================GUI Список контактов из Java-course=======================

package edu.javacourse.contact.test;

import edu.javacourse.contact.gui.ContactFrame;

/\*\*

\* Класс для запуска тестовых вызовов

\*/

public class ContactTest

{

public static void main(String[] args) {

ContactFrame cf = new ContactFrame();

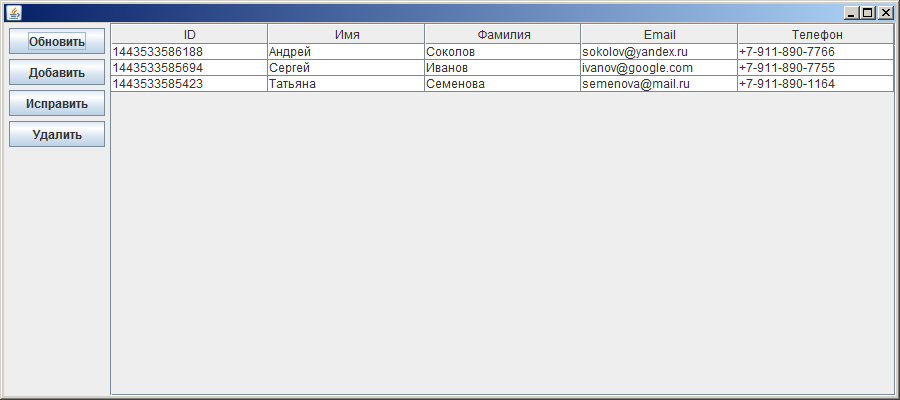
cf.setVisible(true);

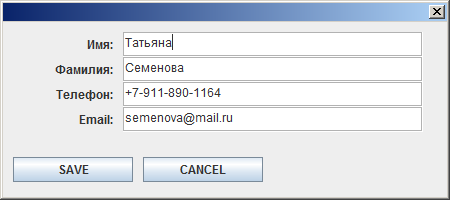
}

}

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138  139  140  141  142  143  144  145  146  147  148  149  150  151  152  153  154  155  156  157  158  159  160  161  162  163  164  165  166  167  168  169  170  171  172  173  174  175  176  177  178  179  180  181  182  183  184  185  186  187  188 | package edu.javacourse.contact.gui;    import edu.javacourse.contact.business.ContactManager;  import edu.javacourse.contact.entity.Contact;  import java.awt.BorderLayout;  import java.awt.GridBagConstraints;  import java.awt.GridBagLayout;  import java.awt.Insets;  import java.awt.event.ActionEvent;  import java.awt.event.ActionListener;  import java.util.List;  import javax.swing.JButton;  import javax.swing.JFrame;  import javax.swing.JOptionPane;  import javax.swing.JPanel;  import javax.swing.JScrollPane;  import javax.swing.JTable;  import javax.swing.ListSelectionModel;    public class ContactFrame extends JFrame implements ActionListener  {      private static final String LOAD = "LOAD";      private static final String ADD = "ADD";      private static final String EDIT = "EDIT";      private static final String DELETE = "DELETE";        private final ContactManager contactManager = new ContactManager();      private final JTable contactTable = new JTable();        // В конструкторе мы создаем нужные элементы      public ContactFrame() {          // Выставляем у таблицы свойство, которое позволяет выделить          // ТОЛЬКО одну строку в таблице          contactTable.setSelectionMode(ListSelectionModel.SINGLE\_INTERVAL\_SELECTION);            // Используем layout manager          GridBagLayout gridbag = new GridBagLayout();          GridBagConstraints gbc = new GridBagConstraints();          // Каждый элемент является последним в строке          gbc.gridwidth = GridBagConstraints.REMAINDER;          // Элемент раздвигается на весь размер ячейки          gbc.fill = GridBagConstraints.BOTH;          // Но имеет границы - слева, сверху и справа по 5. Снизу - 0          gbc.insets = new Insets(5, 5, 0, 5);            // Создаем панель для кнопок          JPanel btnPanel = new JPanel();          // усанавливаем у него layout          btnPanel.setLayout(gridbag);          // Создаем кнопки          btnPanel.add(createButton(gridbag, gbc, "Обновить", LOAD));          btnPanel.add(createButton(gridbag, gbc, "Добавить", ADD));          btnPanel.add(createButton(gridbag, gbc, "Исправить", EDIT));          btnPanel.add(createButton(gridbag, gbc, "Удалить", DELETE));            // Создаем панель для левой колокни с кнопками          JPanel left = new JPanel();          // Выставляем layout BorderLayout          left.setLayout(new BorderLayout());          // Кладем панель с кнопками в верхнюю часть          left.add(btnPanel, BorderLayout.NORTH);          // Кладем панель для левой колонки на форму слева - WEST          add(left, BorderLayout.WEST);            // Кладем панель со скролингом, внутри которой нахоится наша таблица          // Теперь таблица может скроллироваться          add(new JScrollPane(contactTable), BorderLayout.CENTER);            // выставляем координаты формы          setBounds(100, 200, 900, 400);          // При закрытии формы заканчиваем работу приложения          setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);            // Загружаем контакты          loadContact();      }        // Метод создает кнопку с заданными харктеристиками - заголовок и действие      private JButton createButton(GridBagLayout gridbag, GridBagConstraints gbc, String title, String action) {          // Создаем кнопку с заданным загловком          JButton button = new JButton(title);          // Действие будетп роверяться в обработчике и мы будем знать, какую          // именно кнопку нажали          button.setActionCommand(action);          // Обработчиком события от кнопки являемся сама форма          button.addActionListener(this);          // Выставляем свойства для размещения для кнопки          gridbag.setConstraints(button, gbc);          return button;      }        @Override      // Обработка нажатий кнопок      public void actionPerformed(ActionEvent ae) {          // Получаем команду - ActionCommand          String action = ae.getActionCommand();          // В зависимости от команды выполняем действия          switch (action) {              // Перегрузка данных              case LOAD:                  loadContact();                  break;              // Добавление записи              case ADD:                  addContact();                  break;              // Исправление записи              case EDIT:                  editContact();                  break;              // Удаление записи              case DELETE:                  deleteContact();                  break;          }      }        // Загрухить список контактов      private void loadContact() {          // Обращаемся к классу для загрузки списка контактов          List<Contact> contacts = contactManager.findContacts();          // Создаем модель, которой передаем полученный список          ContactModel cm = new ContactModel(contacts);          // Передаем нашу модель таблице - и она может ее отображать          contactTable.setModel(cm);      }        // Добавление контакта      private void addContact() {          // Создаем диалог для ввода данных          EditContactDialog ecd = new EditContactDialog();          // Обрабатываем закрытие диалога          saveContact(ecd);      }        // Редактирование контакта      private void editContact() {          // Получаем выделеннуб строку          int sr = contactTable.getSelectedRow();          // если строка выделена - можно ее редактировать          if (sr != -1) {              // Получаем ID контакта              Long id = Long.parseLong(contactTable.getModel().getValueAt(sr, 0).toString());              // получаем данные контакта по его ID              Contact cnt = contactManager.getContact(id);              // Создаем диалог для ввода данных и передаем туда контакт              EditContactDialog ecd = new EditContactDialog(cnt);              // Обрабатываем закрытие диалога              saveContact(ecd);          } else {              // Если строка не выделена - сообщаем об этом              JOptionPane.showMessageDialog(this, "Вы должны выделить строку для редактирования");          }      }        // Удаление контакта      private void deleteContact() {          // Получаем выделеннуб строку          int sr = contactTable.getSelectedRow();          if (sr != -1) {              // Получаем ID контакта              Long id = Long.parseLong(contactTable.getModel().getValueAt(sr, 0).toString());              // Удаляем контакт              contactManager.deleteContact(id);              // перегружаем список контактов              loadContact();          } else {              JOptionPane.showMessageDialog(this, "Вы должны выделить строку для удаления");          }      }        // Общий метод для добавления и изменения контакта      private void saveContact(EditContactDialog ecd) {          // Если мы нажали кнопку SAVE          if (ecd.isSave()) {              // Получаем контакт из диалогового окна              Contact cnt = ecd.getContact();              if (cnt.getContactId() != null) {                  // Если ID у контакта есть, то мы его обновляем                  contactManager.updateContact(cnt);              } else {                  // Если у контакта нет ID - значит он новый и мы его добавляем                  contactManager.addContact(cnt);              }              loadContact();          }      }  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67 | package edu.javacourse.contact.dao;    import edu.javacourse.contact.entity.Contact;  import java.util.ArrayList;  import java.util.Iterator;  import java.util.List;    public final class ContactSimpleDAO implements ContactDAO  {      private final List<Contact> contacts = new ArrayList<>();        // Прямо в конструкторе добавляем три контакта      public ContactSimpleDAO() {          addContact(new Contact("Андрей", "Соколов", "+7-911-890-7766", "sokolov@yandex.ru"));          addContact(new Contact("Сергей", "Иванов", "+7-911-890-7755", "ivanov@google.com"));          addContact(new Contact("Татьяна", "Семенова", "+7-911-890-1164", "semenova@mail.ru"));      }        @Override      public Long addContact(Contact contact) {          Long id = generateContactId();          contact.setContactId(id);          contacts.add(contact);          return id;      }        @Override      public void updateContact(Contact contact) {          Contact oldContact = getContact(contact.getContactId());          if(oldContact != null) {              oldContact.setFirstName(contact.getFirstName());              oldContact.setLastName(contact.getLastName());              oldContact.setPhone(contact.getPhone());              oldContact.setEmail(contact.getEmail());          }      }        @Override      public void deleteContact(Long contactId) {          for(Iterator<Contact> it = contacts.iterator(); it.hasNext();) {              Contact cnt = it.next();              if(cnt.getContactId().equals(contactId)) {                  it.remove();              }          }      }        @Override      public Contact getContact(Long contactId) {          for(Contact contact : contacts) {              if(contact.getContactId().equals(contactId)) {                  return contact;              }          }          return null;      }        @Override      public List<Contact> findContacts() {          return contacts;      }        private Long generateContactId() {          Long contactId = Math.round(Math.random() \* 1000 + System.currentTimeMillis());          while(getContact(contactId) != null) {              contactId = Math.round(Math.random() \* 1000 + System.currentTimeMillis());          }         return contactId;     } }  **ContactFrame** — основная форма для отображения. Содержит кнопки для редактирования списка контактов и отображает эти самые контакты.   1. **EditContactDialog** — диалоговое окно для редактиварония данных выделенного контакта. Появляется при нажатии кнопок «Добавить» или «Исправить» 2. **ContactModel** — этот класс предназначен для отображения таблицы контактов. Зачем он нужен — мы узнаем в свое время. |





package edu.javacourse.contact.gui;

import edu.javacourse.contact.entity.Contact;

import java.util.List;

import javax.swing.table.AbstractTableModel;

public class ContactModel extends AbstractTableModel

{

    // Список загловков для колонок в таблице

    private static final String[] headers = {"ID", "Имя", "Фамилия", "Email", "Телефон"};

    // Здесь мы храним список контактов, которые будем отображать в таблице

    private final List<Contact> contacts;

    public ContactModel(List<Contact> contacts) {

        this.contacts = contacts;

    }

    @Override

    // Получить количество строк в таблице - у нас это размер коллекции

    public int getRowCount() {

        return contacts.size();

    }

    @Override

    // Получить количество столбцов - их у нас стольк же, сколько полей

    // у класса Contact - всего пять

    public int getColumnCount() {

        return 5;

    }

    @Override

    // Вернуть загловок колонки - мы его берем из массива headers

    public String getColumnName(int col) {

        return headers[col];

    }

    @Override

    // Получить объект для тображения в кокнретной ячейке таблицы

    // В данном случае мы отдаем строковое представление поля

    public Object getValueAt(int row, int col) {

        Contact contact = contacts.get(row);

        // В зависимости от номера колонки возвращаем то или иное поле контакта

        switch (col) {

            case 0:

                return contact.getContactId().toString();

            case 1:

                return contact.getFirstName();

            case 2:

                return contact.getLastName();

            case 3:

                return contact.getEmail();

            default:

                return contact.getPhone();

        }

    }

}

Появление этого класса связано с достаточно известным шаблоном проектирования — MVC (Model, View, Controller) — Модель, Отображение, Контроллер (Управление).

======================================================================

package edu.javacourse.contact.gui;

import edu.javacourse.contact.business.ContactManager;

import edu.javacourse.contact.entity.Contact;

import java.awt.BorderLayout;

import java.awt.GridBagConstraints;

import java.awt.GridBagLayout;

import java.awt.Insets;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import java.util.List;

import javax.swing.JButton;

import javax.swing.JFrame;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JScrollPane;

import javax.swing.JTable;

import javax.swing.ListSelectionModel;

public class ContactFrame extends JFrame implements ActionListener

{

    private static final String LOAD = "LOAD";

    private static final String ADD = "ADD";

    private static final String EDIT = "EDIT";

    private static final String DELETE = "DELETE";

    private final ContactManager contactManager = new ContactManager();

    private final JTable contactTable = new JTable();

    // В конструкторе мы создаем нужные элементы

    public ContactFrame() {

        // Выставляем у таблицы свойство, которое позволяет выделить

        // ТОЛЬКО одну строку в таблице

        contactTable.setSelectionMode(ListSelectionModel.SINGLE\_INTERVAL\_SELECTION);

        // Используем layout manager

        GridBagLayout gridbag = new GridBagLayout();

        GridBagConstraints gbc = new GridBagConstraints();

        // Каждый элемент является последним в строке

        gbc.gridwidth = GridBagConstraints.REMAINDER;

        // Элемент раздвигается на весь размер ячейки

        gbc.fill = GridBagConstraints.BOTH;

        // Но имеет границы - слева, сверху и справа по 5. Снизу - 0

        gbc.insets = new Insets(5, 5, 0, 5);

        // Создаем панель для кнопок

        JPanel btnPanel = new JPanel();

        // усанавливаем у него layout

        btnPanel.setLayout(gridbag);

        // Создаем кнопки

        btnPanel.add(createButton(gridbag, gbc, "Обновить", LOAD));

        btnPanel.add(createButton(gridbag, gbc, "Добавить", ADD));

        btnPanel.add(createButton(gridbag, gbc, "Исправить", EDIT));

        btnPanel.add(createButton(gridbag, gbc, "Удалить", DELETE));

        // Создаем панель для левой колокни с кнопками

        JPanel left = new JPanel();

        // Выставляем layout BorderLayout

        left.setLayout(new BorderLayout());

        // Кладем панель с кнопками в верхнюю часть

        left.add(btnPanel, BorderLayout.NORTH);

        // Кладем панель для левой колонки на форму слева - WEST

        add(left, BorderLayout.WEST);

        // Кладем панель со скролингом, внутри которой нахоится наша таблица

        // Теперь таблица может скроллироваться

        add(new JScrollPane(contactTable), BorderLayout.CENTER);

        // выставляем координаты формы

        setBounds(100, 200, 900, 400);

        // При закрытии формы заканчиваем работу приложения

        setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

        // Загружаем контакты

        loadContact();

    }

    // Метод создает кнопку с заданными харктеристиками - заголовок и действие

    private JButton createButton(GridBagLayout gridbag, GridBagConstraints gbc, String title, String action) {

        // Создаем кнопку с заданным загловком

        JButton button = new JButton(title);

        // Действие будетп роверяться в обработчике и мы будем знать, какую

        // именно кнопку нажали

        button.setActionCommand(action);

        // Обработчиком события от кнопки являемся сама форма

        button.addActionListener(this);

        // Выставляем свойства для размещения для кнопки

        gridbag.setConstraints(button, gbc);

        return button;

    }

    @Override

    // Обработка нажатий кнопок

    public void actionPerformed(ActionEvent ae) {

        // Получаем команду - ActionCommand

        String action = ae.getActionCommand();

        // В зависимости от команды выполняем действия

        switch (action) {

            // Перегрузка данных

            case LOAD:

                loadContact();

                break;

            // Добавление записи

            case ADD:

                addContact();

                break;

            // Исправление записи

            case EDIT:

                editContact();

                break;

            // Удаление записи

            case DELETE:

                deleteContact();

                break;

        }

    }

    // Загрухить список контактов

    private void loadContact() {

        // Обращаемся к классу для загрузки списка контактов

        List<Contact> contacts = contactManager.findContacts();

        // Создаем модель, которой передаем полученный список

        ContactModel cm = new ContactModel(contacts);

        // Передаем нашу модель таблице - и она может ее отображать

        contactTable.setModel(cm);

    }

    // Добавление контакта

    private void addContact() {

        // Создаем диалог для ввода данных

        EditContactDialog ecd = new EditContactDialog();

        // Обрабатываем закрытие диалога

        saveContact(ecd);

    }

    // Редактирование контакта

    private void editContact() {

        // Получаем выделеннуб строку

        int sr = contactTable.getSelectedRow();

        // если строка выделена - можно ее редактировать

        if (sr != -1) {

            // Получаем ID контакта

            Long id = Long.parseLong(contactTable.getModel().getValueAt(sr, 0).toString());

            // получаем данные контакта по его ID

            Contact cnt = contactManager.getContact(id);

            // Создаем диалог для ввода данных и передаем туда контакт

            EditContactDialog ecd = new EditContactDialog(cnt);

            // Обрабатываем закрытие диалога

            saveContact(ecd);

        } else {

            // Если строка не выделена - сообщаем об этом

            JOptionPane.showMessageDialog(this, "Вы должны выделить строку для редактирования");

        }

    }

    // Удаление контакта

    private void deleteContact() {

        // Получаем выделеннуб строку

        int sr = contactTable.getSelectedRow();

        if (sr != -1) {

            // Получаем ID контакта

            Long id = Long.parseLong(contactTable.getModel().getValueAt(sr, 0).toString());

            // Удаляем контакт

            contactManager.deleteContact(id);

            // перегружаем список контактов

            loadContact();

        } else {

            JOptionPane.showMessageDialog(this, "Вы должны выделить строку для удаления");

        }

    }

    // Общий метод для добавления и изменения контакта

    private void saveContact(EditContactDialog ecd) {

        // Если мы нажали кнопку SAVE

        if (ecd.isSave()) {

            // Получаем контакт из диалогового окна

            Contact cnt = ecd.getContact();

            if (cnt.getContactId() != null) {

                // Если ID у контакта есть, то мы его обновляем

                contactManager.updateContact(cnt);

            } else {

                // Если у контакта нет ID - значит он новый и мы его добавляем

                contactManager.addContact(cnt);

            }

            loadContact();

        }

    }

}

1. Конструктор **ContactFrame** — в нем мы «строим» нашу форму. Создаем панели, кнопки, настраиваем их взаимодействие
2. Обработчки нажатий кнопок **actionPerformed** — именно сюда приходят все команды от кнопок и в нем мы выполняем действия оп редактированию нашего списка контактов

По умолчанию форма использует **BorderLayout**, который делит всю форму на пять частей — Север, Юг, Запад, Восток и Центр. Панель с кнопками мы поместим слева, т.е. на Западе. Все остальное пространство будет отдано Центру (т.к. остальные области ничего не содержат).  
Панель с кнопками (**btnPanel**) использует достаточно интересный (и сложный) LayoutManager — **GridBagLayout**, который хоть и располагает элементы в виде сетки, но предоставляет очень мощные инструменты управлния. Чтобы панель с кнопками не «расползалась» на всю левую сторону, я сначала «кладу» ее на другую панель **left** (у которой выставляю BorderLayout), и уже эту панель кладу на форму на Запад (слева).

С таблицей все гораздо проще — мы размещаем ее в объекте класса **JScrollPane**, который позволяет прокручвать элемент внутри себя и уже его кладем на форму в Центр.

Нажатия от кнопок будут обрабатываться нашей формой — пример такой обработки мы уже расcматривали в разделе [Интерфейсы](http://java-course.ru/begin/interfaces/). Для этого наша форма реализует интерфейс **ActionListener**.

Я бы выделил в MVC несколько моментов, которые являются основополагающими:

1. Есть данные и есть их отображатель — это РАЗНЫЕ вещи. Такая конструкция гораздо удобнее, нежели когда вы смешиваете все вместе.
2. Отображение данных и их изменение должны быть взаимосвязаны — если данные изменились, значит надо менять изображение

Т.е. если я поменял данные в таблице или списке, то их изображение должно измениться/обновиться. Сигнал об изменении идет через контроллер, который разобравшись, что надо делать (например, надо удалить), меняет модель. А т.к. модель «связана» с отображением, то она «посылает сигнал об изменении», тем самым побуждая отображение перерисовать модель.  
Иногда контроллер и отображатель совмещены в одном графическом элементе. Например, для передвижения по таблице мы используем клавиши стрелок вверх/вниз. Таблица «ловит» наши нажатия и передает в модель факт того, что была отмечена другая строка — модель делает себе отметку, что текущая строка такая-то и таблица отображает сей факт. Но мжет быть и не так — например в нашем приложении добавление, изменение и удаление будет производится через выполнение кода обработки нажатий наших кнопок. И наша задача — воздействовать именно на модель, а не на отображение, которым является таблица — в нашем случае это класс **JTable** (мы скоро увидим использование этого стандартного класса).  
И еще раз — постарайтесь увидеть эту связь. Модель (как отдельный объект) может подвергаться изменениям. Но т.к. она связана с отображением, то каждый раз при своем изменении модель посылает отображению сигнал, чтобы оторажение себя перерисовало. Но т.к. отображение при своем рисовании берет данные из модели — мы получим обновление данных уже в изображении.  
Так вот, для класса **JTable** надо, чтобы модель реализовывала интерфейс **TableModel**. В обычной жизни заниматься реализацией всех функций этого интерфейса нет необходимости и разработчики Java предлагают уже ПОЧТИ готовый класс **AbstractTableModel**. В этом классе нам достаточно переопределить всего 4 метода.  
Теперь наша модель — класс **ContactModel** — может использоваться совместно с лассом **JTable**.

package edu.javacourse.contact.gui;

import edu.javacourse.contact.entity.Contact;

import java.awt.Rectangle;

import java.awt.event.ActionEvent;

import java.awt.event.ActionListener;

import javax.swing.BorderFactory;

import javax.swing.JButton;

import javax.swing.JDialog;

import javax.swing.JLabel;

import javax.swing.JTextPane;

import javax.swing.SwingConstants;

public class EditContactDialog extends JDialog implements ActionListener

{

    // Заголовки кнопок

    private static final String SAVE = "SAVE";

    private static final String CANCEL = "CANCEL";

    // Размер отступа

    private static final int PAD = 10;

    // Ширина метки

    private static final int W\_L = 100;

    //Ширина поля для ввода

    private static final int W\_T = 300;

    // Ширина кнопки

    private static final int W\_B = 120;

    // высота элемента - общая для всех

    private static final int H\_B = 25;

    // Поле для ввода Имени

    private final JTextPane txtFirstName = new JTextPane();

    // Поле для ввода Фамилии

    private final JTextPane txtLastName = new JTextPane();

    // Поле для ввода Телефона

    private final JTextPane txtPhone = new JTextPane();

    // Поле для ввода E-mail

    private final JTextPane txtEmail = new JTextPane();

    // Поле для хранения ID контакта, если мы собираемся редактировать

    // Если это новый контакт - cjntactId == null

    private Long contactId = null;

    // Надо ли записывать изменения после закрытия диалога

    private boolean save = false;

    public EditContactDialog() {

        this(null);

    }

    public EditContactDialog(Contact contact) {

        // Убираем layout - будем использовать абсолютные координаты

        setLayout(null);

        // Выстраиваем метки и поля для ввода

        buildFields();

        // Если нам передали контакт - заполняем поля формы

        initFields(contact);

        // выстариваем кнопки

        buildButtons();

        // Диалог в модальном режиме - только он активен

        setModal(true);

        // Запрещаем изменение размеров

        setResizable(false);

        // Выставляем размеры формы

        setBounds(300, 300, 450, 200);

        // Делаем форму видимой

        setVisible(true);

    }

    // Размещаем метки и поля ввода на форме

    private void buildFields() {

        // Набор метки и поля для Имени

        JLabel lblFirstName = new JLabel("Имя:");

        // Выравнивание текста с правой стороны

        lblFirstName.setHorizontalAlignment(SwingConstants.RIGHT);

        // Выставляем координаты метки

        lblFirstName.setBounds(new Rectangle(PAD, 0 \* H\_B + PAD, W\_L, H\_B));

        // Кладем метку на форму

        add(lblFirstName);

        // Выставляем координаты поля

        txtFirstName.setBounds(new Rectangle(W\_L + 2 \* PAD, 0 \* H\_B + PAD, W\_T, H\_B));

        // Делаем "бордюр" для поля

        txtFirstName.setBorder(BorderFactory.createEtchedBorder());

        // Кладем поле на форму

        add(txtFirstName);

        // Набор метки и поля для Фамилии

        JLabel lblLastName = new JLabel("Фамилия:");

        lblLastName.setHorizontalAlignment(SwingConstants.RIGHT);

        lblLastName.setBounds(new Rectangle(PAD, 1 \* H\_B + PAD, W\_L, H\_B));

        add(lblLastName);

        txtLastName.setBounds(new Rectangle(W\_L + 2 \* PAD, 1 \* H\_B + PAD, W\_T, H\_B));

        txtLastName.setBorder(BorderFactory.createEtchedBorder());

        add(txtLastName);

        // Набор метки и поля для Телефона

        JLabel lblPhone = new JLabel("Телефон:");

        lblPhone.setHorizontalAlignment(SwingConstants.RIGHT);

        lblPhone.setBounds(new Rectangle(PAD, 2 \* H\_B + PAD, W\_L, H\_B));

        add(lblPhone);

        txtPhone.setBounds(new Rectangle(W\_L + 2 \* PAD, 2 \* H\_B + PAD, W\_T, H\_B));

        txtPhone.setBorder(BorderFactory.createEtchedBorder());

        add(txtPhone);

        // Набор метки и поля для Email

        JLabel lblEmail = new JLabel("Email:");

        lblEmail.setHorizontalAlignment(SwingConstants.RIGHT);

        lblEmail.setBounds(new Rectangle(PAD, 3 \* H\_B + PAD, W\_L, H\_B));

        add(lblEmail);

        txtEmail.setBounds(new Rectangle(W\_L + 2 \* PAD, 3 \* H\_B + PAD, W\_T, H\_B));

        txtEmail.setBorder(BorderFactory.createEtchedBorder());

        add(txtEmail);

    }

    // Если нам епередали контакт - заполняем поля из контакта

    private void initFields(Contact contact) {

        if (contact != null) {

            contactId = contact.getContactId();

            txtFirstName.setText(contact.getFirstName());

            txtLastName.setText(contact.getLastName());

            txtEmail.setText(contact.getEmail());

            txtPhone.setText(contact.getPhone());

        }

    }

    // Размещаем кнопки на форме

    private void buildButtons() {

        JButton btnSave = new JButton("SAVE");

        btnSave.setActionCommand(SAVE);

        btnSave.addActionListener(this);

        btnSave.setBounds(new Rectangle(PAD, 5 \* H\_B + PAD, W\_B, H\_B));

        add(btnSave);

        JButton btnCancel = new JButton("CANCEL");

        btnCancel.setActionCommand(CANCEL);

        btnCancel.addActionListener(this);

        btnCancel.setBounds(new Rectangle(W\_B + 2 \* PAD, 5 \* H\_B + PAD, W\_B, H\_B));

        add(btnCancel);

    }

    @Override

    // Обработка нажатий кнопок

    public void actionPerformed(ActionEvent ae) {

        String action = ae.getActionCommand();

        // Если нажали кнопку SAVE (сохранить изменения) - запоминаем этой

        save = SAVE.equals(action);

        // Закрываем форму

        setVisible(false);

    }

    // Надо ли сохранять изменения

    public boolean isSave() {

        return save;

    }

    // Создаем контакт из заполенных полей, который можно будет записать

    public Contact getContact() {

        Contact contact = new Contact(contactId, txtFirstName.getText(),

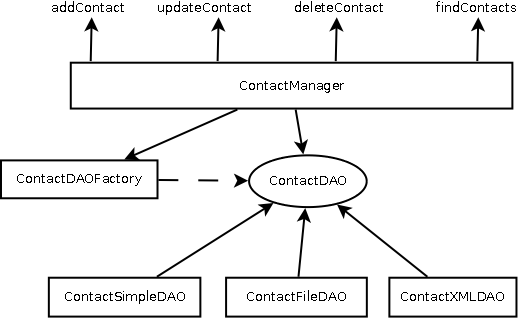
                txtLastName.getText(), txtPhone.getText(), txtEmail.getText());

        return contact;

    }

}

================GUI Список контактов из Java-course======================= 27



package edu.javacourse.contact.entity;

/\*\*

\*   Класс для хранения данных контакта

\*/

public class Contact

{

    // Идентификатор контакта

    private Long contactId;

    // Имя

    private String firstName;

    // Фамилия

    private String lastName;

    // Телефон

    private String phone;

    // email

    private String email;

    public Contact() {

    }

    public Contact(String firstName, String lastName, String phone, String email) {

        this.firstName = firstName;

        this.lastName = lastName;

        this.phone = phone;

        this.email = email;

    }

    public Contact(Long contactId, String firstName, String lastName, String phone, String email) {

        this.contactId = contactId;

        this.firstName = firstName;

        this.lastName = lastName;

        this.phone = phone;

        this.email = email;

    }

    public Long getContactId() {

        return contactId;

    }

    public void setContactId(Long contactId) {

        this.contactId = contactId;

    }

    public String getFirstName() {

        return firstName;

    }

    public void setFirstName(String firstName) {

        this.firstName = firstName;

    }

    public String getLastName() {

        return lastName;

    }

    public void setLastName(String lastName) {

        this.lastName = lastName;

    }

    public String getPhone() {

        return phone;

    }

    public void setPhone(String phone) {

        this.phone = phone;

    }

    public String getEmail() {

        return email;

    }

    public void setEmail(String email) {

        this.email = email;

    }

    @Override

    public String toString() {

        return "Contact{" + "contactId=" + contactId + ", firstName=" + firstName + ", lastName=" + lastName + ", phone=" + phone + ", email=" + email + '}';

    }

}

==========================================================

package edu.javacourse.contact.dao;

import edu.javacourse.contact.entity.Contact;

import java.util.List;

/\*\*

\* Интерфейс для определения функций хранлиза данных о контактах

\*/

public interface ContactDAO

{

    // Добавление контакта - возвращает ID добавленного контакта

    public Long addContact(Contact contact);

    // Редактирование контакта

    public void updateContact(Contact contact);

    // Удаление контакта по его ID

    public void deleteContact(Long contactId);

    // Получение контакта

    public Contact getContact(Long contactId);

    // Получение списка контактов

    public List<Contact> findContacts();

}

=======================================================================

package edu.javacourse.contact.dao;

/\*\*

\* Фабрика для создания экземпляра ContactDAO

\*/

public class ContactDAOFactory

{

    public static ContactDAO getContactDAO() {

        return new ContactSimpleDAO();

    }

}