ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. А.Н. Тихонова

Департамент компьютерной инженерии

Отчет по практической работе №3

«Синхронизация»

по курсу «Распределенные базы данных и сетевые вычисления»

Выполнили:
Студенты группы МКС212
Журсунова Найля
Шабалина Анастасия Владимировна
Приняла:
Байбикова Татьяна Николаевна

Москва 2022 г.

Оглавление

1. Задание....3
2. Синхронизация потоков....3
2.1. Блокировки....3
2.2. Методы wait(), notify(), notifyAll() класса Object....4
2.3. Реализация синхронизации....4
2.3.1. Реализация....4
2.3.2. Результат выполнения работы с использованием notify()....7
2.3.3. Результат выполнения работы с использованием notifyAll()....7
3. Выводы....8

1. Задание

Изучить теоретический материал для практики №1. Разобрать прилагаемые примеры. Создать примеры, в которых потоки синхронизируются. Подготовить отчет, включить в него свои примеры (листинги) с кратким описанием:

1. Блокировки.
2. Методы wait(), notify(), notifyAll() класса Object.

В отчет включить:

- задание;
- разработанную программу или набор программ;
- результаты работы программы;
- краткие выводы.

Работа выполнялась на языке программирования Java.

2. Синхронизация потоков

Java поддерживает несколько потоков для выполнения. Это может привести к тому, что два или более потока получат доступ к одному и тому же полю или объекту. ***Синхронизация*** — это процесс, который позволяет выполнять все параллельные потоки в программе синхронно. Она позволяет избежать ошибок согласованности памяти, вызванные из-за непоследовательного доступа к общей памяти.

2.1. Блокировки

С каждым объектом ассоциирован некоторый монитор, а потоки могут его заблокировать "lock" или разблокировать "unlock". ***Монитор*** — это действительно механизм обеспечения синхронизации доступа нескольких потоков к общим ресурсам.

Когда метод объявлен как синхронизированный — нить держит монитор для объекта, метод которого исполняется. Если другой поток выполняет синхронизированный метод, наш поток заблокируется до тех пор, пока другой поток не отпустит монитор. Синхронизация достигается в Java использованием зарезервированного слова ***synchronized***.

synchronized можно использовать только с методами и блоками кода. Эти методы или блоки могут быть статическими или не-статическими.

Когда какой-либо поток входит в синхронизированный метод или блок, он приобретает блокировку, и всякий раз, когда поток выходит из синхронизированного метода или блока, JVM снимает блокировку. Блокировка снимается, даже если нить оставляет синхронизированный метод после завершения из-за каких-либо ошибок или исключений.

2.2. Методы wait(), notify(), notifyAll() класса Object

Каждый объект в *Java* имеет не только блокировку для ***synchronized*** блоков и методов, но и так называемый ***wait-set***, набор потоков исполнения. В этот набор входят ряд методов класса ***Object***:

- **wait()**: освобождает монитор и переводит вызывающий поток в состояние ожидания до тех пор, пока другой поток не вызовет метод notify()
- **notify()**: продолжает работу потока, у которого ранее был вызван метод wait()
- **notifyAll()**: возобновляет работу всех потоков, у которых ранее был вызван метод wait()

Все эти методы вызываются только из синхронизированного контекста – синхронизированного блока или метода.

2.3. Реализация синхронизации

2.3.1. Реализация

В ***PetrolSyn.java*** продемонстрирована реализация синхронизации потоков с использованием методов из ***wait-set***. Все пояснения в комментариях к коду.

```
package rbd.thread;

// Создание потока с помощью реализации интерфейса Runnable
class MyRunnable implements Runnable {
  private int count; //переменная для хранения кол-ва авто
  private Object lock; //объект, к которому будет применена блокировка

  MyRunnable(Object obj, int carCount) { // Конструктор
    count = carCount;
    lock = obj;
  }
```

```
  @Override // Реализация метода run() из интерфейса Runnable
  public void run() {
    // поток будет ждать, пока его не оповестят через lock
    synchronized (lock) {  //блокировка
      int n=0; //переменная для подсчета уже заправленных автомобилей
      try {
        System.out.println("Ждем открытия " + Thread.currentThread().getName());
        lock.wait(); //приостанавливаем поток
        System.out.println(Thread.currentThread().getName() + " открыта...");
        for (int i=count; i>0; i--) {
          n++;
          System.out.println("Заправлено автомобилей на " +
Thread.currentThread().getName() + ": " + n);
        }
      }
      catch (InterruptedException e) {
        System.out.println("Поток на " + Thread.currentThread().getName() + "
прерван.");
      }
      System.out.println("Поток на " + Thread.currentThread().getName() + "
завершен.");
    }
  }
}

public class PetrolSyn {
  public static void main(String[] args) {
    Object lock = new Object(); //создаем объект класса Object для вызова методов
из wait-set
    Thread t1 = new Thread(new MyRunnable(lock, 12), "Бензоколонка №1");
    Thread t2 = new Thread(new MyRunnable(lock, 9), "Бензоколонка №2");
    // создание новых потоков
    t1.start();
    t2.start();
    try {
      Thread.currentThread().sleep(3000);
      System.out.println("Заправка открыта...");
      synchronized(lock) {
            //два потока ожидают на объекте lock
            //метод notify() разбудит только один из них – другой поток все еще
ждет свое уведомление.
            lock.notify();
            //lock.notifyAll();
            System.out.println("Бензоколонки готовы к работе...");
      }
    }
    catch (InterruptedException e) {
      System.out.println("Поток на заправке прерван.");
    }

    try {
```

```
      t1.join(); // ожидаем завершения дочернего потока, чтобы главный поток
завершился последним
      //t2.join();
    }
    catch (InterruptedException e) {
      System.out.println("Поток на бензоколонках прерван.");
    }
    System.out.println("Заправка закрыта.");

  }
}
```

Для использования метода **notifyAll()** поменяем код в ***main.***

```
public class PetrolSyn {
  public static void main(String[] args) {
    Object lock = new Object(); //создаем объект класса Object для вызова методов
из wait-set
    Thread t1 = new Thread(new MyRunnable(lock, 12), "Бензоколонка №1");
    Thread t2 = new Thread(new MyRunnable(lock, 9), "Бензоколонка №2");
    // создание новых потоков
    t1.start();
    t2.start();
    try {
      Thread.currentThread().sleep(3000);
      System.out.println("Заправка открыта...");
      synchronized(lock) {
            //два потока ожидают на объекте lock
            //метод notifyAll() возобновляет выполнение всех потоков
            lock.notifyAll();
            System.out.println("Бензоколонки готовы к работе...");
      }
    }
    catch (InterruptedException e) {
      System.out.println("Поток на заправке прерван.");
    }

    // ожидаем завершения дочерних потоков, чтобы главный поток завершился
последним
    try {
      t1.join();
      t2.join();
    }
    catch (InterruptedException e) {
      System.out.println("Поток на бензоколонках прерван.");
    }
    System.out.println("Заправка закрыта.");

  }
```

2.3.2. Результат выполнения работы с использованием notify()

```
Ждем открытия Бензоколонка №1
Ждем открытия Бензоколонка №2
Заправка открыта ...
Бензоколонки готовы к работе ...
Бензоколонка №1 открыта ...
Заправлено автомобилей на Бензоколонка №1: 1
Заправлено автомобилей на Бензоколонка №1: 2
Заправлено автомобилей на Бензоколонка №1: 3
Заправлено автомобилей на Бензоколонка №1: 4
Заправлено автомобилей на Бензоколонка №1: 5
Заправлено автомобилей на Бензоколонка №1: 6
Заправлено автомобилей на Бензоколонка №1: 7
Заправлено автомобилей на Бензоколонка №1: 8
Заправлено автомобилей на Бензоколонка №1: 9
Заправлено автомобилей на Бензоколонка №1: 10
Заправлено автомобилей на Бензоколонка №1: 11
Заправлено автомобилей на Бензоколонка №1: 12
Поток на Бензоколонка №1 завершен.
Заправка закрыта.
```

2.3.3. Результат выполнения работы с использованием notifyAll()

```
Ждем открытия Бензоколонка №1
Ждем открытия Бензоколонка №2
Заправка открыта ...
Бензоколонки готовы к работе ...
Бензоколонка №1 открыта ...
Заправлено автомобилей на Бензоколонка №1: 1
Заправлено автомобилей на Бензоколонка №1: 2
Заправлено автомобилей на Бензоколонка №1: 3
Заправлено автомобилей на Бензоколонка №1: 4
Заправлено автомобилей на Бензоколонка №1: 5
Заправлено автомобилей на Бензоколонка №1: 6
Заправлено автомобилей на Бензоколонка №1: 7
Заправлено автомобилей на Бензоколонка №1: 8
Заправлено автомобилей на Бензоколонка №1: 9
Заправлено автомобилей на Бензоколонка №1: 10
Заправлено автомобилей на Бензоколонка №1: 11
Заправлено автомобилей на Бензоколонка №1: 12
Поток на Бензоколонка №1 завершен.
Бензоколонка №2 открыта ...
Заправлено автомобилей на Бензоколонка №2: 1
Заправлено автомобилей на Бензоколонка №2: 2
Заправлено автомобилей на Бензоколонка №2: 3
Заправлено автомобилей на Бензоколонка №2: 4
Заправлено автомобилей на Бензоколонка №2: 5
Заправлено автомобилей на Бензоколонка №2: 6
Заправлено автомобилей на Бензоколонка №2: 7
Заправлено автомобилей на Бензоколонка №2: 8
Заправлено автомобилей на Бензоколонка №2: 9
Поток на Бензоколонка №2 завершен.
Заправка закрыта.
```

3. Выводы

В результате выполнения задания 3 практической работы №1 был получен навык работы с синхронизацией потоков на языке Java. На практике были использованы методы **wait(), notify()** и **notifyAll().**