Оглавление

Лабораторная работа №1. Создание потоков в Java.

В языке Java изначально заложена возможность создания многопоточных приложений. Для этого используются следующие варианты: реализация собственного класса, унаследовав его от класса Thread, или реализация собственного класса с интерфейсом Runnable. Рассмотрим оба способа на простом примере.

Задача: Написать класс, объект которого при запуске в отдельном потоке будет выводить в консоль нужную строку символов.

Вариант 1 – наследование от класса Thread.

```
public class SimpleThread extends Thread {
	private String s;
	public SimpleThread (String str){
		s=str;
	}
	@Override
	public void run(){
	    while (this.getState()==Thread.State.RUNNABLE){
	        System.out.println(s);
				try {
					this.sleep(500);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
	    }
	}
}
```

Вариант 2 – реализация интерфейса Runnable.

```
public class SimpleRunnable implements Runnable {

	private String s;
	public SimpleRunnable (String str){
		s=str;
	}
	@Override
	public void run(){
	    while (Thread.currentThread().getState()==Thread.State.RUNNABLE){
	        System.out.println(s);
				try {
					Thread.currentThread().sleep(500);
				} catch (InterruptedException e) {
					e.printStackTrace();
				}
	    }
	}
}
```

А в другом классе покажем использование объектов обоих классов.

```
public class MainThread {

	public static void main(String[] args) {
		SimpleThread thread1=new SimpleThread("A");
		thread1.start();
		Thread thread2=new Thread(new SimpleRunnable("B"));
		thread2.start();
	}
}
```

После запуска программы в консоль будут выводиться буквы A и B от разных потоков. Для остановки программы потребуется принудительно прервать программу средствами Eclipse.

Реализация интерфейса Runnable используется в случаях, когда класс уже наследует какой-либо родительский класс, и тем самым не позволяет расширить класс Thread. Реализация интерфейсов считается хорошим тоном программирования в java. Это связано с тем, что в java может наследоваться только один родительский класс, таким образом, унаследовав класс Thread, нельзя наследовать какой-либо другой класс.

Расширение класса Thread целесообразно использовать, когда необходимо переопределить другие методы класса Thread, помимо метода run(). Но это используется довольно редко.

Задание

Создать класс, реализующий интерфейс Runnable. Объекты класса должны характеризоваться уникальным номером, именем и интервалом задержки.

В методе run() должна в бесконечном цикле выводиться с интервалом задержки информация: номер объекта, любой текст, имя объекта. В методе main() создать и стартовать потоки для нескольких объектов. Выполнить прерывание потоков методом interrupt() из метода main() после некоторого интервала времени. Для этого можно воспользоваться следующим шаблоном:

```
Date currentDate=new Date();
long time1=currentDate.getTime();

… //действия, длительность которых замеряется

currentDate=new Date();
long time2=currentDate.getTime();;
while ((time2-time1)<2000) { //цикл, пока разница не составит 2 секунды
      currentDate=new Date();
      time2=currentDate.getTime();
}
thread1.interrupt(); //прерываем потоки
thread2.interrupt();
```

Но в этом случае есть риск нарваться на исключение вот в этом участке кода из `SimpleRunnable`:

```
try {
      Thread.currentThread().sleep(500);
} catch (InterruptedException e) { //исключение сработает, если попытаться прервать
      e.printStackTrace(); // этот поток через interrupt
}
```

Поэтому меняем его на

```
try {
      Thread.currentThread().sleep(500);
} catch (InterruptedException e) {
      System.out.println(s+" exit"); //печатаем сообщение и выходим из потока
      return;
}
```

Лабораторная работа №2. Потоки и GUI

При работе с потоками с использованием графического интерфейса появляются определенные сложности. Например, если несколько потоков одновременно пытаются произвести вывод информации в GUI-элемент, то может произойти ошибка, т.к. GUI – однопоточно, т.е. только 1 поток может работать с GUI в единицу времени.

Если несколько потоков будут работать с одним объектом (вызывать методы), то потребуется синхронизация потоков. Каждый класс Java имеет ассоциированный с ним ***монитор***, поэтому метод, объявленный с ключевым словом synchronized, становится синхронизированным, и если один поток вызовет этот метод, то остальные потоки не смогут вызывать ни один синхронизированный метод данного объекта, пока первый поток не выйдет из данного метода.

Монитор, в понимании параллельного программирования, инкапсулирует представление абстрактного объекта, разделяемого несколькими процессами, и обеспечивает набор операций для доступа к нему. Вся синхронизация доступа обеспечивается самим монитором.

Говоря простым языком, ***монитор*** – это такой объект, который следит за другим объектом (объектами), доступ к которым хотят получить несколько потоков. Процесс, перед использованием такого объекта, должен сначала "попросить разрешения" у монитора. Если объект никем не используется, то монитор даст разрешение.

Дополнительно можно синхронизироваться на определенных объектах, определяя блоки с ключевым словом synchronized.

Например, переделаем пример из прошлой лабораторной. Сделаем вывод сообщений от потоков не в консоль, а в графический интерфейс. Также сделаем кнопки для старта потока, паузы и продолжения. И предусмотрим одновременную работу с двумя объектами-потоками.

Сначала создадим поточный класс, реализующий интерфейс Runnable. Объекты этого класса смогут выводить данные прямо в JTextArea (будет использоваться синхронизация):

```
import javax.swing.JTextArea;

class SimpleRunnable implements Runnable{

 private boolean suspendFlag = false;
 private JTextArea ta; //локальный объект типа JTextArea
 private String text;  //текст, который поток будет выводить

        public SimpleRunnable(JTextArea ta, String textOut){
//при создании объектов в главном классе будет вызываться данный конструктор
//и передаваться через параметр ссылка на объект JTextArea из интерфейса пользователя
               this.ta = ta; //локальный объект присваиваем переданному параметру
               text=textOut;
        }

        @Override
        public void run() {
               while(true){
                      ta.append(text+"\n"); //выводим текст в объект JTextArea
                      try {
                            Thread.currentThread().sleep(700);
                     } catch (InterruptedException e) {
                            e.printStackTrace();
                     }
                      //код проверки на останов потока в синхронизированном контексте
                      synchronized(this){
                             //если suspendFlag установлен - прекращаем действие потока
                             while(suspendFlag) {try {
                                    wait(); //приостанавливает поток
                             } catch (InterruptedException ex) {
                                    ta.append("Произошла ошибка в потоке\n");
                             }
                            }
                      }
               }
        }

        synchronized void suspendThread(){ //синхронизированный метод для приостановки потока
//для остановки потока достаточно взвести флаг - в коде потока он проверится и вызовется метод wait()
//данный метод будем вызывать из главного класса
               suspendFlag = true;
               ta.append("Остановка потока "+text+"\n");
        }

       synchronized void resumeThread(){//синхронизированный метод для возобновления потока
//данный метод будем вызывать из главного класса
               suspendFlag = false; //сбрасываем флаг
               ta.append("Запуск потока"+text+"\n");
               notify(); //запускаем поток, который был остановлен (то есть сам себя)
        }
}
```

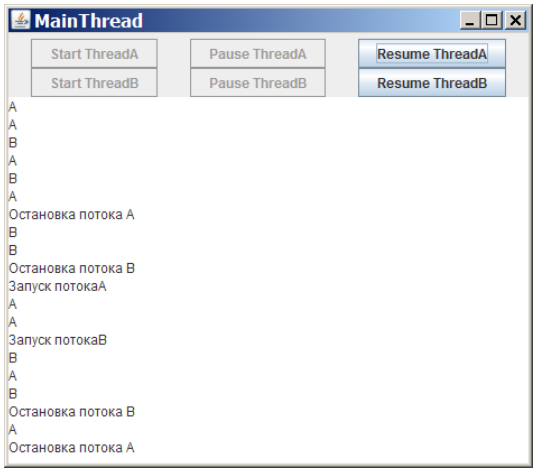
Теперь можно использовать объекты созданного класса в основной программе, т.е. создаем главный класс проекта:

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.Box;
import javax.swing.BoxLayout;
```

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JTextArea;

public class TestThread extends JFrame {

	private JButton startThreadA; //кнопки для управления первым потоком
	private JButton pauseThreadA;
	private JButton resumeThreadA;

	private JButton startThreadB; //кнопки для управления вторым потоком
	private JButton pauseThreadB;
	private JButton resumeThreadB;

	private JTextArea myTextArea; //текстовая область, в нее пишут потоки
	private Thread t1;  //первый поток
	private SimpleRunnable run1; //первый объект нашего созданного класса

	private Thread t2; //второй поток
	private SimpleRunnable run2; //второй объект нашего созданного класса

	public TestThread(String name){ //конструктор, строим интерфейс пользователя
		super(name);
		setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		myTextArea=new JTextArea();
		add(myTextArea,BorderLayout.CENTER); //вставляем текстовую область в центр окна

		Box yBox=new Box(BoxLayout.Y_AXIS); //создаем контейнер для панелей с кнопками
		yBox.add(Box.createVerticalStrut(5)); //распорка
		yBox.add(setToolBarA()); //панель с кнопками для управления первым потоком
		yBox.add(Box.createVerticalGlue()); //пружина
		yBox.add(setToolBarB()); //панель с кнопками для управления вторым потоком
		yBox.add(Box.createHorizontalStrut(5)); //распорка

		add(yBox,BorderLayout.NORTH); //вставляем контейнер в верхнюю часть окна
		setSize(500, 300); //размер окна
		setVisible(true); //делаем окно видимым
		setMinimumSize(getSize()); //минимальный размер окна
	}

	private Box setToolBarA(){ //метод, возвращает панель с кнопками управления первым потоком
		Box toolBar; //контейнер для кнопок
		startThreadA=new JButton("Start ThreadA"); //кнопка для запуска потока
		startThreadA.addActionListener(new ActionListener() { //обработчик кнопки

			@Override
			public void actionPerformed(ActionEvent arg0) {
				startThreadA.setEnabled(false); //делаем кнопку запуска неактивной,
				//чтобы не было возможности еще раз запустить уже запущенный поток
				//т.к. это приведет к ошибке
				run1=new SimpleRunnable(myTextArea,"A");//создаем объект нашего класса
				//и передаем ему текстовую область и строку для вывода
				t1=new Thread(run1); //создаем поток и передаем ему наш объект
				t1.start(); //стартуем поток
				pauseThreadA.setEnabled(true); //делаем активной кнопку паузы
			}
		});

		pauseThreadA=new JButton("Pause ThreadA");//кнопка для приостановки потока
		pauseThreadA.setEnabled(false); //делаем ее неактивной, т.к. если поток еще не
		//запущен, то и приостанавливать пока нечего
		pauseThreadA.addActionListener(new ActionListener() {//обработчик кнопки

			@Override
			public void actionPerformed(ActionEvent arg0) {
//вызываем нами созданный метод для приостановки потока (см. класс SimpleRunnable выше)
				run1.suspendThread();
				resumeThreadA.setEnabled(true); //делаем активной кнопку возобновления
				pauseThreadA.setEnabled(false); //делаем неактивной кнопку паузы
			}
		});

		resumeThreadA=new JButton("Resume ThreadA"); //кнопка для возобновления потока
		resumeThreadA.setEnabled(false); //делаем ее неактивной,
		//т.к. в начале работы программы еще ничего не нужно возобновлять
		resumeThreadA.addActionListener(new ActionListener() {//обработчик кнопки
```

```
                    @Override
                    public void actionPerformed(ActionEvent arg0) {
//вызываем нами созданный метод для возобновления потока (см. класс SimpleRunnable выше)
                            run1.resumeThread();
                            //делаем неактивной кнопку возобновления
                            resumeThreadA.setEnabled(false);
                            pauseThreadA.setEnabled(true); //делаем активной кнопку паузы
                    }
            });

            toolBar=new Box(BoxLayout.X_AXIS); //создаем контейнер
            toolBar.add(Box.createHorizontalStrut(20)); // распорка
            toolBar.add(startThreadA);  //вставляем кнопку в контейнер
            toolBar.add(Box.createHorizontalGlue()); //пружина
            toolBar.add(pauseThreadA); //вставляем кнопку в контейнер
            toolBar.add(Box.createHorizontalGlue());//пружина
            toolBar.add(resumeThreadA); //вставляем кнопку в контейнер
            toolBar.add(Box.createHorizontalStrut(20)); //распорка
            return toolBar; //возвращаем контейнер с кнопками
      }

      private Box setToolBarB(){//метод, возвращает панель с кнопками управления вторым потоком
      //все, как в предыдущем методе, но для других кнопок и для run2 и t2
            Box toolBar;
            startThreadB=new JButton("Start ThreadB");
            startThreadB.addActionListener(new ActionListener() {

                  @Override
                  public void actionPerformed(ActionEvent arg0) {
                        startThreadB.setEnabled(false);
                        run2=new SimpleRunnable(myTextArea,"B");
                        t2=new Thread(run2);
                        t2.start();
                        pauseThreadB.setEnabled(true);
                  }
            });

            pauseThreadB=new JButton("Pause ThreadB");
            pauseThreadB.setEnabled(false);
            pauseThreadB.addActionListener(new ActionListener() {

                  @Override
                  public void actionPerformed(ActionEvent arg0) {
                        run2.suspendThread();
                        resumeThreadB.setEnabled(true);
                        pauseThreadB.setEnabled(false);
                  }
            });

            resumeThreadB=new JButton("Resume ThreadB");
            resumeThreadB.setEnabled(false);
            resumeThreadB.addActionListener(new ActionListener() {

                  @Override
                  public void actionPerformed(ActionEvent arg0) {
                        run2.resumeThread();
                        resumeThreadB.setEnabled(false);
                        pauseThreadB.setEnabled(true);
                  }
            });

            toolBar=new Box(BoxLayout.X_AXIS);
            toolBar.add(Box.createHorizontalStrut(20));
            toolBar.add(startThreadB);
            toolBar.add(Box.createHorizontalGlue());
            toolBar.add(pauseThreadB);
            toolBar.add(Box.createHorizontalGlue());
            toolBar.add(resumeThreadB);
            toolBar.add(Box.createHorizontalStrut(20));
            return toolBar;
      }

      public static void main(String[] args) {
            new TestThread("MainThread"); //создаем окно
      }
}
```

Внешний вид получившейся программы примерно следующий:



Задания

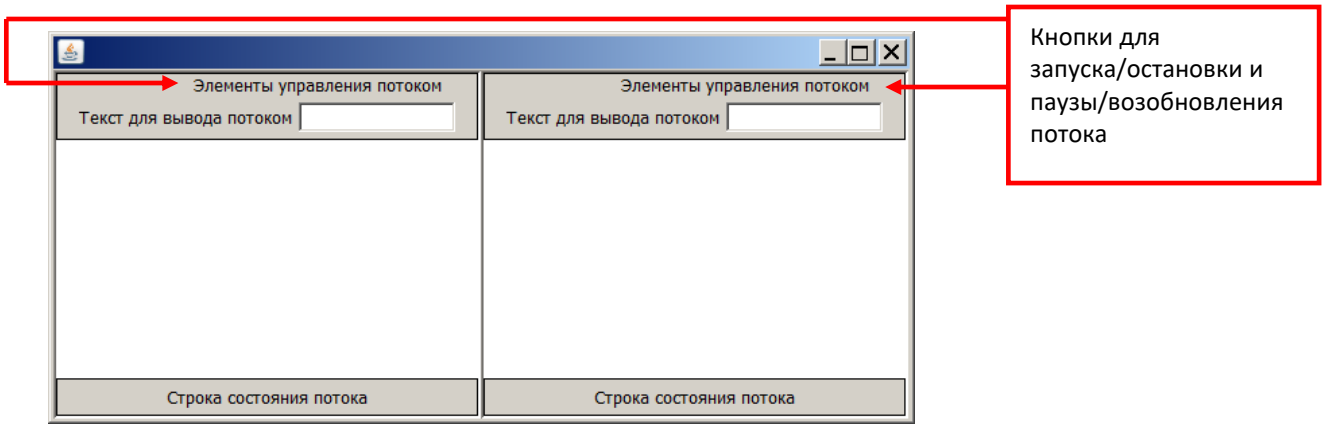
1. Поменять программу из приведенного примера: убрать у функций ключевое слово `synchronized` и проследить за изменениями в программе.

2. Поменять внешний вид программы из приведенного примера: вместо кнопок Pause Thread и Resume Thread сделать переключатели (JRadioButton) (см. пример для 1 потока на рисунке).

3. Поменять программу еще раз: вместо кнопок Pause Thread и Resume Thread сделать один JCheckBox, меняющий активность потока (см. пример для 1 потока на рисунке)

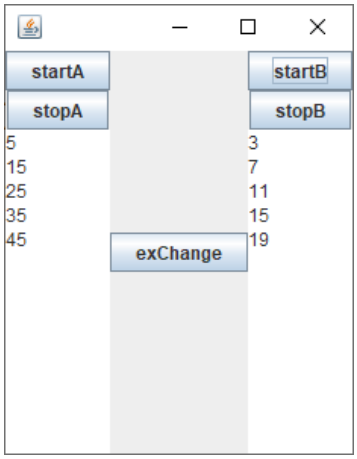
4. Добавить возможность прерывания потока по нажатию специальной кнопки (см. применение метода interrupt из лабораторной №1).

5. Создать новую программу на основе предыдущих 4 задач: разделить окно программы на 2 части (чтобы отдельно работать с каждым потоком). У каждого потока свои управляющие элементы. Предусмотреть возможность ввода пользователем информации, которую выводит поток. Каждый поток должен выводить информацию только в своей части окна. При этом состояние каждого потока (работает/приостановлен/прерван) выводится в отдельную строку состояния (у каждого своя). Выглядеть все должно примерно как рисунке:

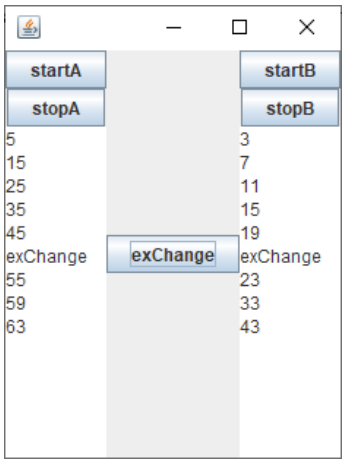


Лабораторная работа №3. Передача сообщений между потоками

Самый простой способ передачи данных между потоками – вызов соответствующих методов нужных объектов, представляющих потоки. Например, создадим 2 потока, которые будут в автономном режиме увеличивать начальное значение на некое число (у каждого потока свое). По нажатию на кнопку они этими числами меняются и продолжают работу уже исходя из новых значений (рис. 1). Главный поток приложения выступает в роли связующего звена.



а)



б)

Рис. 1. Работа потоков:

а) после запуска – поток А начал с 5 и увеличивает на 10, поток Б начал с 3 и увеличивает на 4;

б) после нажатия кнопки exChange – потоки обменялись числами и теперь поток А увеличивает на 4, поток Б увеличивает на 10.

Код для потокового класса:

```
public class MyThread extends Thread {
        private JTextArea ta; // локальный объект типа JTextArea
        private int x; // начальное число
        private int dx; // число, на которое увеличивать

        public MyThread(int x, int dx) {  //конструктор
                this.x = x;
                this.dx = dx;
        }

        @Override
        public void run() {
                while (true) {
                        ta.append(x + "\n"); // выводим текст в объект JTextArea
                        x += dx;         //увеличиваем x на dx
                        try {
                                Thread.currentThread().sleep(1000);
                        } catch (InterruptedException e) {
                                System.out.println(" exit");
                                return;
                        }
                }
        }
// геттеры и сеттеры для всех полей класса
        public int getDx() {
                return dx;
        }

        public void setDx(int dx) {
                this.dx = dx;
                ta.append("exChange\n");        //для обозначения, что произошел обмен данными
        }

        public JTextArea getTa() {
                return ta;
        }

        public void setTa(JTextArea ta) {
                this.ta = ta;
        }
}
```

Код класса с графическим интерфейсом:

```
public class MainClass {
        static MyThread threadA; //объекты-потоки
        static MyThread threadB;
```

```
	public static void main(String[] args) {
		createGUI();   //вызов ф-ии, которая создает GUI
	}

	private static void createGUI() {
		JFrame myFrame = new JFrame();  // создаем окно
		myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		Box centralBox = new Box(BoxLayout.X_AXIS); //создаем коробку для центральной области
		threadA = new MyThread(5, 10);  //создаем потоковые объекты с нужными параметрами
		threadB = new MyThread(3, 4);
		Box leftBox = createBox("A", threadA); //вызов ф-ии для создания областей для каждого потока
		Box rightBox = createBox("B", threadB);

		JButton chahgeButton = new JButton("exChange"); //объекты для обмена значениями
		chahgeButton.addActionListener(l->{ //слушатель нажатия кнопки
			int dx=threadA.getDx();  //меняем значения dx в потоках
			threadA.setDx(threadB.getDx());
			threadB.setDx(dx);
		});
		centralBox.add(leftBox); //вставляем области для каждого потока и кнопку обмена в центральный бокс
		centralBox.add(chahgeButton);
		centralBox.add(rightBox);

		myFrame.add(centralBox, BorderLayout.CENTER); //вставляем центральный бокс в центр окна
		myFrame.setSize(300, 400);  // размер окна
		myFrame.setLocationRelativeTo(null); //для отображения в центре экрана
		myFrame.setVisible(true);  //показываем окно
	}
	//ф-ия для создания области для потока, параметры – название потока и сам поток
	private static Box createBox(String text, MyThread thread) {
		Box tempBox = new Box(BoxLayout.Y_AXIS); //главная коробка области
		//кнопки старта и стопа с соответствующими названиями
		JButton startButton = new JButton("start"+text);
		JButton stopButton = new JButton("stop"+text);
		JTextArea textArea = new JTextArea();//текстовая область
		startButton.setAlignmentX(Component.CENTER_ALIGNMENT); //выравниваем по центру все элементы
		stopButton.setAlignmentX(Component.CENTER_ALIGNMENT);
		textArea.setAlignmentX(Component.CENTER_ALIGNMENT);
		thread.setTa(textArea); //передаем в поток ссылку на текстовую область
		startButton.addActionListener(l->{ //слушатель для кнопки старта
			thread.start();
		});

		stopButton.addActionListener(l->{  //слушатель для кнопки стопа
			thread.interrupt();
		});
		tempBox.add(startButton); //добавляем все объекты в коробку
		tempBox.add(stopButton);
		tempBox.add(textArea);
		return tempBox;  //возвращаем коробку
	}
}
```

Задания

1. Сделать двухпотоковый пинг-понг: запускаются 2 потока, 1 имеет некое значение и передает его второму потоку. Второй получает его, выводит и передает обратно первому потоку. И всё это отображается в графическом интерфейсе.

2. (Для итоговой оценки 4) Переделать первую задачу для n потоков, т.е. многопоточный пинг-понг – первый передает второму, второй – третьему и т.д., последний передает первому и по новому кругу. Предусмотреть возможность задать начальное передаваемое значение и кол-во кругов. Причем должна быть возможность динамического добавления потоков.

Лабораторная работа №4. Работа с общим ресурсом

Поменяем условие предыдущей лабораторной: нужно, чтобы потоки брали значения из общего ресурса по одному строго по очереди, и эти значения из общего ресурса удалялись. Интерфейс может быть примерно следующим:

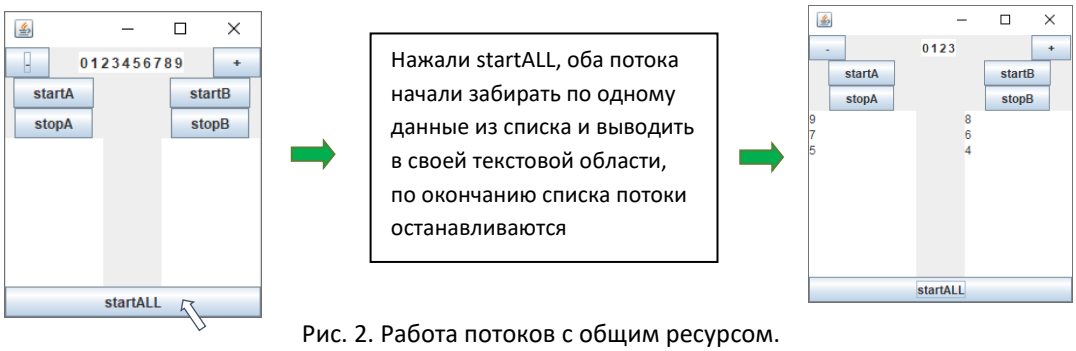


Рис. 2. Работа потоков с общим ресурсом.

Соответствующий код для создания интерфейса и обработки событий может быть таким:

```
public class MainClass2 {
	static MyThread2 threadA;  //потоки
	static MyThread2 threadB;
	static JList<Integer> myList; //объект для отображения списка чисел

	public static void main(String[] args) {
		createGUI();  //вызываем функцию для создания интерфейса
	}

	private static void createGUI() {  // функция для создания интерфейса
		JFrame myFrame = new JFrame();  //создаем окно
		myFrame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		Box centralBox = new Box(BoxLayout.X_AXIS); //центральная коробка
		threadA = new MyThread2();  //создаем потоки
		threadB = new MyThread2();
		Box uppperBox = createUpperBox();  //верхняя коробка с кнопками +, - и набором чисел
		Box leftBox = createBox("A", threadA);  //коробки для каждого потока, создаются функцией
		Box rightBox = createBox("B", threadB);

		centralBox.add(leftBox);  //добавляем левую коробку
		centralBox.add(Box.createHorizontalStrut(50));  //добавляем отступ
		centralBox.add(rightBox); //добавляем правую коробку

		JButton butDown = new JButton("startALL");  // кнопка для старта обоих потоков сразу
		butDown.addActionListener(e->{   //слушатель кнопки
		//передаем в потоки модель данных для списка с числами
			threadA.setListModel((DefaultListModel<Integer>) myList.getModel());
			threadB.setListModel((DefaultListModel<Integer>) myList.getModel());
			threadA.start();  //стартуем потоки
			threadB.start();
		});
		myFrame.add(uppperBox, BorderLayout.NORTH);  //добавляем всё на фрейм в нужные места
		myFrame.add(centralBox, BorderLayout.CENTER);
		myFrame.add(butDown, BorderLayout.SOUTH);
		myFrame.setSize(500, 500); //задаем размер окна
		myFrame.setLocationRelativeTo(null);  //чтоб был в центре экрана
		myFrame.setVisible(true);  //показываем окно
	}

	private static Box createUpperBox() { //функция для создания верхней коробки с кнопками +, - и набором чисел
		Box tempBox = new Box(BoxLayout.X_AXIS); //сама коробка
		myList = new JList<Integer>();  //список с числами
		myList.setLayoutOrientation(JList.HORIZONTAL_WRAP); //для горизонтального отображения списка
		myList.setVisibleRowCount(1);                       //в один ряд

		DefaultListModel<Integer> myModel = new DefaultListModel<>(); //модель данных для списка
		myList.setModel(myModel);  //связываем список с моделью
		for (int i = 0; i<10; i++)  //в цикле заполняем модель данными
			myModel.addElement(i);

		JButton butPlus = new JButton("+");  //кнопка для увеличения списка
```

```
            butPlus.addActionListener(e->{ //добавляем новый элемент, больше последнего на 1
                  myModel.addElement(myModel.lastElement()+1);
            });
            JButton butMinus = new JButton("-"); //кнопка для уменьшения списка
            butMinus.addActionListener(e->{ //удаляем последний элемент
                  myModel.remove(myModel.getSize()-1);
            });
            tempBox.add(butMinus);  //добавляем всё в коробку
            tempBox.add(Box.createHorizontalGlue());  //между объектами вставляем пружины
            tempBox.add(myList);
            tempBox.add(Box.createHorizontalGlue());
            tempBox.add(butPlus);
            return tempBox;  //возвращаем коробку
      }

      private static Box createBox(String text, MyThread2 thread) { //функция для коробки каждого потока
            Box tempBox = new Box(BoxLayout.Y_AXIS); //сама коробка
            tempBox.setAlignmentX(Box.CENTER_ALIGNMENT);  //выравнивание в коробке
            JButton startButton = new JButton("start"+text); //кнопки старта и стопа
            JButton stopButton = new JButton("stop"+text);
            JTextArea textArea = new JTextArea();  //текстовое поле
            startButton.setAlignmentX(Component.CENTER_ALIGNMENT); //выравнивание каждого элемента в коробке
            stopButton.setAlignmentX(Component.CENTER_ALIGNMENT);
            textArea.setAlignmentX(Component.CENTER_ALIGNMENT);

            thread.setTa(textArea);  //передаем текстовое поле в поток
            startButton.addActionListener(l->{  //слушатель кнопки старт
                  thread.setListModel((DefaultListModel<Integer>) myList.getModel());
                  thread.start();
            });

            stopButton.addActionListener(l->{  //слушатель кнопки стоп
                  thread.interrupt();
            });
            tempBox.add(startButton); //добавляем всё в коробку
            tempBox.add(stopButton);
            tempBox.add(textArea);
            return tempBox;   //возвращаем коробку
      }
}
```

И код для потокового класса MyThread2:

```
public class MyThread2 extends Thread {
      private JTextArea ta; // локальный объект типа JTextArea
      private int x; // начальное число
      private DefaultListModel<Integer> listModel; //локальный объект модели данных списка с окна

      public MyThread2() { //конструктор
            x = 0;
      }

      @Override
      public void run() {
            while (true) { //в бесконечном цикле
//создаем очередь для обработки действий с объектами в GUI, в том числе со списком
                  EventQueue.invokeLater(new Runnable() {
                        public void run() {
                              if (!listModel.isEmpty()) {//если список еще не пустой
                              //забираем последний элемент и убираем его из списка
                                    x = listModel.remove(listModel.getSize() - 1);
                                    ta.append(x + "\n"); //выводим этот элемент в своем поле
                              } else { //иначе, если список пустой, прерываем поток
                                    interrupt();
                              }
                        }
                  });
                  try {
                        Thread.currentThread().sleep(200); //для задержки потока на 200 мс
                  } catch (InterruptedException e) {
                        System.out.println(" exit"); //будет выведено, если поток будет прерван
                        return;
                  }
            }
      }
```

```
	public void setTa(JTextArea ta) { //сеттер для текстовой области, чтоб получить ее из окна
		this.ta = ta;
	}

	public void setListModel(DefaultListModel<Integer> listModel) { //сеттер для модели данных списка
		this.listModel = listModel;
		System.out.println(listModel.size());  //вывод в консоль для отладки
	}
}
```

Задания

1. Сделать аналогичную синхронизацию потоков для списка букв, причем буквы должны задаваться из промежутка, вводимого пользователем. Также добавить возможность сохранения букв в потоке и обмена этими буквами между потоками (например, первый поток берет по 2 буквы, второй по 1, после обмена первый будет брать по 1, второй – по 2).

2. (Для итоговой оценки 4) Добавить второй список, сделать динамическое добавление потоков. Потоки должны брать элемент из одного списка из начала, из второго – с конца.

Итоговая контрольная работа (на оценку 5)

Объединить лабораторные 2-4 в один интерфейс, чтобы можно было увидеть одновременную работу во всех лабораторных (в виде панелей внутри 1 окна, в виде разных окон, запускаемых на выбор и т.д.). И сделать вывод статистики по каждому заданию: сколько времени каждый процесс отработал.