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

**ОТЧЕТ**

**по лабораторной работе № 8**

|  |  |
| --- | --- |
| **Название:** | Потоки |
| **Дисциплина:** | Языки программирования для работы с большими данными |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ6-22М |  | 12.05.2023 | И.Б. Нуриддинов |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | П.В. Степанов |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2023

Цель работы: Освоение и применение концепций потоков в языке программирования Java.

*Вариант 1. Задание 1.* 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.

class LegThread implements Runnable {

private String legName;

public LegThread(String legName) {

this.legName = legName;

}

@Override

public void run() {

for (int i = 0; i < 5; i++) {

if (legName.equals("LEFT")) {

System.out.println("LEFT");

} else if (legName.equals("RIGHT")) {

System.out.println("RIGHT");

}

try {

Thread.sleep(1000); // Задержка между шагами

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

public class RobotApplication {

public static void main(String[] args) {

Thread leftLegThread = new Thread(new LegThread("LEFT"));

Thread rightLegThread = new Thread(new LegThread("RIGHT"));

leftLegThread.start();

rightLegThread.start();

}

}

*Вариант 1. Задание 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.

class Producer implements Runnable {

private Store store;

private Random random = new Random();

public Producer(Store store) {

this.store = store;

}

@Override

public void run() {

for (int i = 0; i < 5; i++) {

int productCount = random.nextInt(10) + 1; // Случайное количество продуктов от 1 до 10

store.produce(productCount);

try {

Thread.sleep(random.nextInt(1000));

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class Consumer implements Runnable {

private Store store;

private Random random = new Random();

public Consumer(Store store) {

this.store = store;

}

@Override

public void run() {

for (int i = 0; i < 5; i++) {

int productCount = random.nextInt(10) + 1; // Случайное количество продуктов от 1 до 10

store.consume(productCount);

try {

Thread.sleep(random.nextInt(1000));

} catch (InterruptedException e) {

e.printStackTrace();

}

}

}

}

class Store {

private int productCount = 0;

private Lock lock = new ReentrantLock();

private Semaphore producerSemaphore = new Semaphore(1);

private Semaphore consumerSemaphore = new Semaphore(0);

public void produce(int count) {

try {

producerSemaphore.acquire();

lock.lock();

productCount += count;

System.out.println("Производитель поставил на склад " + count + " продукт(ов). Общее количество: " + productCount);

consumerSemaphore.release();

} catch (InterruptedException e) {

e.printStackTrace();

} finally {

lock.unlock();

}

}

public void consume(int count) {

try {

consumerSemaphore.acquire();

lock.lock();

if (productCount >= count) {

productCount -= count;

System.out.println("Покупатель забрал " + count + " продукт(ов). Общее количество: " + productCount);

} else {

System.out.println("Недостаточно продуктов на складе");

}

producerSemaphore.release();

} catch (InterruptedException e) {

e.printStackTrace();

} finally {

lock.unlock();

}

}

}

public class ShopApplication {

public static void main(String[] args) {

Store store = new Store();

Thread producerThread = new Thread(new Producer(store));

Thread consumerThread = new Thread(new Consumer(store));

producerThread.start();

consumerThread.start();

}

}

Вывод: Лабораторная работа позволила овладеть важными навыками работы с потоками и многопоточностью. Освоение концепций потоков и применение соответствующих методов и операций позволяет создавать эффективные и отзывчивые программы, способные параллельно выполнять задачи и использовать ресурсы системы наиболее эффективно.

Многопоточность является важным аспектом разработки программного обеспечения, особенно в сфере высокопроизводительных вычислений, серверных приложений или при работе с большими объемами данных. Правильное использование потоков и синхронизация доступа к общим ресурсам позволяют достичь более высокой производительности, эффективности и отзывчивости системы.