|  |
| --- |
|  |
| МИНОБРНАУКИ РОССИИ |
| *Федеральное государственное бюджетное образовательное учреждение высшего образования* ***«МИРЭА – Российский технологический университет»***  **РТУ МИРЭА** |

**Институт информационных технологий (ИТ)**

**Кафедра инструментального и прикладного программного обеспечения (ИиППО)**

**Дисциплина «Программирование на языке Джава»**

**ОТЧЕТ**

**ПО ПРАКТИЧЕСКОМУ ЗАНЯТИЮ №9, №10**

Выполнил студент группы ИНБО-02-20 Самойленко М. А.

Принял Степанов П.В.

Практические работы выполнены «\_\_\_»\_\_\_\_\_\_\_2020г.

«\_\_\_\_\_\_\_\_\_\_\_\_\_\_» «\_\_\_»\_\_\_\_\_\_\_2020г.

Отметка о выполнении

**Москва – 2020 г.**

**Практическое занятие №7**

**Цель работы:** научиться создавать собственные исключения.

**Задание.**

Клиент совершает покупку онлайн. При оформлении заказа у пользователя запрашивается фио и номер ИНН. В программе проверяется, действителен ли номер ИНН для такого клиента. Исключение будет выдано в том случае, если введен недействительный ИНН.

Предлагается модернизировать задачу из предыдущей лабораторной работы (см. методические указания по выполнению лабораторных работ №1-8) – задача сортировки студентов по среднему баллу. Необходимо разработать пользовательский интерфейс для задачи поиска и сортировки (использовать массив интерфейсных ссылок- пример в лекции 5). Дополнить ее поиском студента по фио – в случае отсутствия такого студента необходимо выдавать собственное исключение.

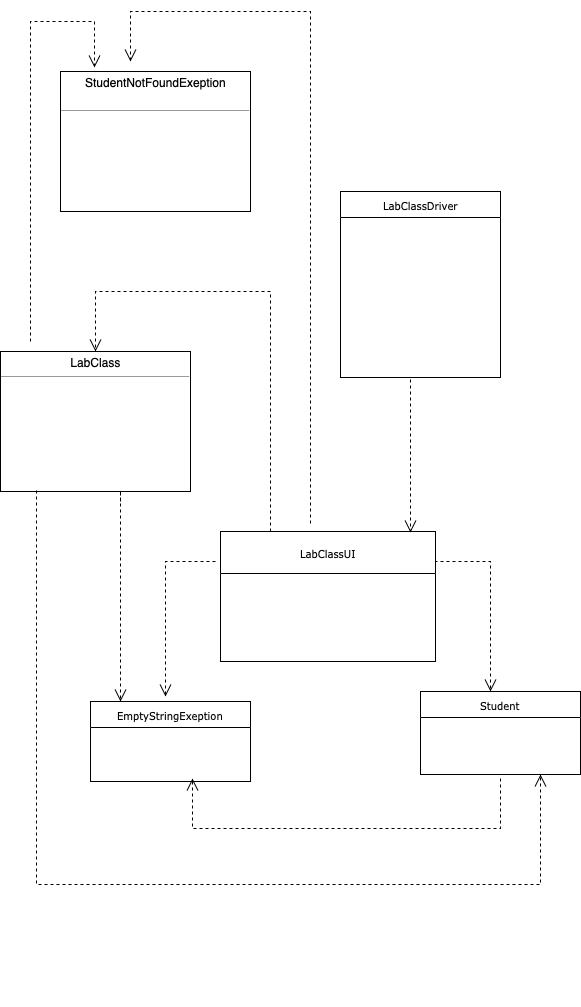
****

Рис. 1 – UML – диаграмма

**Класс LabClassDriver**

import javax.swing.JFrame;  
public class LabClassDriver {  
 public static void main (String[] args)  
 {  
 JFrame frame = new JFrame ("Students");  
 frame.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);  
 frame.getContentPane().add (new LabClassUI());  
 frame.pack();  
 frame.setVisible(true);  
 }  
}

**Класс EmptyStringExeption**

public class EmptyStringExeption extends Exception{  
}

**Интерфейс LabClass**

public interface LabClass {  
 void AddStudent(Student s);  
 void FindStudent(Student s);  
 void SortStudents();  
}

**Класс LabClassUI**

import javax.swing.\*;  
import java.awt.\*;  
import java.awt.event.ActionEvent;  
import java.util.ArrayList;  
import java.util.Comparator;  
import java.util.List;  
  
public class LabClassUI extends JPanel implements LabClass {  
 private final int WIDTH = 1200, HEIGHT = 1200;  
 private List<Student> students = new ArrayList<>();  
  
 JLabel mainLabel = new JLabel();  
 JLabel innLabel = new JLabel();  
 JLabel filLabel = new JLabel();  
 JTextField fio = new JTextField();  
 JTextField inn = new JTextField();  
 JButton Adding = new JButton(), Finding = new JButton(), Sorting = new JButton();  
  
 JPanel[] pnl = new JPanel[3];  
 JPanel[] pnlD = new JPanel[3];  
 JPanel[] pnlCenterDown = new JPanel[2];  
 JPanel[] pnlTop = new JPanel[3];  
 JPanel[] pnlLabel = new JPanel[2];  
 LabClassUI(){  
 setPreferredSize(new Dimension(WIDTH, HEIGHT));  
 setLayout(new GridLayout(3, 1));  
 for(int i = 0 ; i < pnl.length ; i++)  
 {  
 pnl[i] = new JPanel();  
 add(pnl[i]);  
 }  
 pnl[0].setLayout(new GridLayout(1, 3));  
 pnl[1].setLayout(new GridLayout(1, 3));  
 pnl[2].setLayout(new GridLayout(1, 3));  
 for(int i = 0 ; i < 3 ; i++)  
 {  
 pnlD[i] = new JPanel();  
 pnl[2].add(pnlD[i]);  
 pnlTop[i] = new JPanel();  
 pnl[0].add(pnlTop[i]);  
 }  
 pnlD[1].setLayout(new GridLayout(2, 1));  
 pnlD[0].setLayout(new GridLayout(2, 1));  
 for(int i = 0 ; i < 2 ; i++)  
 {  
 pnlCenterDown[i] = new JPanel();  
 pnlD[1].add(pnlCenterDown[i]);  
 pnlLabel[i] = new JPanel();  
 pnlD[0].add(pnlLabel[i]);  
 }  
 mainLabel.setPreferredSize(new Dimension(1200, 400));  
 fio.setPreferredSize(new Dimension(400, 150));  
 inn.setPreferredSize(new Dimension(400, 150));  
 Adding.setPreferredSize(new Dimension(400, 400));  
 Finding.setPreferredSize(new Dimension(400, 400));  
 Sorting.setPreferredSize(new Dimension(400, 400));  
  
 Font font = new Font("Times New Roman", Font.*BOLD*, 30);  
  
 mainLabel.setVerticalAlignment(JLabel.*CENTER*);  
 mainLabel.setHorizontalAlignment(JLabel.*CENTER*);  
 mainLabel.setFont(font);  
 mainLabel.setForeground(Color.*BLACK*);  
 mainLabel.setText("Введите ФИО и ИНН");  
  
 innLabel.setVerticalAlignment(JLabel.*CENTER*);  
 innLabel.setHorizontalAlignment(JLabel.*CENTER*);  
 innLabel.setFont(font);  
 innLabel.setForeground(Color.*BLACK*);  
 innLabel.setText("Введите ИНН");  
  
 filLabel.setVerticalAlignment(JLabel.*BOTTOM*);  
 filLabel.setHorizontalAlignment(JLabel.*CENTER*);  
 filLabel.setFont(font);  
 filLabel.setForeground(Color.*BLACK*);  
 filLabel.setText("Введите ФИО");  
  
 Adding.setVerticalAlignment(JLabel.*CENTER*);  
 Adding.setHorizontalAlignment(JLabel.*CENTER*);  
 Adding.setFont(font);  
 Adding.setForeground(Color.*BLACK*);  
 Adding.setText("Добавить студента");  
  
 Finding.setVerticalAlignment(JLabel.*CENTER*);  
 Finding.setHorizontalAlignment(JLabel.*CENTER*);  
 Finding.setFont(font);  
 Finding.setForeground(Color.*BLACK*);  
 Finding.setText("Оформить заказ");  
  
 Sorting.setVerticalAlignment(JLabel.*CENTER*);  
 Sorting.setHorizontalAlignment(JLabel.*CENTER*);  
 Sorting.setFont(font);  
 Sorting.setForeground(Color.*BLACK*);  
 Sorting.setText("Отсортировать студентов");  
  
 fio.setHorizontalAlignment(JLabel.*HORIZONTAL*);  
 fio.setFont(font);  
 fio.setForeground(Color.*BLACK*);  
  
 inn.setHorizontalAlignment(JLabel.*CENTER*);  
 inn.setFont(font);  
 inn.setForeground(Color.*BLACK*);  
  
 pnl[1].add(mainLabel);  
 pnlCenterDown[0].add(fio);  
 pnlCenterDown[1].add(inn);  
 pnlTop[0].add(Sorting);  
 pnlTop[1].add(Adding);  
 pnlTop[2].add(Finding);  
 pnlLabel[0].add(filLabel);  
 pnlLabel[1].add(innLabel);  
  
 Adding.addActionListener(new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String Fio = fio.getText();  
 String Inn = inn.getText();  
 Student stu = new Student(Fio, Inn);  
 AddStudent(stu);  
 }  
 });  
  
 Sorting.addActionListener(new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 SortStudents();  
 }  
 });  
  
 Finding.addActionListener(new AbstractAction() {  
 @Override  
 public void actionPerformed(ActionEvent e) {  
 String Fio = fio.getText();  
 String Inn = inn.getText();  
 Student stu = new Student(Fio, Inn);  
 FindStudent(stu);  
 }  
 });  
 }  
  
 @Override  
 public void FindStudent(Student s) {  
 boolean f = true;  
 try{  
 for (Student student: students){  
 if (student.getName().equals(s.getName())){  
 f = false;  
 if(s.getInn().equals(student.getInn())){  
 mainLabel.setText("Заказ оформлен");  
 } else  
 throw new EmptyStringExeption();  
 break;  
 }  
 }  
 if (f)  
 throw new NotFoundStudentExeption();  
 } catch (NotFoundStudentExeption err){  
 mainLabel.setText("Такого студента нет в базе");  
 } catch (EmptyStringExeption err){  
 mainLabel.setText("Ваш ИНН недействителен");  
 }  
 }  
  
 @Override  
 public void AddStudent(Student s) {  
 try {  
 if (!(s.getName().equals("") || s.getInn().equals(""))) {  
 students.add(s);  
 mainLabel.setText("Студент успешно добавлен");  
 } else {  
 throw new EmptyStringExeption();  
 }  
 }  
 catch (EmptyStringExeption err) {  
 mainLabel.setText("Заполните все поля");  
 }  
 }  
  
 @Override  
 public void SortStudents() {  
 Comparator<Student> comp = new SortingStudents();  
 students.sort(comp);  
 mainLabel.setText("Список успешно отсортирован");  
 }  
}

**Класс NotFoundStudentExeption**

public class NotFoundStudentExeption extends Exception{  
}

**Класс SortingStudents**

import java.util.Comparator;  
  
public class SortingStudents implements Comparator<Student> {  
  
 @Override  
 public int compare(Student o1, Student o2) {  
 return o1.getName().hashCode() - o2.getName().hashCode();  
 }  
}

**Класс Student**

package ru.mirea.java.practice9;  
  
public class Student {  
 private String name;  
 private String inn;  
  
 public Student(String name, String inn){  
 this.name = name;  
 this.inn = inn;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public String getInn() {  
 return inn;  
 }  
  
 public void setInn(String inn) {  
 this.inn = inn;  
 }  
}

**Тесты**

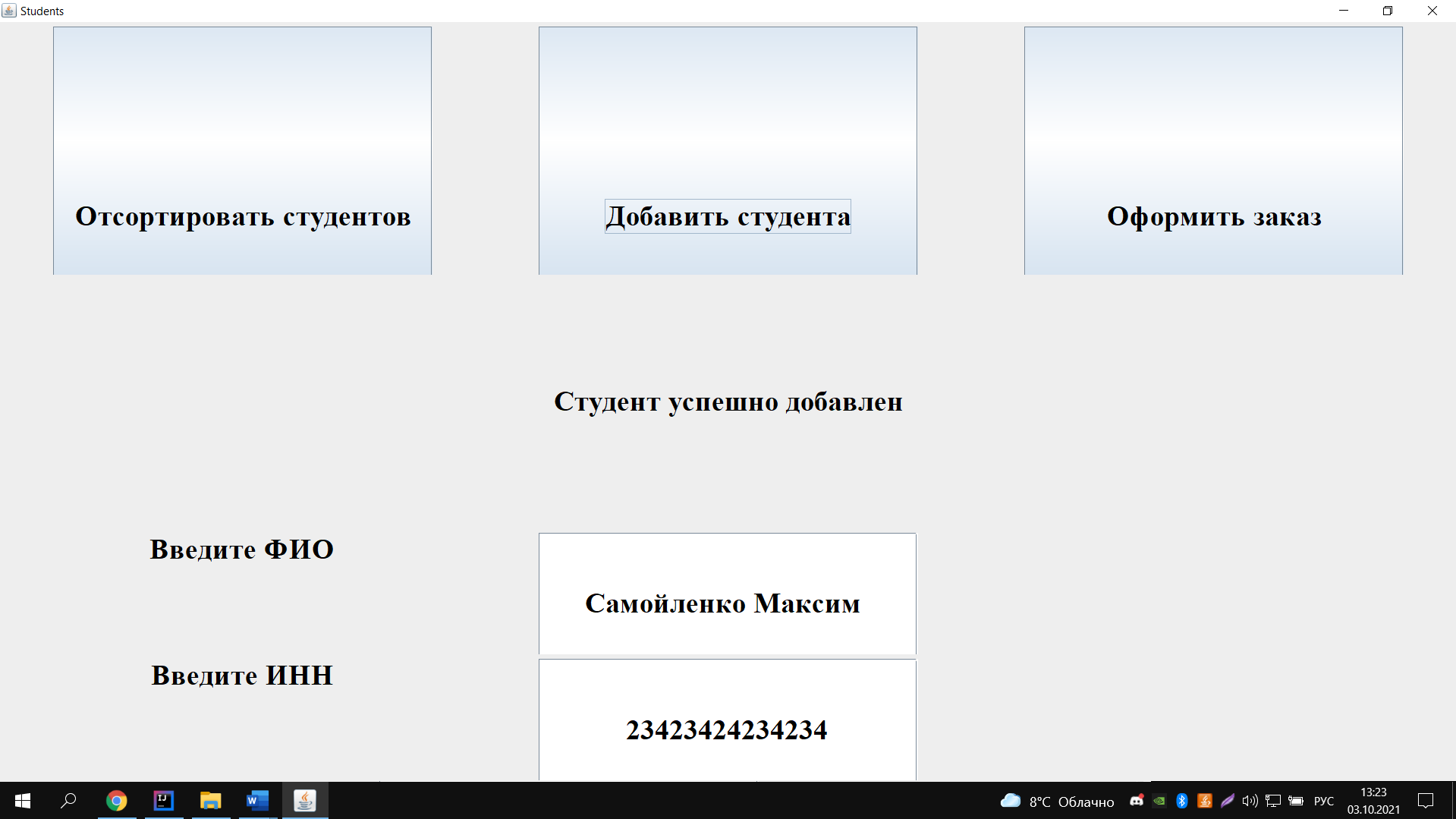


Рис. 2 – добавление студента в базу

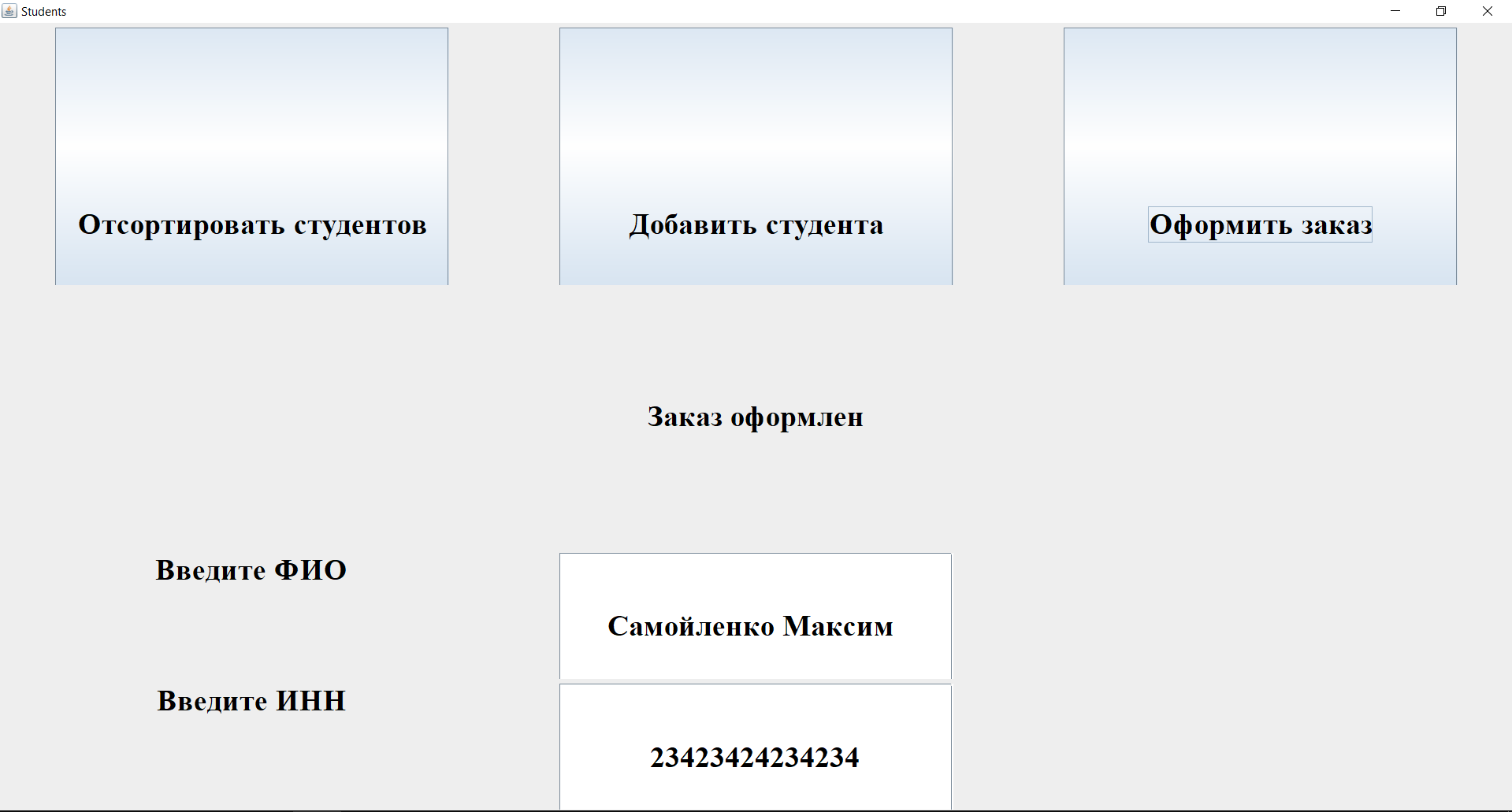


Рис.3 – оформление заказа от добавленного студента

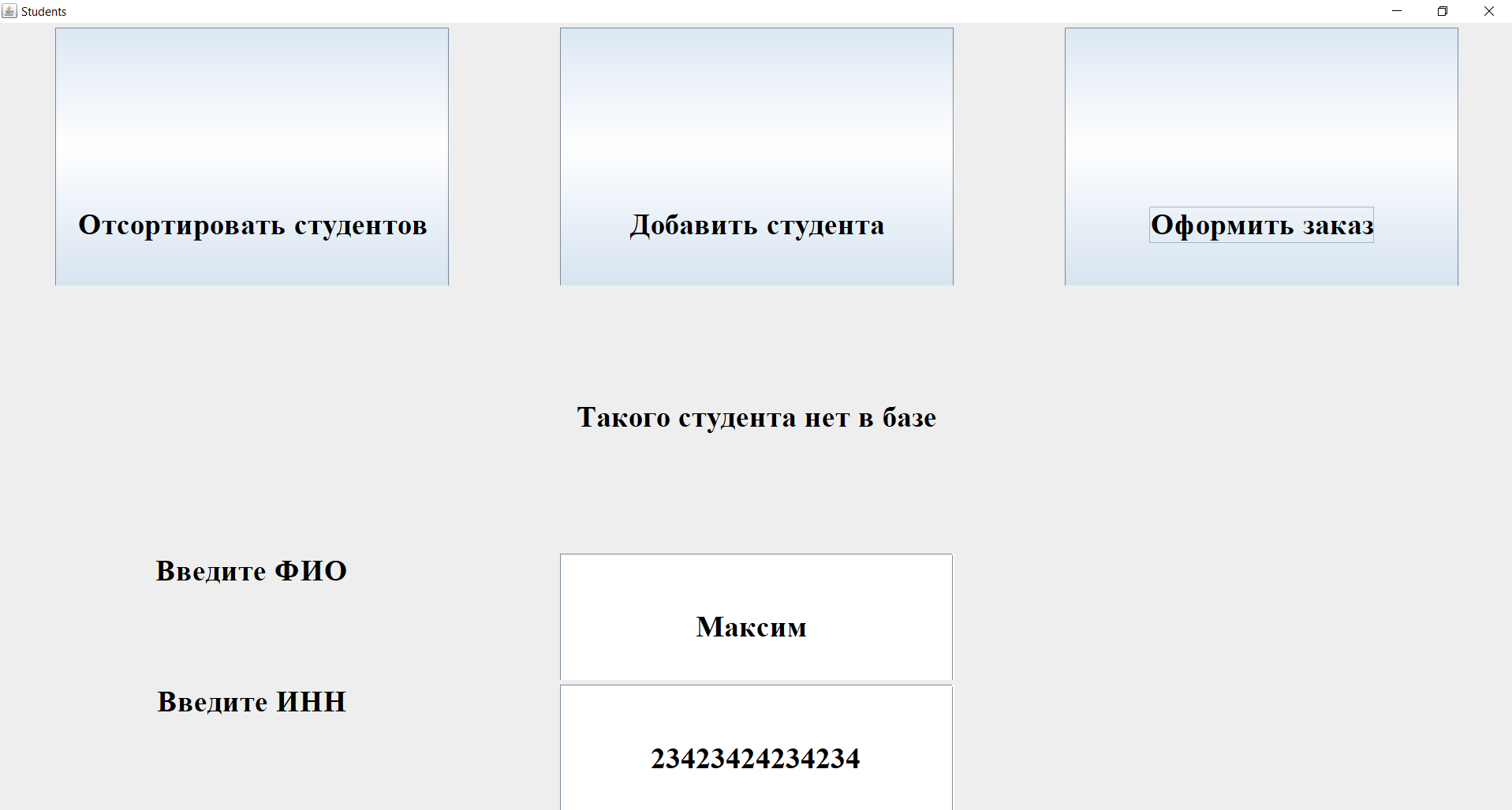


Рис.4 – оформление заказа от неизвестного студента

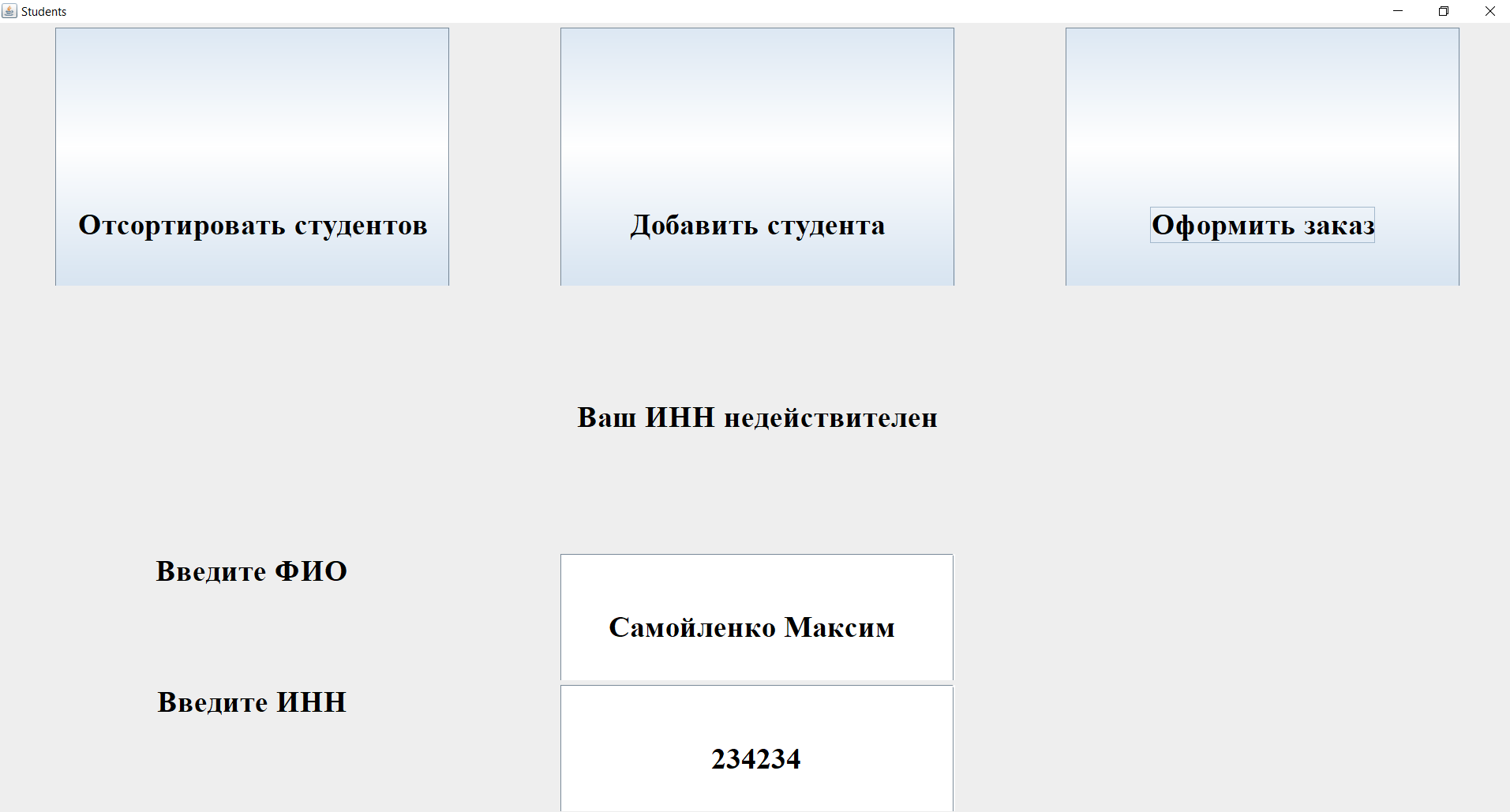


Рис.5 – оформление заказа с неправильным ИНН

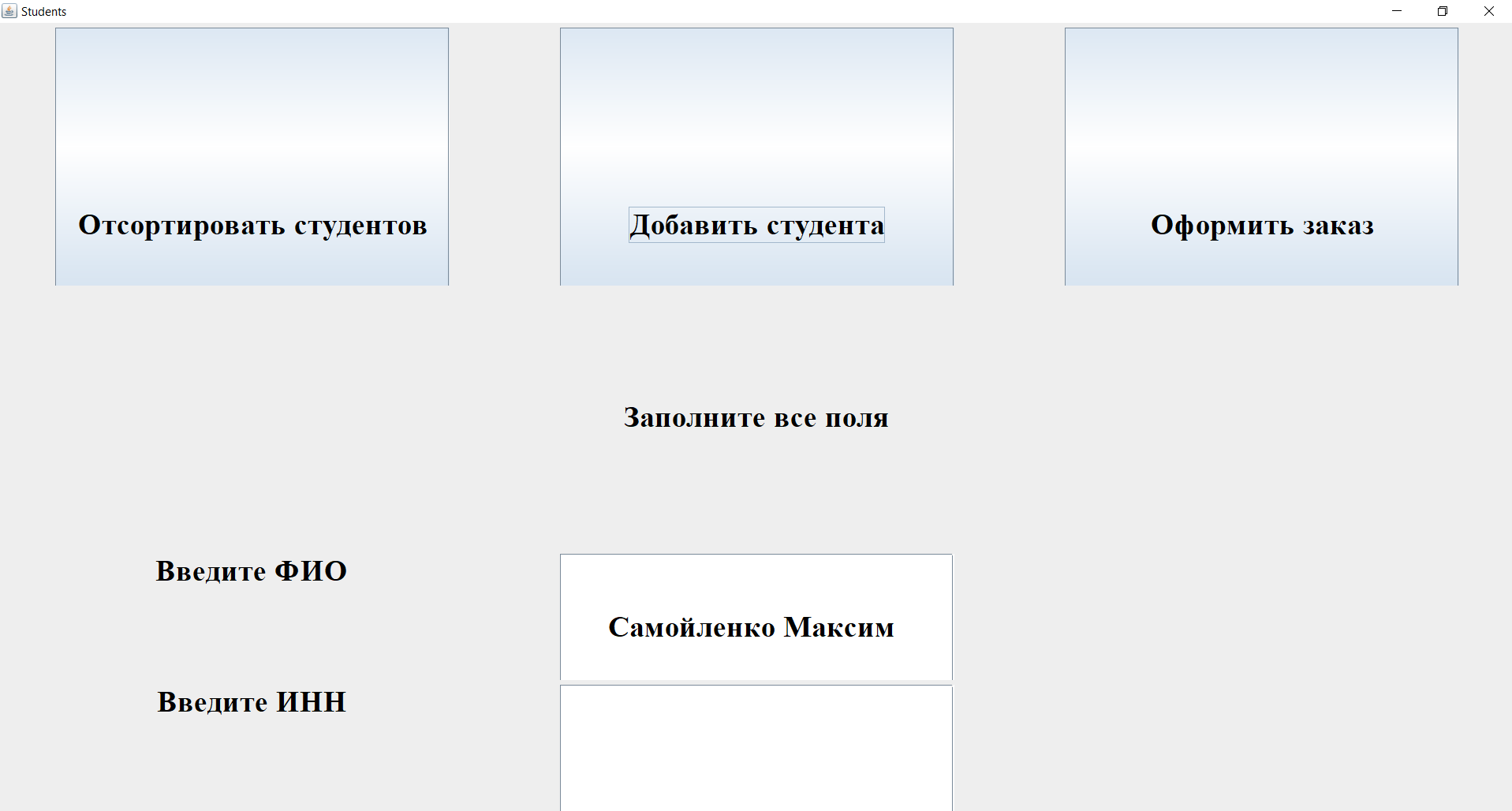


Рис. 6 – неполностью заполненные поля

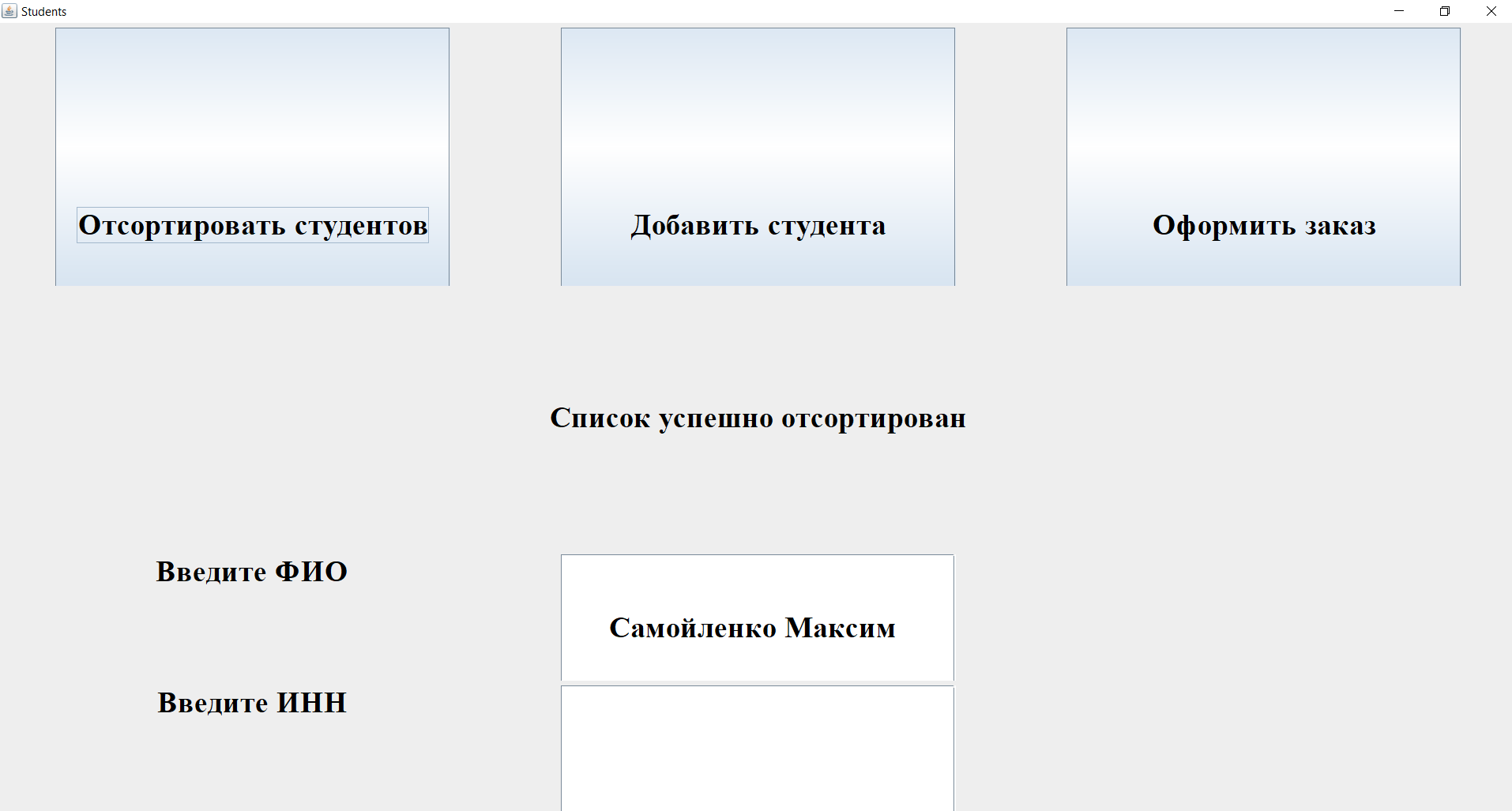


Рис.7 – сортировка списка студентов

**Практическое занятие №10**

**ЦЕЛЬ РАБОТЫ:**

Цель данной практической работы – научиться применять порождающие паттерны при разработке программ на Java. В данной практической работе рекомендуется использовать следующие паттерны: Абстрактная фабрика и фабричный метод.

**Задание 1:**

Реализовать класс Абстрактная фабрика для комплексных чисел.

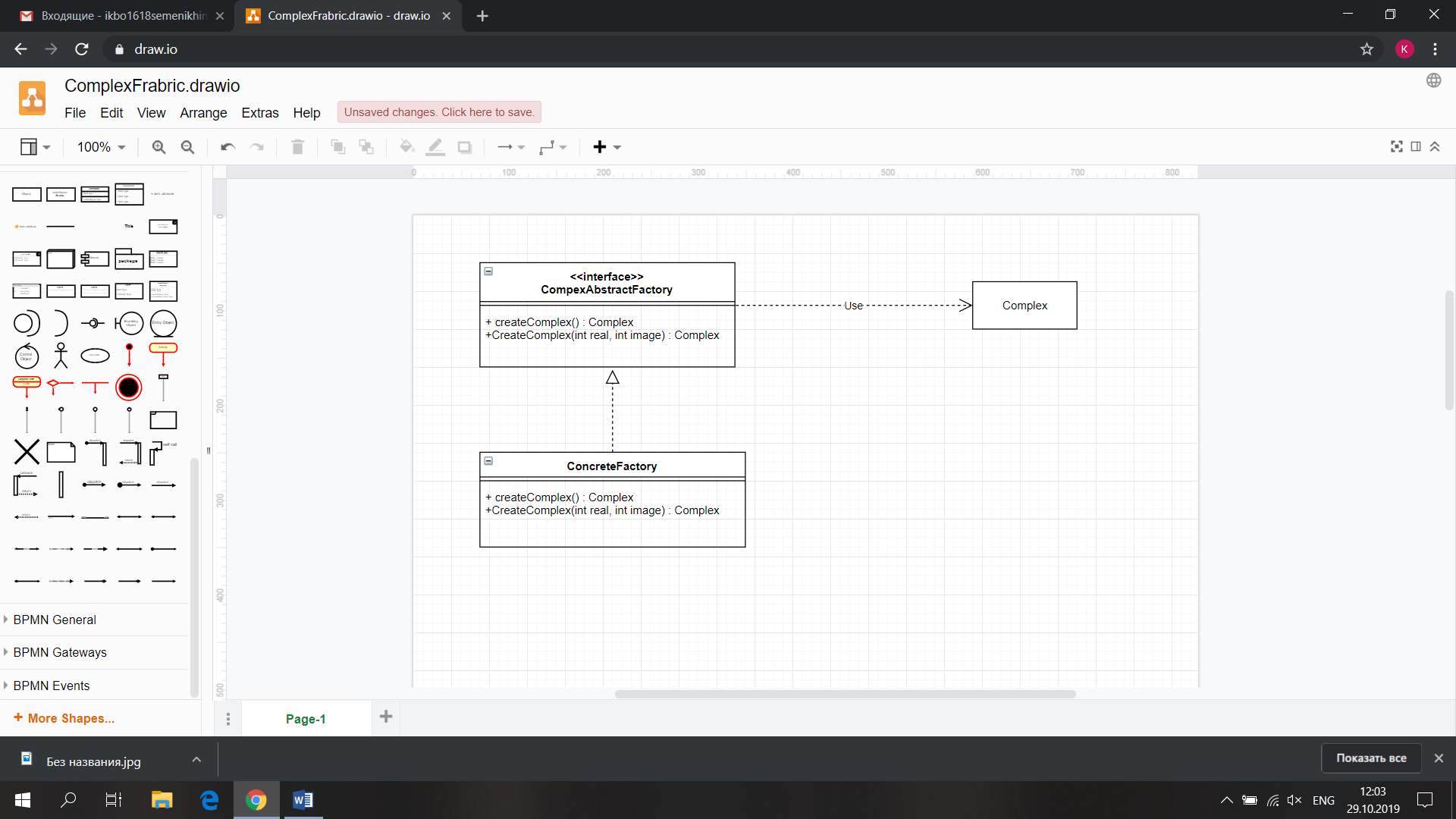


Рис 1. – UML-диграмма

**Код программы:**

**Интерфейс ComplexAbstactFactory**

public interface ComplexAbstactFactory {  
 public Complex createComplex();  
 public Complex createComplex(int real, int imaginary);  
}

**Класс ConcreteFactory**

public class ConcreteFactory implements ComplexAbstactFactory{  
  
 @Override  
 public Complex createComplex() {  
 Scanner in = new Scanner(System.*in*);  
 int real = in.nextInt();  
 int imaginary = in.nextInt();  
 Complex com = new Complex(real,imaginary);  
 return com;  
 }  
  
 @Override  
 public Complex createComplex(int real, int imaginary) {  
 Complex com = new Complex(real,imaginary);  
 return com;  
 }  
}

**Класс Complex**

public class Complex {  
 private int imaginary;  
 private int real;  
  
 public Complex(int real,int imaginary) {  
 this.imaginary = imaginary;  
 this.real = real;  
 }  
  
 public Complex() {  
 }  
  
 public int getImaginary() {  
 return imaginary;  
 }  
  
 public int getReal() {  
 return real;  
 }  
  
 public void setImaginary(int imaginary) {  
 this.imaginary = imaginary;  
 }  
  
 public void setReal(int real) {  
 this.real = real;  
 }  
  
 @Override  
 public String toString() {  
 return "Complex{" + " real=" + real +  
 ", imaginary= " + imaginary +  
 '}';  
 }  
}

**Класс Main**

public class Main {  
 public static void main(String[] args) {  
 ConcreteFactory n = new ConcreteFactory();  
 Complex com = n.createComplex();  
 Complex com2 = n.createComplex(5,6);  
 System.*out*.println(com.toString());  
 System.*out*.println(com2.toString());  
 }  
}

**Тест**

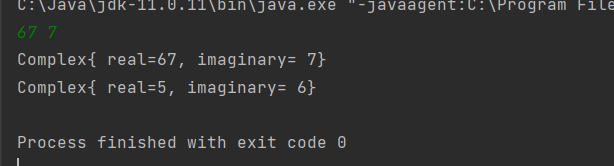


Рис. 2 – тест

**Задание 2:**

Реализовать класс Абстрактная фабрика для различных типов стульев: Викторианский стул, Многофункциональный стул, Магический стул, а также интерфейс Стул, от которого наследуются все классы стульев, и класс Клиент, который использует интерфейс стул в своем методе Sit (Chair chair).

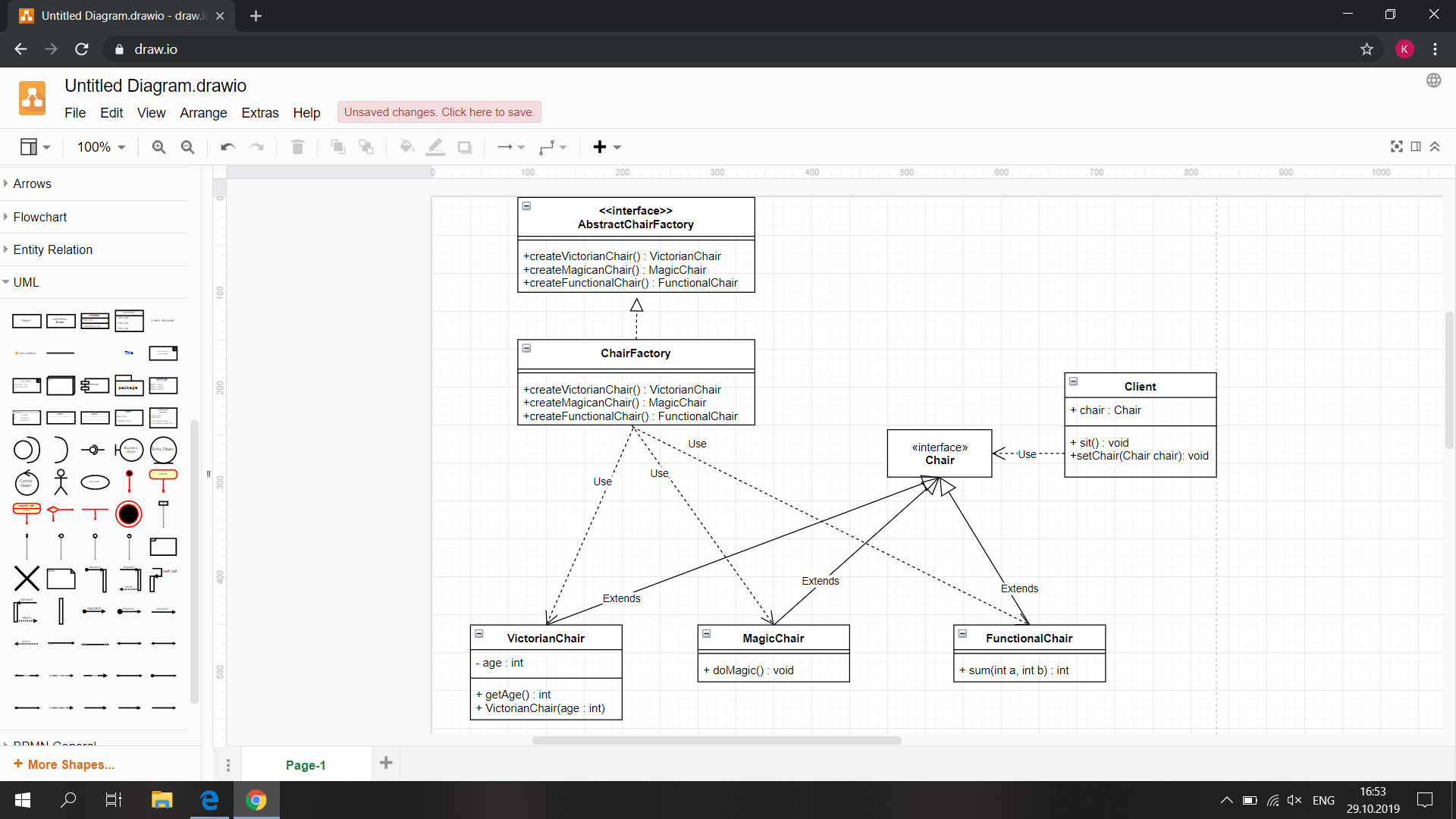


Рис.3 – UML-диаграмма

**Код программы:**

**Интерфейс AbstractChairFactory**

public interface AbstractChairFactory {  
 public VictorianChair createVictorianChair();  
 public MagicChair createMagicanChair();  
 public FunctionalChair createFunctionalChair();  
}

**Класс ChairFactory**

public class ChairFactory implements AbstractChairFactory{  
 @Override  
 public VictorianChair createVictorianChair() {  
 return new VictorianChair();  
 }  
  
 @Override  
 public MagicChair createMagicanChair() {  
 return new MagicChair();  
 }  
  
 @Override  
 public FunctionalChair createFunctionalChair() {  
 return new FunctionalChair();  
 }  
}

**Класс Client**

public class Client {  
 public Chair chair;  
 public void sit() {  
 System.*out*.println("Sitting");  
 }  
  
 public Client() {  
 }  
  
 public void setChair(Chair chair) {  
 this.chair = chair;  
 }  
}

**Класс Main**

public class Main {  
 public static void main(String[] args) {  
 Client c = new Client();  
 ChairFactory factory = new ChairFactory();  
 c.setChair(factory.createMagicanChair());  
 ((MagicChair) c.chair).doMagic();  
 c.setChair(factory.createFunctionalChair());  
 int sum = ((FunctionalChair) c.chair).sum(16, 4);  
 System.*out*.println("Сумма: " + sum);  
 c.setChair(factory.createVictorianChair());  
 ((VictorianChair) c.chair).setAge(67);  
 int age = ((VictorianChair) c.chair).getAge();  
 System.*out*.println("Возраст: " + age);  
 }  
}

**Интерфейс Chair**

public interface Chair {  
}

**Класс FunctionalChair**

public class FunctionalChair implements Chair {  
 public int sum(int a, int b){  
 return a+b;  
 }  
  
 public FunctionalChair() {  
 }  
}

**Класс MagicChair**

public class MagicChair implements Chair {  
 public void doMagic(){  
 System.*out*.println("\*\*Magic\*\*");  
 }  
  
 public MagicChair() {  
 }  
}

**Класс VictorianChair**

public class VictorianChair implements Chair {  
 private int age;  
  
 public int getAge() {  
 return age;  
 }  
  
 public VictorianChair(int age) {  
 this.age = age;  
 }  
  
 public VictorianChair() {  
 }  
  
 public void setAge(int age) {  
 this.age = age;  
 }  
}

**Тест**

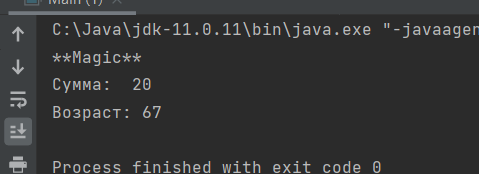


Рис.4 - тест

**Выводы**

Я научился создавать собственные исключения и реализовывать паттерны – типичные способы решения встречающихся проблем или проектировании программ.