

Объявляются и инициализируются глобальные данные/переменные, которые требуют синхронизации (например, "count"). Связанный мьютекс также объявляется и инициализируется.

Создаются потоки А и В для выполнения работы.

Поток А:

- Работает до тех пор, пока не наступит определенное условие (например, значение "счетчика" достигнет указанного значения).
- Блокирует связанный мьютекс и проверяет значение глобальной переменной.
- Вызывает `pthread_cond_wait()` для ожидания сигнала от потока В. Заметьте, что при вызове `pthread_cond_wait()` автоматически и атомарно разблокируется связанный мьютекс, чтобы поток В мог использовать его.
- После получения сигнала поток А просыпается, а мьютекс автоматически блокируется атомарно.

Поток В:

- Изменяет значение глобальной переменной, на которую ожидает поток А.
- Проверяет значение глобальной переменной, чтобы удовлетворить нужное условие потока А. Если условие выполнено, поток В посылает сигнал потоку А.
- Разблокирует мьютекс.

Код программы:

```
public class Lab3_task1_sem2_114m {

    public static void main(String[] args) {
        Common com = new Common();
        //создание и запуск потоков А и В
        Thread_A threadA = new Thread_A(com);
        Thread_B threadB = new Thread_B(com);
        new Thread(threadA).start();
        new Thread(threadB).start();
    }
    //класс Common содержит два синхронизированных метода для
    //увеличения счетчика add() и его уменьшения cut()
    public static class Common{
        private int count = 0; //глобальная переменная
        //Уменьшает значение глобальной переменной count, которую ожидает
        Thread_A.
        public synchronized void cut(){
            while (count < 1) {
                try {wait();}
            } catch (InterruptedException e) {}
        }
    }
}
```

```

        count--; //уменьшает глобальную переменную
        System.out.println("Thread_B: count -1");
        System.out.println("\tcount= " + count);
        notify();
    }
    //увеличивает значение счетчика count до заданного
    public synchronized void add() {
        while (count >= 2) {
            //wait() автоматически и атомарно разблокирует связанную
            переменную мьютекса, чтобы она могла использоваться Thread_B.
            try {wait();
            } catch (InterruptedException e) {}
        }
        count++; //увеличивает глобальную переменную
        System.out.println("Thread_A: count +1");
        System.out.println("\tcount= " + count);
        //
        notify();
    }
}
//класс потока A реализуют интерфейс Runnable, методы
run() переопределен
public static class Thread_A implements Runnable{
    Common com;
    Thread_A(Common com) {
        this.com=com;
    }
    @Override
    public void run() {
        //количество вызова метода увеличения ограничено
        for (int i = 1; i < 4; i++)
            {com.add();
            }
    }
}
//класс потока B реализуют интерфейс Runnable, метод
run() переопределен
public static class Thread_B implements Runnable{
    Common com;
    Thread_B(Common com) {
        this.com=com;
    }
    @Override
    public void run(){
        //количество вызова метода уменьшения ограничено
        for (int i = 1; i < 4; i++) {
            com.cut();
        }
    }
}
}

```

Вывод:

```
Вывод - lab3_task1_sem2_114m (run)
run:
Thread_A: count +1
        count= 1
Thread_A: count +1
        count= 2
Thread_B: count -1
        count= 1
Thread_B: count -1
        count= 0
Thread_A: count +1
        count= 1
Thread_B: count -1
        count= 0
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время)
```

Разработайте многопоточное приложение с использованием общей переменной. В каждом потоке должна быть объявлена локальная переменная, например, $i=0$. В ходе выполнения каждого потока, i должна увеличиваться, например, на 100. При каждом увеличении i , его значение должно выводиться на консоль.

Код программы:

```
public class Lab3_task2_sem2_114m {
    public static void main(String[] args) {
        //создание запуск и ожидание завершения потоков
        Thread first = new Thread(new TheFirstStream());
        Thread second = new Thread(new TheSecondStream());
        first.start();
        second.start();
        try {
            first.join();
            second.join();
        } catch (InterruptedException e) {}
    }
    //общий класс с общей переменной
    class Supportive {
        private static Supportive sup = new Supportive(); //экземпляр
        класса
        private int count = 0; //общая переменная
        //получение экземпляра класса
        public static Supportive getInstance() {
            return sup;
        }
        //получение значения общей переменной
        public int getCount() {
            return count;
        }
        //увеличение переменной и вывод
        public void increase(String nameThread) {
            synchronized (Supportive.class) {
```

```

        System.out.println(nameThread + " count = " + count++);
    }
    try {
        Thread.sleep(100);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
}
}
//первый поток
class TheFirstStream implements Runnable {
    @Override
    public void run() {
        Supportive sup = Supportive.getInstance();
        //пока значение общей переменной не достигнет 100
        while (sup.getCount() < 100) {
            sup.increase("Первый поток: ");
        }
    }
}
//второй поток
class TheSecondStream implements Runnable {
    @Override
    public void run() {
        Supportive sup = Supportive.getInstance();
        //пока значение общей переменной не достигнет 100
        while (sup.getCount() < 100) {
            sup.increase("Второй поток: ");
        }
    }
}
}
}
}

```

Вывод:

```

Вывод - Lab3_task2_sem2_114m (run)
Первый поток: count = 90
Первый поток: count = 91
Первый поток: count = 92
Первый поток: count = 93
Первый поток: count = 94
Первый поток: count = 95
Второй поток: count = 96
Второй поток: count = 97
Второй поток: count = 98
Второй поток: count = 99
Первый поток: count = 100
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 10 секунды)

```

Разработайте приложение, в котором один поток считывает текст из программы, а другой поток анализирует считанную строку на наличие образца и выводит строку на экран в зависимости от условий.

Код программы:

```

public class Lab3_task2_sem2_114m {
    public static void main(String[] args) {
        Supportive supr = new Supportive(); //вспомогательный класс
        //создание экземпляров классов
        TheFirstStream firstStream = new TheFirstStream(supr);
        TheSecondStream secondStream = new TheSecondStream(supr);
    }
}

```

```

        //запуск потоков
        new Thread(firstStream).start();
        new Thread(secondStream).start();
    }
    //вспомогательный класс
    static public class Supportive {
        int element = 0; //индекс текущий элемент списка
        //список строк, куда читается файл
        List<String> list = Collections.synchronizedList(new
ArrayList<String>());
        BufferedReader reader;
        boolean isEmpty = true;
        Supportive() {
            openFile();
        }
        //открытие файла текущей программы
        void openFile() {
            list.add(null);
            try {
                reader = new BufferedReader(new FileReader(new
File("D:/Users/Documents/NetBeansProjects/Magistratura/Lab3_task3_sem2_
_114m/src/lab3_task3_sem2_114m/Lab3_task3_sem2_114m.java")));
            } catch (FileNotFoundException ex) {}
        }
        void setElem(int i) { //передача текущего индекса
            element = i;
        }
        //чтение файла построчно и запись в список
        public synchronized void setVectorInt() throws IOException,
InterruptedException {
            String line;//читанная строка
            //текущая позиция вектора уже заполнена
            if (list.get(element) != null) { //ждем вывода строки
                wait();
            }
            //если файл не закончился, считываем строку и помещаем в
список
            if ((line = reader.readLine()) != null) {
                list.add(element, line);
            } else { //иначе сигнализируем, что конец файла
                isEmpty = false;
            }
            notify(); //продолжает работу потока, у которого ранее был
вызван метод wait()
        }
        //получение считанной строки, анализ вхождение образца и вывод
строки на экран.
        public synchronized void getVectorInt() throws
InterruptedException {
            if (list.get(element) == null) { //ждем чтения строки
                wait();
            }
            //если строка содержит заданное слово, она выводится
            if(isNotEmpty!=false &&
list.get(element).contains("java")){
                System.out.println(list.get(element));
            }
            notify(); //продолжает работу потока

```

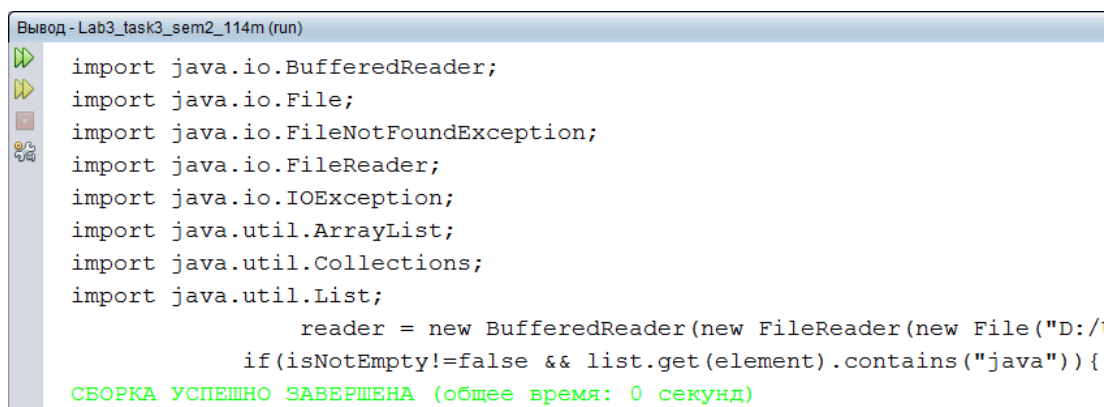
```

    }
}
//класс первого потока реализует интерфейс Runnable
static class TheFirstStream implements Runnable {
    Supportive supr;
    TheFirstStream(Supportive supr) {
        this.supr = supr;
    }
    @Override
    public void run() {

        while(supr.isNotEmpty!=false) {
            try { //чтение файла
                supr.setVectorInt();
            } catch (IOException ex) {}
            catch (InterruptedException ex) {}
        }
    }
}
//класс второго потока выводит считанную строку с условием
static class TheSecondStream implements Runnable {
    Supportive supr;
    TheSecondStream(Supportive supr) {
        this.supr = supr;
    }
    @Override
    public void run() {
        int index=1;//индекс текущей позиции
        while(supr.isNotEmpty!=false) {
            try {
                supr.getVectorInt();//вывод считанной строки
            } catch (InterruptedException ex) {}
            supr.setElem(index);
            index++;
        }
    }
}
}

```

Вывод:



```

import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

        reader = new BufferedReader(new FileReader(new File("D:/
        if(isNotEmpty!=false && list.get(element).contains("java")){
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 0 секунд)

```

Разработайте приложение, которое имитирует кольцевой буфер в задаче "производитель - потребитель". Программа должна создавать $K > 1$ процессов "производителей" и $M > 1$ процессов "потребителей". Буфер должен иметь размер N элементов. Средняя частота работы

производителя должна быть в 10 раз меньше, чем у потребителя. Частоты работы должны быть установлены случайным образом. Производитель должен записывать в буфер в среднем 10 значений. **Код программы:**

```
public class Lab3_task4_sem2_114m {

    public static void main(String[] args) {
        int size = 10; //размер буфера
        Random r = new Random();
        int rand = r.nextInt(1000); //случайная частота
        Buffer buff = new Buffer(size); //буфер
        //создание и запуск потоков производителя и покупателя
        Thread producer = new Thread(new Producer(buff, size, rand));
        Thread consumer = new Thread(new Consumer(buff, size, rand));
        producer.start();
        consumer.start();
    }
    //реализация кольцевого буфера
    public static class Buffer {
        String a[];
        int first, last;
        public Buffer(int size) {
            a = new String[size];
            first = last = -1;
        }
        //метод добавление в буфер
        public synchronized boolean add(String element) {
            int pop;
            pop = (last + 1) % a.length;
            //буфер заполнен
            if (pop == first) {
                return false;
            } else {
                last = pop;
                a[last] = element;
                if (first == -1) {
                    first = 0;
                }
                return true;
            }
        }
        //буфер пустой
        public boolean empty() {
            return first == -1;
        }
        //удаление элемента
        public synchronized String delete() {
            String result = a[first];
            if (first == last) {
                first = last = -1;
            } else {
                first = (first + 1) % a.length;
            }
            return result;
        }
    }
    //поток производителя
    static class Producer extends Thread {
```

```

Buffer buff;
int size;
long slep;
public Producer(Buffer buff, int size, long slep) {
    this.buff = buff;
    this.size = size;
    this.slep = slep;
}
//добавление элемента
public void run() {
    int i = 0;
    while (true) {
        try {
            String element = "элемента" + "_" + ++i;
            boolean bool = buff.add(element);
            if (bool) {
                System.out.println("Производство " + element);
                Thread.sleep(this.slep);
            }
            synchronized (buff) {
                buff.notifyAll();
            }
        } catch (Exception e) { e.printStackTrace();}
    }
}
//поток потребителя
static class Consumer extends Thread {
    Buffer buff;
    int size;
    long slep;
    public Consumer(Buffer buff, int size, long slep) {
        this.buff = buff;
        this.size = size;
        this.slep = slep;
    }
    //потребление элемента
    public void run() {
        while (true) {
            while (buff.empty()) {
                synchronized (buff) {
                    try {
                        buff.wait();
                    } catch (Exception e) { e.printStackTrace();}
                }
            }
            String element = null;
            synchronized (buff) {
                element = buff.delete();
                try {
                    Thread.sleep(this.slep);
                } catch (InterruptedException ex) {}
            }
            System.out.println("Потребление " + element);
        }
    }
}

```

Вывод:

Вывод - Lab3_task4_sem2_114m (run)



run:

Производство элемента_1
Производство элемента_2
Производство элемента_3
Потребление элемента_1
Потребление элемента_2
Производство элемента_4
Потребление элемента_3
Производство элемента_5
Потребление элемента_4
Производство элемента_6
Потребление элемента_5
Потребление элемента_6
Производство элемента_7
Потребление элемента_7