

Задача 1 заключается в изучении основных принципов создания многопоточных приложений в рамках написания тестовых заданий.

Необходимо создать поток двумя способами:

Путем расширения класса Thread;

Путем переопределения метода run() и реализации интерфейса Runnable.

Если класс расширяет класс Thread, поток можно запустить, создав экземпляр класса и вызвав его метод start(). Предоставьте пример расширения.

Если класс реализует интерфейс Runnable, поток можно запустить, передав экземпляр класса в конструктор объекта Thread и вызвав метод start(). Предоставьте пример реализации.

Код программы:

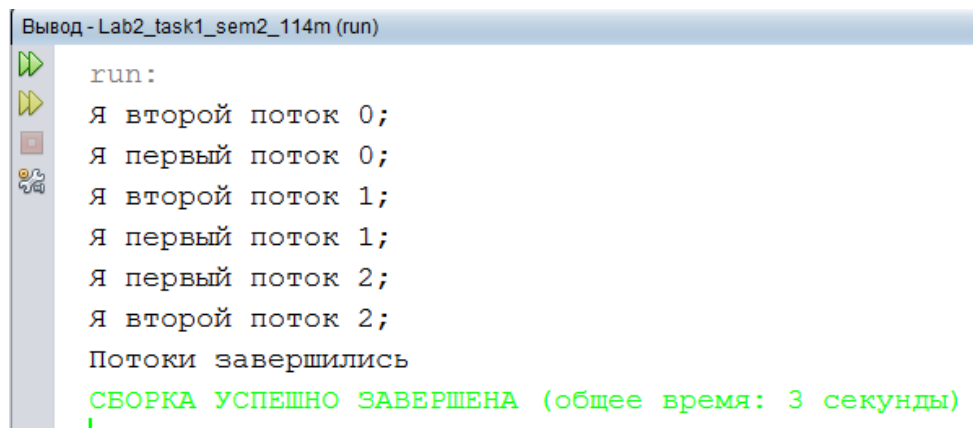
```
public class Lab2_task1_sem2_114m {
    public static void main(String[] args) {
        //создание экземпляров классов
        TheFirstStream first = new TheFirstStream();
        TheSecondStream Second= new TheSecondStream();
        //передача экземпляра класса конструктору Thread-объекта
        Thread secondTread = new Thread(Second);
        first.start(); // Запуск первого потока
        secondTread.start(); //запуск второго потока
        try {
            first.join(); //ждем завершения 1го потока
            secondTread.join();//ждем завершения 2го потока
        } catch (InterruptedException e){}
        System.out.println("Потоки завершились");
    }
}
//класс TheFirstStream расширяет класс Thread
class TheFirstStream extends Thread {
    @Override
    //в цикле выводится строка и кол-во итераций через
    промежутки 1с.
    public void run() {
        for (int i = 0; i < 3; i++) {
            System.out.println("Я первый поток " + i + ";");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}
```

```

//класс TheSecondStream реализует интерфейс Runnable
class TheSecondStream implements Runnable {
    @Override
    public void run() {
        //в цикле выводится строка и кол-во итераций через
        //промежутки 1с.
        for (int i = 0; i < 3; i++) {
            System.out.println("Я второй поток " + i + ";");
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

```

Вывод:



```

Вывод - Lab2_task1_sem2_114m (run)
run:
Я второй поток 0;
Я первый поток 0;
Я второй поток 1;
Я первый поток 1;
Я первый поток 2;
Я второй поток 2;
Потоки завершились
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 3 секунды)

```

Задача 2. Создайте два класса потоков (наследники класса Thread), взаимодействующих с использованием промежуточного объекта типа Vector.

Первый поток последовательно заполняет вектор (изначально заполненный нулями) произвольными различными значениями (например, случайными), отличными от нуля. При каждом добавлении значения в вектор он выводит на экран сообщение вида "Write: 100.5 to position 3". Когда достигается конец вектора, поток завершает свое выполнение.

Второй поток последовательно считывает значения из вектора и выводит их на экран с сообщениями вида "Read: 100.5 from position 3". По достижении конца вектора поток завершает свое выполнение.

В методе main() необходимо создать 3 участвующих объекта и запустить потоки на выполнение. Запускайте программу несколько раз. Попробуйте варьировать приоритеты потоков.

Код программы:

```

public class Lab2_task2_sem2_114m {

```

```

static int sizeVector; //размер вектора
static Vector coolVector = new Vector(5); //вектор

//запись в вектор поэлементно случайных чисел
public static void setVectorInt(int i) {
    final Random random = new Random();
    int rm = random.nextInt(99) + 1;
    coolVector.add(i, rm); //в позицию вектора i случ.число
rm
    System.out.println("Write: " + rm + " to position " +
i);
}
//чтение вектора поэлементно
public static void getVectorInt(int i) {
    System.out.println(" Read: " + coolVector.get(i) + "
from position " + i);
}
//заполнение исходного вектора нулями
public static void setVectorZeros() {
    for (int i = 0; i < coolVector.capacity();
i++) coolVector.add(0);
    sizeVector = coolVector.size();
}
public static void main(String[] args) {
    setVectorZeros(); //заполнение вектора нулями
    //создание экземпляров классов
    TheFirstStream first = new TheFirstStream();
    TheSecondStream second = new TheSecondStream();
    //расставление приоритетов
    first.setPriority(Thread.MAX_PRIORITY);
    second.setPriority(Thread.MIN_PRIORITY);
    first.start(); // Запуск первого потока
    second.start(); //запуск второго потока
    try {
        first.join(); //ждем завершения 1го потока
        System.out.println("Первый поток завершился");
        second.join(); //ждем завершения 2го потока
        System.out.println("Второй поток завершился");
    } catch (InterruptedException e) {
    }

    }
}
//класс первого потока наследует от класса Thread
class TheFirstStream extends Thread {
    @Override
    public void run() {
        //заполняет вектор случ.числами и выводит их
        for (int i = 0; i < Lab2_task2_sem2_114m.sizeVector;
i++) {
            Lab2_task2_sem2_114m.setVectorInt(i);
            //приостановка потока

```

```

        try {
            Thread.sleep(500);
        } catch (InterruptedException e) {
            System.out.println(e);
        }
    }
}

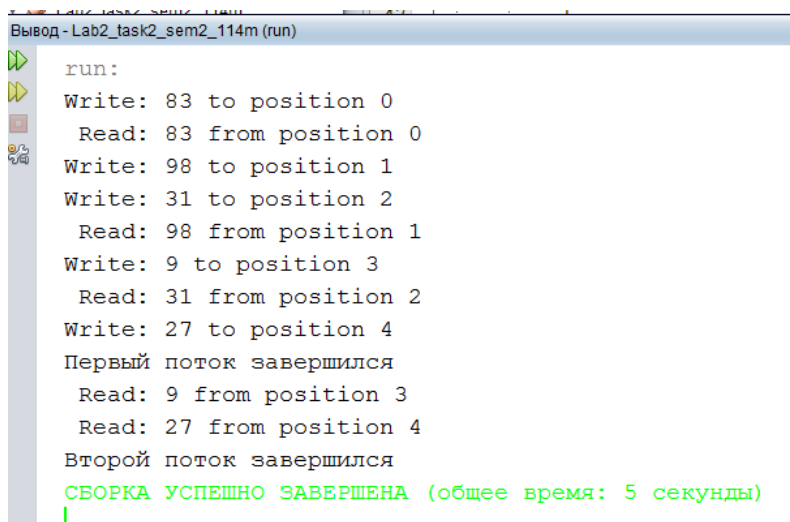
//класс второго потока наследует от класса Thread
class TheSecondStream extends Thread {
    @Override
    public void run() {
        //чтение вектора поэлементно и вывод
        for (int i = 0; i < Lab2_task2_sem2_114m.sizeVector;
i++) {
            Lab2_task2_sem2_114m.getVectorInt(i);
            //приостановка потока
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

```

Вывод:

Приоритет первого потока: MAX_PRIORITY;

Приоритет второго потока: MIN_PRIORITY.



```

run:
Write: 83 to position 0
Read: 83 from position 0
Write: 98 to position 1
Write: 31 to position 2
Read: 98 from position 1
Write: 9 to position 3
Read: 31 from position 2
Write: 27 to position 4
Первый поток завершился
Read: 9 from position 3
Read: 27 from position 4
Второй поток завершился
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 5 секунды)

```

Приоритет первого потока: MIN_PRIORITY;

Приоритет второго потока: MAX_PRIORITY.

```
Lab2_task2_sem2_114m
Вывод - Lab2_task2_sem2_114m (run)

run:
Read: 0 from position 0
Write: 89 to position 0
Read: 0 from position 1
Write: 78 to position 1
Read: 0 from position 2
Write: 73 to position 2
Read: 0 from position 3
Write: 82 to position 3
Read: 0 from position 4
Write: 1 to position 4
Второй поток завершился
Первый поток завершился
СВОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 5 секунды)
```

Задача 3. Создайте два новых модифицированных класса потоков (реализующих интерфейс Runnable), которые обеспечат последовательность операций чтения-записи (т.е. сообщения будут выводиться в порядке записи-чтения-записи-чтения-...) независимо от приоритетов потоков. Для этого потребуется создать вспомогательный класс, объект которого будет использоваться при взаимодействии потоков.

Код программы:

```
public class Lab2_task3_sem2_114m {

    public static void main(String[] args) {
        Supportive supr = new Supportive(); //вспомогательный
класс
        supr.setVectorZeros(); //заполнение вектора нулями
        //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 вектора
        static int sizeVector; //размер вектора
        static Vector coolVector = new Vector(5); //вектор

        void setElem(int i) { //передача текущего индекса
            element = i;
        }
        //заполнение исходного вектора нулями
        public static void setVectorZeros() {
```

```

        for (int i = 0; i < coolVector.capacity(); i++) {
            coolVector.add(0);
        }
        sizeVector = coolVector.size();
    }
    //запись в вектор поэлементно случайных чисел
    public synchronized void setVectorInt() {
        if ((int) coolVector.get(element) != 0) { //текущая
позиция вектора уже заполнена
            try { //ждем чтения текущей позиции
                wait();
            } catch (InterruptedException e) {}
        }
        //если текущая позиция содержит ноль, заполняем ее
случ.значением
        final Random random = new Random();
        int rm = random.nextInt(99) + 1;
        coolVector.add(element, rm); //в позицию вектора
element случ.число rm
        System.out.println("Write: " + rm + " to position "
+ element);
        notify(); //продолжает работу потока, у которого
ранее был вызван метод wait()
    }
    //чтение вектора поэлементно
    public synchronized void getVectorInt() {
        if ((int) coolVector.get(element) == 0) { //текущая
позиция вектора еще не заполнена
            try { //ждем записи в текущую позицию
                wait();
            } catch (InterruptedException e) {}
        }
        //если текущая позиция соеджит ненулевое значение,
считываем
        System.out.println(" Read: " +
coolVector.get(element) + " from position " + 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() {
        //заполняет вектор случ.числами и выводит их
        for (int i = 0; i < supr.sizeVector; i++) {
            supr.setVectorInt();
        }
    }
}

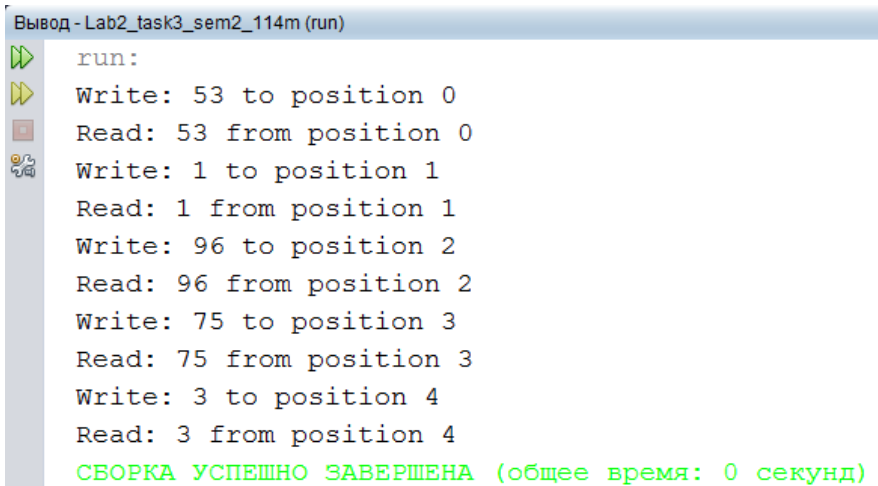
```

```

    }
    //класс второго потока реализует интерфейс Runnable
    static class TheSecondStream implements Runnable {
        Supportive supr;
        TheSecondStream(Supportive supr) {
            this.supr = supr;
        }
        @Override
        public void run() {
            for (int i = 0; i < supr.sizeVector; i++) {
                //чтение вектора поэлементно и вывод
                supr.getVectorInt();
                supr.setElem(i);
            }
        }
    }
}

```

Вывод:



```

Вывод - Lab2_task3_sem2_114m (run)
run:
Write: 53 to position 0
Read: 53 from position 0
Write: 1 to position 1
Read: 1 from position 1
Write: 96 to position 2
Read: 96 from position 2
Write: 75 to position 3
Read: 75 from position 3
Write: 3 to position 4
Read: 3 from position 4
СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 0 секунд)

```

Задача 4. В класс с методами обработки векторов добавьте метод `synchronizedVector(Vector vector)`, который будет возвращать безопасную с точки зрения многопоточности оболочку указанного вектора. Для этого необходимо создать новый класс, который будет реализовывать интерфейс `Vector` и перегружать методы класса `Object`. **Код программы:**

```

class CoolVector extends Vector {
    protected Vector vector;
    protected ReentrantLock lock;

    public CoolVector(Vector vector) { //конструктор
        this.vector = vector;
        this.lock = new ReentrantLock();
    }
    @Override // Получение элемента по индексу
    public Object get(int i) throws
        ArrayIndexOutOfBoundsException {

```

```

        lock.lock();
        int x = (int) this.vector.get(i);
        lock.unlock();
        return x;
    }
    @Override //Получение кол-ва элементов вектора
    public int size() {
        lock.lock();
        int size = this.vector.size();
        lock.unlock();
        return size;
    }
    @Override //Получение длины вектора
    public int capacity() {
        lock.lock();
        int capacity = this.vector.capacity();
        lock.unlock();
        return capacity;
    }
    @Override //Добавление элемента
    public boolean add(Object x) {
        lock.lock();
        boolean add = this.vector.add(x);
        lock.unlock();
        return add;
    }
    //Добавление элемента по индексу
    @Override
    public void add(int index, Object x) {
        lock.lock();
        this.vector.add(index, x);
        lock.unlock();
    }
}

public class Lab2_task4_sem2_114m {

    static int sizeVector; //размер вектора
    //возвращает ссылку на оболочку указанного вектора
    public static Vector synchronizedVector(Vector vector) {
        return new CoolVector(vector);
    }
    static Vector coolVector = synchronizedVector(new
Vector(5));
    //запись в вектор поэлементно случайных чисел
    public static void setVectorInt(int i) {
        final Random random = new Random();
        int rm = random.nextInt(99) + 1;
        coolVector.add(i, rm); //в позицию вектора i случ.число
rm
        System.out.println("Write: " + rm + " to position " +
i);
    }
}

```



```

    }
    //чтение вектора поэлементно
    public static void getVectorInt(int i) {
        System.out.println(" Read: " + coolVector.get(i) + "
from position " + i);
    }
    //заполнение исходного вектора нулями
    public static void setVectorZeros() {
        for (int i = 0; i < coolVector.capacity(); i++) {
            coolVector.add(0);
        }
        sizeVector = coolVector.size();
    }

    public static void main(String[] args) {

        setVectorZeros(); //заполнение вектора нулями
        //создание экземпляров классов
        TheFirstStream first = new TheFirstStream();
        TheSecondStream second = new TheSecondStream();
        first.start(); // Запуск первого потока
        second.start(); //запуск второго потока
        try {
            first.join(); //ждем завершения 1го потока
            System.out.println("Первый поток завершился");
            second.join(); //ждем завершения 2го потока
            System.out.println("Второй поток завершился");
        } catch (InterruptedException e) {}
    }
}
//класс первого потока наследует от класса Thread
class TheFirstStream extends Thread {
    @Override
    public void run() {
        //заполняет вектор случ. числами и выводит их
        for (int i = 0; i < Lab2_task4_sem2_114m.sizeVector;
i++) {
            Lab2_task4_sem2_114m.setVectorInt(i);
            //приостановка потока
            try {
                Thread.sleep(500);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}
//класс второго потока наследует от класса Thread
class TheSecondStream extends Thread {
    @Override
    public void run() {
        //чтение вектора поэлементно и вывод

```

```

        for (int i = 0; i < Lab2_task4_sem2_114m.sizeVector;
i++) {
            Lab2_task4_sem2_114m.getVectorInt(i);
            //приостановка потока
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

```

Вывод:

Вывод - Lab2_task4_sem2_114m (run)

```

run:
  Read: 0 from position 0
  Write: 17 to position 0
  Write: 9 to position 1
  Write: 91 to position 2
  Read: 9 from position 1
  Write: 99 to position 3
  Read: 91 from position 2
  Write: 32 to position 4
  Первый поток завершился
  Read: 99 from position 3
  Read: 32 from position 4
  Второй поток завершился
  СБОРКА УСПЕШНО ЗАВЕРШЕНА (общее время: 5 секунды)

```