.currentThread().interrupt();

}

}

private int generateRandomAmount() {

return (int) (Math.random() \* 100) + 1;

}

}

public class Main {

public static void main(String[] args) {

BankAccount account = new BankAccount(1000);

BankThread thread1 = new BankThread(account);

BankThread thread2 = new BankThread(account);

Thread t1 = new Thread(thread1);

Thread t2 = new Thread(thread2);

t1.start();

t2.start();

}

}

**Вывод:** в ходе выполнения данной лабораторной работы были изучены и применены на практике приемы работы с многопоточными приложениями на языке Java.