



# Java course

- [Начало Java](#)
- [Проект «Отдел кадров»](#)
- [Курсы](#)
- [Статьи](#)
- [Контакты/Вопросы](#)

- [Введение](#)
- [Установка JDK](#)
- [Основные шаги](#)
- [Данные](#)
- [Порядок операций](#)
- [IDE NetBeans](#)
- [ООП](#)
- [Инкапсуляция](#)
- [Наследование](#)
- [Пакеты](#)
- [Переопределение и перегрузка](#)
- [Полиморфизм](#)
- [Статические свойства и методы](#)
- [Отношения между классами](#)
- [Визуализация робота](#)
- [Пример — очередь объектов](#)
- [Массивы — знакомство](#)
- [Многомерные массивы](#)
- [Абстрактные классы](#)
- [Интерфейсы](#)

## Список контактов в виде простого GUI-приложения

Пришла пора сделать из нашего приложения «Список контактов» более элегантное решение. Для этого я расширил наше приложение из раздела [Список контактов – начало](#). Теперь самое время разобраться, что именно я предлагаю вам посмотреть.

В общем-то в части уже готовых классов я поменял только два класса — **ContactTest** и **ContactSimpleDAO**.

**ContactTest** теперь служит просто для запуска основной формы, которая отображает список контактов. На этом его функциональность заканчивается.

- [Расширенное описание классов](#)
- [Исключения](#)
- [Решения на основе классов](#)
- [Список контактов — начало](#)
- [Коллекции — базовые принципы](#)
- [Коллекции — продолжение](#)
- [Список контактов — GUI приложение](#)
- [Что такое JAR-файлы](#)
- [Многопоточность — первые шаги](#)
- [Многопоточность и синхронизация](#)
- [Работаем с XML](#)
- [Reflection — основы](#)
- [Установка СУБД PostgreSQL](#)
- [Базы данных на Java — первые шаги](#)
- [Возможности JDBC — второй этап](#)
- [JDBC — групповые операции](#)
- [Список контактов — работаем с БД](#)
- [Переезжаем на Maven](#)
- [Потоки ввода-вывода](#)
- [Сетевое взаимодействие](#)
- [С чего начинается Web](#)

```
1 package edu.javacourse.contact.test;
2
3 import edu.javacourse.contact.gui.ContactFrame;
4
5 /**
6  * Класс для запуска тестовых вызовов
7  */
8 public class ContactTest
9 {
10     public static void main(String[] args) {
11         ContactFrame cf = new ContactFrame();
12         cf.setVisible(true);
13     }
14 }
```

Класс **ContactSimpleDAO** я дополнил кодом, который сразу вставляет в мое хранилище три контакта — чтобы не смотреть на пустой список — а то как-то пусто будет на форме. Советую уже более подробно разобрать код этого класса — мы уже познакомились с коллекциями и тут вы можете посмотреть как можно их использовать.

```
1 package edu.javacourse.contact.dao;
2
3 import edu.javacourse.contact.entity.Contact;
4 import java.util.ArrayList;
5 import java.util.Iterator;
6 import java.util.List;
7
8 public final class ContactSimpleDAO implements ContactDAO
9 {
10     private final List<Contact> contacts = new ArrayList<>();
11
12     // Прямо в конструкторе добавляем три контакта
13     public ContactSimpleDAO() {
14         addContact(new Contact("Андрей", "Соколов", "+7-911-890-7766", "sokolov@yandex.ru"));
15         addContact(new Contact("Сергей", "Иванов", "+7-911-890-7755", "ivanov@google.com"));
16         addContact(new Contact("Татьяна", "Семенова", "+7-911-890-1164", "semenova@mail.ru"));
17     }
18
19     @Override
20     public Long addContact(Contact contact) {
21         Long id = generateContactId();
22         contact.setContactId(id);
23         contacts.add(contact);
24         return id;
25     }
26
27     @Override
28     public void updateContact(Contact contact) {
29         Contact oldContact = getContact(contact.getContactId());
30         if (oldContact != null) {
31             oldContact.setFirstName(contact.getFirstName());
32             oldContact.setLastName(contact.getLastName());
33             oldContact.setPhone(contact.getPhone());
34             oldContact.setEmail(contact.getEmail());
35         }
36     }
37
38     @Override
39     public void deleteContact(Long contactId) {
40         for (Iterator<Contact> it = contacts.iterator(); it.hasNext();) {
41             Contact cnt = it.next();
42             if (cnt.getContactId().equals(contactId)) {
43                 it.remove();
44             }
45         }
46     }
47 }
```

```

46     }
47
48     @Override
49     public Contact getContact(Long contactId) {
50         for(Contact contact : contacts) {
51             if(contact.getContactId().equals(contactId)) {
52                 return contact;
53             }
54         }
55         return null;
56     }
57
58     @Override
59     public List<Contact> findContacts() {
60         return contacts;
61     }
62
63     private Long generateContactId() {
64         Long contactId = Math.round(Math.random() * 1000 + System.currentTimeMillis());
65         while(getContact(contactId) != null) {
66             contactId = Math.round(Math.random() * 1000 + System.currentTimeMillis());
67         }
68         return contactId;
69     }
70 }

```

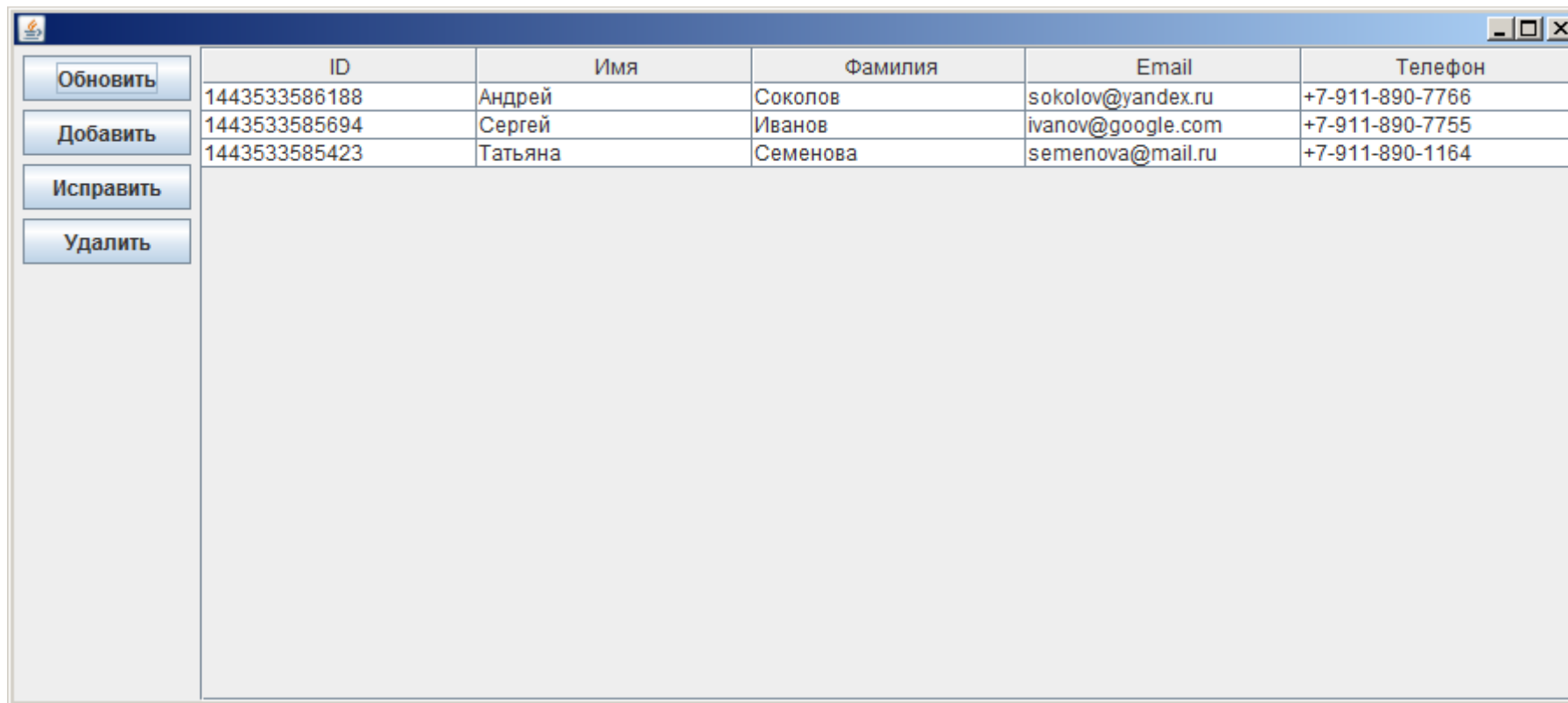
Теперь все наше внимание будет сосредоточено на том, каким образом я сделал графический интерфейс (возможно не идеально — серьезные специалисты по GUI наверняка что-нибудь могут подсказать — я в этой области не имел большого опыта).

Графический интерфейс состоит из трех классов:

1. **ContactFrame** — основная форма для отображения. Содержит кнопки для редактирования списка контактов и отображает эти самые контакты.
2. **EditContactDialog** — диалоговое окно для редактирования данных выделенного контакта. Появляется при нажатии кнопок «Добавить» или «Исправить»
3. **ContactModel** — этот класс предназначен для отображения таблицы контактов. Зачем он нужен — мы узнаем в свое время.

Чтобы наглядно продемонстрировать, что именно у нас получится, предлагаю посмотреть картинки, на которых изображены основная форма и диалоговая форма для ввода данных.

Наша основная форма будет выглядеть вот так:



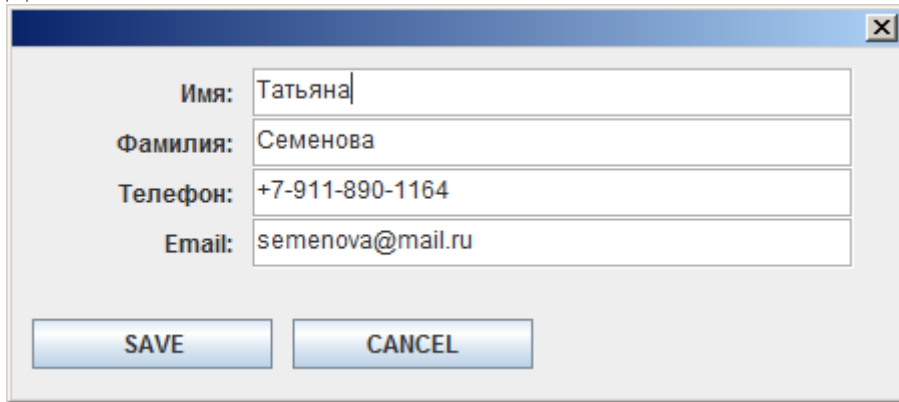
The screenshot shows a web application window with a sidebar on the left containing four buttons: «Обновить», «Добавить», «Исправить», and «Удалить». The main area contains a table with five columns: ID, Имя, Фамилия, Email, and Телефон. The table has three data rows.

ID	Имя	Фамилия	Email	Телефон
1443533586188	Андрей	Соколов	sokolov@yandex.ru	+7-911-890-7766
1443533585694	Сергей	Иванов	ivanov@google.com	+7-911-890-7755
1443533585423	Татьяна	Семенова	semenova@mail.ru	+7-911-890-1164

Назначение кнопок следующее:

- «Обновить» — перегрузить список контактов.
- «Добавить» — открыть диалоговое окно для ввода данных нового контакта и сохранения их в нашем DAO
- «Исправить» — открыть диалоговое окно, загрузить в него данные из выделенной строки, редактировать и сохранить в DAO
- «Удалить» — удалить выделенную запись

Диалог для ввода данных выглядит вот так:



A screenshot of a Java Swing dialog box titled "Contact Model". It contains four text input fields with labels: "Имя:" (Name) with the value "Татьяна", "Фамилия:" (Surname) with the value "Семенова", "Телефон:" (Phone) with the value "+7-911-890-1164", and "Email:" with the value "semenova@mail.ru". At the bottom, there are two buttons: "SAVE" and "CANCEL".

Давайте сначала разберемся с классом **ContactModel**. Приведу его код а потом мы рассмотрим вопросы, связанные с этим классом.

```
1 package edu.javacourse.contact.gui;
2
3 import edu.javacourse.contact.entity.Contact;
4 import java.util.List;
5 import javax.swing.table.AbstractTableModel;
6
7 public class ContactModel extends AbstractTableModel
8 {
9     // Список заголовков для колонок в таблице
10    private static final String[] headers = {"ID", "Имя", "Фамилия", "Email", "Телефон"};
11
12    // Здесь мы храним список контактов, которые будем отображать в таблице
13    private final List<Contact> contacts;
14
15    public ContactModel(List<Contact> contacts) {
16        this.contacts = contacts;
17    }
18
19    @Override
20    // Получить количество строк в таблице - у нас это размер коллекции
21    public int getRowCount() {
22        return contacts.size();
23    }
24
25    @Override
26    // Получить количество столбцов - их у нас столько же, сколько полей
27    // у класса Contact - всего пять
28    public int getColumnCount() {
```

```

29         return 5;
30     }
31
32     @Override
33     // Вернуть заголовок колонки - мы его берем из массива headers
34     public String getColumnName(int col) {
35         return headers[col];
36     }
37
38     @Override
39     // Получить объект для отображения в конкретной ячейке таблицы
40     // В данном случае мы отдаем строковое представление поля
41     public Object getValueAt(int row, int col) {
42         Contact contact = contacts.get(row);
43         // В зависимости от номера колонки возвращаем то или иное поле контакта
44         switch (col) {
45             case 0:
46                 return contact.getId().toString();
47             case 1:
48                 return contact.getFirstName();
49             case 2:
50                 return contact.getLastName();
51             case 3:
52                 return contact.getEmail();
53             default:
54                 return contact.getPhone();
55         }
56     }
57 }

```

Появление этого класса связано с достаточно известным шаблоном проектирования — MVC (Model, View, Controller) — Модель, Отображение, Контроллер (Управление). В интернете можно найти много картинок и текста, связанных с этим шаблоном, но тем не менее я опишу своими словами свое понимание этого шаблона.

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 может быть и не так — например в нашем приложении добавление, изменение и удаление будет производиться 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 отображение 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 со стандартным классом **JTable**.

В нашем примере мы не используем возможности редактирования модели — просто при загрузке контактов создаем новую модель и отдаем ее таблице.

## Класс ContactFrame

Наш класс **ContactFrame** — я сделал комментарии в коде, которые должны помочь вам разобраться.

```
1 package edu.javacourse.contact.gui;
2
3 import edu.javacourse.contact.business.ContactManager;
4 import edu.javacourse.contact.entity.Contact;
5 import java.awt.BorderLayout;
6 import java.awt.GridBagConstraints;
7 import java.awt.GridBagLayout;
8 import java.awt.Insets;
9 import java.awt.event.ActionEvent;
10 import java.awt.event.ActionListener;
11 import java.util.List;
12 import javax.swing.JButton;
13 import javax.swing.JFrame;
14 import javax.swing.JOptionPane;
15 import javax.swing.JPanel;
16 import javax.swing.JScrollPane;
17 import javax.swing.JTable;
18 import javax.swing.ListSelectionModel;
19
20 public class ContactFrame extends JFrame implements ActionListener
21 {
22     private static final String LOAD = "LOAD";
23     private static final String ADD = "ADD";
24     private static final String EDIT = "EDIT";
25     private static final String DELETE = "DELETE";
```



```

26
27 private final ContactManager contactManager = new ContactManager();
28 private final JTable contactTable = new JTable();
29
30 // В конструкторе мы создаем нужные элементы
31 public ContactFrame() {
32     // Выставляем у таблицы свойство, которое позволяет выделить
33     // ТОЛЬКО одну строку в таблице
34     contactTable.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECTION);
35
36     // Используем layout manager
37     GridBagLayout gridbag = new GridBagLayout();
38     GridBagConstraints gbc = new GridBagConstraints();
39     // Каждый элемент является последним в строке
40     gbc.gridwidth = GridBagConstraints.REMAINDER;
41     // Элемент раздвигается на весь размер ячейки
42     gbc.fill = GridBagConstraints.BOTH;
43     // Но имеет границы - слева, сверху и справа по 5. Снизу - 0
44     gbc.insets = new Insets(5, 5, 0, 5);
45
46     // Создаем панель для кнопок
47     JPanel btnPanel = new JPanel();
48     // усанавливаем у него layout
49     btnPanel.setLayout(gridbag);
50     // Создаем кнопки
51     btnPanel.add(createButton(gridbag, gbc, "Обновить", LOAD));
52     btnPanel.add(createButton(gridbag, gbc, "Добавить", ADD));
53     btnPanel.add(createButton(gridbag, gbc, "Исправить", EDIT));
54     btnPanel.add(createButton(gridbag, gbc, "Удалить", DELETE));
55
56     // Создаем панель для левой колодки с кнопками
57     JPanel left = new JPanel();
58     // Выставляем layout BorderLayout
59     left.setLayout(new BorderLayout());
60     // Кладем панель с кнопками в верхнюю часть
61     left.add(btnPanel, BorderLayout.NORTH);
62     // Кладем панель для левой колонки на форму слева - WEST
63     add(left, BorderLayout.WEST);
64
65     // Кладем панель со скролингом, внутри которой нахоится наша таблица
66     // Теперь таблица может скроллиться
67     add(new JScrollPane(contactTable), BorderLayout.CENTER);
68
69     // выставляем координаты формы
70     setBounds(100, 200, 900, 400);
71     // При закрытии формы заканчиваем работу приложения
72     setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

```

```
73
74     // Загружаем контакты
75     loadContact();
76 }
77
78 // Метод создает кнопку с заданными характеристиками - заголовок и действие
79 private JButton createButton(GridBagLayout gridbag, GridBagConstraints gbc, String title, String action) {
80     // Создаем кнопку с заданным заголовком
81     JButton button = new JButton(title);
82     // Действие будет проверяться в обработчике и мы будем знать, какую
83     // именно кнопку нажали
84     button.setActionCommand(action);
85     // Обработчиком события от кнопки являемся сама форма
86     button.addActionListener(this);
87     // Выставляем свойства для размещения для кнопки
88     gridbag.setConstraints(button, gbc);
89     return button;
90 }
91
92 @Override
93 // Обработка нажатий кнопок
94 public void actionPerformed(ActionEvent ae) {
95     // Получаем команду - ActionCommand
96     String action = ae.getActionCommand();
97     // В зависимости от команды выполняем действия
98     switch (action) {
99         // Перегрузка данных
100         case LOAD:
101             loadContact();
102             break;
103         // Добавление записи
104         case ADD:
105             addContact();
106             break;
107         // Исправление записи
108         case EDIT:
109             editContact();
110             break;
111         // Удаление записи
112         case DELETE:
113             deleteContact();
114             break;
115     }
116 }
117
118 // Загрузить список контактов
119 private void loadContact() {
```

```

120 // Обращаемся к классу для загрузки списка контактов
121 List<Contact> contacts = contactManager.findContacts();
122 // Создаем модель, которой передаем полученный список
123 ContactModel cm = new ContactModel(contacts);
124 // Передаем нашу модель таблице - и она может ее отображать
125 contactTable.setModel(cm);
126 }
127
128 // Добавление контакта
129 private void addContact() {
130     // Создаем диалог для ввода данных
131     EditContactDialog ecd = new EditContactDialog();
132     // Обрабатываем закрытие диалога
133     saveContact(ecd);
134 }
135
136 // Редактирование контакта
137 private void editContact() {
138     // Получаем выделенную строку
139     int sr = contactTable.getSelectedRow();
140     // если строка выделена - можно ее редактировать
141     if (sr != -1) {
142         // Получаем ID контакта
143         Long id = Long.parseLong(contactTable.getModel().getValueAt(sr, 0).toString());
144         // получаем данные контакта по его ID
145         Contact cnt = contactManager.getContact(id);
146         // Создаем диалог для ввода данных и передаем туда контакт
147         EditContactDialog ecd = new EditContactDialog(cnt);
148         // Обрабатываем закрытие диалога
149         saveContact(ecd);
150     } else {
151         // Если строка не выделена - сообщаем об этом
152         JOptionPane.showMessageDialog(this, "Вы должны выделить строку для редактирования");
153     }
154 }
155
156 // Удаление контакта
157 private void deleteContact() {
158     // Получаем выделенную строку
159     int sr = contactTable.getSelectedRow();
160     if (sr != -1) {
161         // Получаем ID контакта
162         Long id = Long.parseLong(contactTable.getModel().getValueAt(sr, 0).toString());
163         // Удаляем контакт
164         contactManager.deleteContact(id);
165         // перегружаем список контактов
166         loadContact();

```

```

167     } else {
168         JOptionPane.showMessageDialog(this, "Вы должны выделить строку для удаления");
169     }
170 }
171
172 // Общий метод для добавления и изменения контакта
173 private void saveContact(EditContactDialog ecd) {
174     // Если мы нажали кнопку SAVE
175     if (ecd.isSave()) {
176         // Получаем контакт из диалогового окна
177         Contact cnt = ecd.getContact();
178         if (cnt.getId() != null) {
179             // Если ID у контакта есть, то мы его обновляем
180             contactManager.updateContact(cnt);
181         } else {
182             // Если у контакта нет ID - значит он новый и мы его добавляем
183             contactManager.addContact(cnt);
184         }
185         loadContact();
186     }
187 }
188 }

```

Давайте рассмотрим важные (на мой взгляд) моменты. Изучение и анализ кода я бы советовал начинать с двух частей:

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

## Конструктор ContactFrame

Наш конструктор должен создать необходимые элементы нашей яормы — вот и будем смотреть, как это происходит. На рисунке нашей формы мы видим две области — слева область кнопок, справа — таблица с данными.

Прежде, чем вы станете читать дальше, я вам настоятельно рекомендую посмотреть статью [Что такое LayoutManager](#). В ней вы найдете много информации, которая поможет вам понять, что и как мы делаем.

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 управления. Чтобы панель с кнопками не «расползалась» на всю левую сторону, я сначала «кладу» ее на другую панель **left** (у которой выставлю **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 прокручивать элемент внутри себя и уже его кладем на форму в Центр.

Нажатия от кнопок будут обрабатываться нашей формой — пример такой обработки мы уже рассматривали в разделе [Интерфейсы](#). Для этого наша форма реализует интерфейс **ActionListener**.

## Обработка кнопок

Все кнопки вызывают метод **actionPerformed** и передают туда объект класса **ActionEvent**. Нас в этом объекте интересует метод **getActionCommand()**. При создании кнопок мы каждой «выдали» определенное значение, по которому мы теперь и сможем понять, какая именно кнопка была нажата и какая послала нам сообщение. Дальше достаточно просто аккуратно пройти по шагам и вы сами увидите, что мы для каждой кнопки вызываем отдельный метод, который выполняет нужную функцию.

Перегрузка (метод **loadContact**) и удаление (метод **deleteContact**) достаточно простые и думаю, что будет достаточно просто посмотреть комментарии (и возможно глянуть документацию по классу **JTable**).

Что же касается добавления и редактирования (методы **addContact** и **editContact**), то они в общем тоже не представляют проблемы — в них мы вызываем диалоговое окно. Но при редактировании мы передаем в это окно выделенный контакт, чтобы заполнить поля в диалоге.

## Класс ContactDialog

Думаю, что вы уже сможете разобраться в коде самостоятельно. Но кое-какие моменты хотелось бы обозначить.

1. Точно так же есть смысл смотреть две «входные точки» — конструктор, где мы строим все элементы и обработчик нажатия кнопок
2. Я сделал ДВА конструктора — один является основным и выполняет все настройки а также проверяет, что если контакт передали, то надо заполнить поля и (ЧТО ВАЖНО) присваивает ID контакта переменной **contactId**. Второй сделан исключительно для красоты — и вызывается тогда, когда мы создаем новый контакт. В принципе можно было обойтись одним конструктором — просто передавать в него **null**
3. Это есть в статье по **LayoutManager** — мы полностью отключаем **LayoutManager** и это позволяет нам размещать элементы жестко по координатам. В этом есть смысл, т.к. диалог не меняет размер и ничего страшного в абсолютных координатах в данном случае я не вижу.
4. Механизм возврата введенных данных через метод **getContact** — я сделал его в таком виде. Хотя это не значит, что нельзя сделать иначе (можете подумать и поискать иные варианты). Замечу, что мы создаем контакт и передаем туда переменную **contactId**. Если это новый контакт — значит она будет равна **null** и мы можем считать, что то новый контакт. Если же там есть какое-то число — значит это существующий контакт и мы должны его обновить

```
1 package edu.javacourse.contact.gui;
2
3 import edu.javacourse.contact.entity.Contact;
4 import java.awt.Rectangle;
5 import java.awt.event.ActionEvent;
6 import java.awt.event.ActionListener;
7 import javax.swing.BorderFactory;
8 import javax.swing.JButton;
9 import javax.swing.JDialog;
10 import javax.swing.JLabel;
```

```
11 import javax.swing.JTextPane;
12 import javax.swing.SwingConstants;
13
14 public class EditContactDialog extends JDialog implements ActionListener
15 {
16     // Заголовки кнопок
17     private static final String SAVE = "SAVE";
18     private static final String CANCEL = "CANCEL";
19
20     // Размер отступа
21     private static final int PAD = 10;
22     // Ширина метки
23     private static final int W_L = 100;
24     //Ширина поля для ввода
25     private static final int W_T = 300;
26     // Ширина кнопки
27     private static final int W_B = 120;
28     // высота элемента - общая для всех
29     private static final int H_B = 25;
30
31     // Поле для ввода Имени
32     private final JTextPane txtFirstName = new JTextPane();
33     // Поле для ввода Фамилии
34     private final JTextPane txtLastName = new JTextPane();
35     // Поле для ввода Телефона
36     private final JTextPane txtPhone = new JTextPane();
37     // Поле для ввода E-mail
38     private final JTextPane txtEmail = new JTextPane();
39
40     // Поле для хранения ID контакта, если мы собираемся редактировать
41     // Если это новый контакт - cJntactId == null
42     private Long contactId = null;
43     // Надо ли записывать изменения после закрытия диалога
44     private boolean save = false;
45
46     public EditContactDialog() {
47         this(null);
48     }
49
50     public EditContactDialog(Contact contact) {
51         // Убираем layout - будем использовать абсолютные координаты
52         setLayout(null);
53
54         // Выстраиваем метки и поля для ввода
55         buildFields();
56         // Если нам передали контакт - заполняем поля формы
57         initFields(contact);
```

```
58     // выстариваем кнопки
59     buildButtons();
60
61     // Диалог в модальном режиме - только он активен
62     setModal(true);
63     // Запрещаем изменение размеров
64     setResizable(false);
65     // Выставляем размеры формы
66     setBounds(300, 300, 450, 200);
67     // Делаем форму видимой
68     setVisible(true);
69 }
70
71 // Размещаем метки и поля ввода на форме
72 private void buildFields() {
73     // Набор метки и поля для Имени
74     JLabel lblFirstName = new JLabel("Имя:");
75     // Выравнивание текста с правой стороны
76     lblFirstName.setHorizontalAlignment(SwingConstants.RIGHT);
77     // Выставляем координаты метки
78     lblFirstName.setBounds(new Rectangle(PAD, 0 * H_B + PAD, W_L, H_B));
79     // Кладем метку на форму
80     add(lblFirstName);
81     // Выставляем координаты поля
82     txtFirstName.setBounds(new Rectangle(W_L + 2 * PAD, 0 * H_B + PAD, W_T, H_B));
83     // Делаем "бorders" для поля
84     txtFirstName.setBorder(BorderFactory.createEtchedBorder());
85     // Кладем поле на форму
86     add(txtFirstName);
87
88     // Набор метки и поля для Фамилии
89     JLabel lblLastName = new JLabel("Фамилия:");
90     lblLastName.setHorizontalAlignment(SwingConstants.RIGHT);
91     lblLastName.setBounds(new Rectangle(PAD, 1 * H_B + PAD, W_L, H_B));
92     add(lblLastName);
93     txtLastName.setBounds(new Rectangle(W_L + 2 * PAD, 1 * H_B + PAD, W_T, H_B));
94     txtLastName.setBorder(BorderFactory.createEtchedBorder());
95     add(txtLastName);
96
97     // Набор метки и поля для Телефона
98     JLabel lblPhone = new JLabel("Телефон:");
99     lblPhone.setHorizontalAlignment(SwingConstants.RIGHT);
100    lblPhone.setBounds(new Rectangle(PAD, 2 * H_B + PAD, W_L, H_B));
101    add(lblPhone);
102    txtPhone.setBounds(new Rectangle(W_L + 2 * PAD, 2 * H_B + PAD, W_T, H_B));
103    txtPhone.setBorder(BorderFactory.createEtchedBorder());
104    add(txtPhone);
```

```
105
106 // Набор метки и поля для Email
107 JLabel lblEmail = new JLabel("Email:");
108 lblEmail.setHorizontalAlignment(SwingConstants.RIGHT);
109 lblEmail.setBounds(new Rectangle(PAD, 3 * H_B + PAD, W_L, H_B));
110 add(lblEmail);
111 txtEmail.setBounds(new Rectangle(W_L + 2 * PAD, 3 * H_B + PAD, W_T, H_B));
112 txtEmail.setBorder(BorderFactory.createEtchedBorder());
113 add(txtEmail);
114 }
115
116 // Если нам епередали контакт - заполняем поля из контакта
117 private void initFields(Contact contact) {
118     if (contact != null) {
119         contactId = contact.getId();
120         txtFirstName.setText(contact.getFirstName());
121         txtLastName.setText(contact.getLastName());
122         txtEmail.setText(contact.getEmail());
123         txtPhone.setText(contact.getPhone());
124     }
125 }
126
127 // Размещаем кнопки на форме
128 private void buildButtons() {
129     JButton btnSave = new JButton("SAVE");
130     btnSave.setActionCommand(SAVE);
131     btnSave.addActionListener(this);
132     btnSave.setBounds(new Rectangle(PAD, 5 * H_B + PAD, W_B, H_B));
133     add(btnSave);
134
135     JButton btnCancel = new JButton("CANCEL");
136     btnCancel.setActionCommand(CANCEL);
137     btnCancel.addActionListener(this);
138     btnCancel.setBounds(new Rectangle(W_B + 2 * PAD, 5 * H_B + PAD, W_B, H_B));
139     add(btnCancel);
140 }
141
142 @Override
143 // Обработка нажатий кнопок
144 public void actionPerformed(ActionEvent ae) {
145     String action = ae.getActionCommand();
146     // Если нажали кнопку SAVE (сохранить изменения) - запоминаем этой
147     save = SAVE.equals(action);
148     // Закрываем форму
149     setVisible(false);
150 }
151
```



```

152 // Надо ли сохранять изменения
153 public boolean isSave() {
154     return save;
155 }
156
157 // Создаем контакт из заполненных полей, который можно будет записать
158 public Contact getContact() {
159     Contact contact = new Contact(contactId, txtFirstName.getText(),
160         txtLastName.getText(), txtPhone.getText(), txtEmail.getText());
161     return contact;
162 }
163 }

```

Полный код примера можно скачать отсюда — [ContactProject\\_02.zip](#)

## Домашнее задание

1. Разобраться в работе примера и сделать несложные изменения — перенести в основной форме кнопки на правую сторону и вниз.
2. Сделать мультязычную версию — это сложное задание. С моей точки зрения, надо сделать отдельный класс, который будет загружать ресурс и обладать набором методов для предоставления нужных данных.

Удачи.

И теперь нас ждет следующая статья: [Что такое JAR-файлы](#).

## 9 comments to *Список контактов — GUI приложение*



- Апрель 1, 2018 at 21:13  
Максим says:

Антон, спасибо Вам за учебные материалы, как видео, так и статьи.

1. В комментарии к методу «public Object getValueAt(int row, int col)» класса «ContactModel extends AbstractTableModel» Вы пишете о том, что мы отдаем строковое представление (однако приведения типов мы почему-то не делали). В то же время когда я пытаюсь вставить туда некий объект (не String), то в выведенной таблице объект отображается в виде «gu.package.NameOfClass@40bad95c». Выходит по умолчанию в ячейках таблицы делается toString()? Возможно ли в ячейках таблицы выводить объекты в графическом (не строковом) виде.
2. Возможно ли вставить таблицу в таблицу (чтобы строка разделилась на 2-3)?

[Reply](#)



o

Апрель 1, 2018 at 22:39

*admin* says:

1. Метод отдает тип Object, но когда он отображается в ячейке, то приводится к строке через toString(). Соответственно, мы и отдаем строковое представление для каждой колонки.
2. Можно «вручную» рисовать внутри ячейки или вставлять свой компонент туда — но я не сильно в этом разобрался, посему не скажу точно что и как.

[Reply](#)



•

Апрель 1, 2018 at 21:33

*Максим* says:

Видимо для реализации моего страстного желания разделения строк, косвенным образом, должен помочь метод «void setValueAt(Object aValue, int rowIndex, int columnIndex)» и двойное употребление цикла forEach (в первом цикле — для внешней коллекции, во втором — для внутреннего списка).

[Reply](#)



o

Апрель 1, 2018 at 22:41

*admin* says:

Я уже ответил, что в графической библиотеке я сильно не разобрался.

[Reply](#)



•

Апрель 4, 2018 at 20:13

Максим says:

Относительно разделения строк разобрался. Для этого пришлось написать:

1. Метод для определения однозначным образом конкретного элемента внешнего списка по строке таблицы row.
2. Метод для доступа к строке вложенного списка, учитывая впереди идущие члены внешнего списка, также по строке таблицы row.

[Reply](#)



Май 4, 2018 at 12:50

Дмитрий says:

У меня есть вопрос и пара замечаний.

1. В классе ContactFrame в строке 143, где мы извлекаем из ячейки значение:

```
1 | ID Long id = Long.parseLong(contactTable.getModel().getValueAt(sr, 0).toString());
```

зачем нам getModel()? Без него тоже работает, в чём разница?

2. Там же, две следующие строчки:

```
1 | Contact cnt = contactManager.getContact(id);  
2 | EditContactDialog ecd = new EditContactDialog(contactManager.getContact(id));
```

Мы создаём переменную Contact, но в конструктор для ecd передаём анонимный объект. Так что либо избыточная переменная, либо её передавать в конструктор.

Ещё замечаю много опечаток в тексте, есть ли какая-нибудь возможность Вам сообщать о них? Не хочется этим забивать комментарии.

[Reply](#)



o

Май 4, 2018 at 14:19

admin says:

Ла, это ошибочка — исправил. Спасибо.

По поводу опечаток — бывает, не всегда удастся отследить. Можно присылать на адрес: [webinar@java-course.ru](mailto:webinar@java-course.ru)

[Reply](#)



Июнь 24, 2018 at 20:15

*Денис* says:

Добрый день.

Можете подсказать по домашнему заданию?

Я заменил строгие названия кнопок на переменные и передаю в эти переменные данные (русский, английский язык) но перерисовать кнопки с новыми данными в `ContactFrame` не могу.

Как можно перерисовать кнопки с новыми данными?

[Reply](#)



Июнь 30, 2018 at 03:00

*admin* says:

Не очень понимаю идею, но наверно надо вызвать у формы `repaint()`. Хотя как я у еж неоднократно писал, графическое приложение написано просто для того, чтобы было не так скучно. Я крайне неглубоко погружался в Swing. Так что поищите информацию еще где-нибудь.

[Reply](#)

**Leave a reply**

Comment

You may use these HTML tags and attributes: <a href="" title=""> <abbr title=""> <acronym title=""> <b> <blockquote cite=""> <cite> <code class="" title="" data-url=""> <del datetime=""> <em> <i> <q cite=""> <s> <strike> <strong> <pre class="" title="" data-url=""> <span class="" title="" data-url="">

Имя \*

E-mail \*

Сайт

7 ×  = 49 

Copyright © 2018 [Java Course](#)

Designed by [Blog templates](#), thanks to: [Free WordPress themes for photographers](#), [LizardThemes.com](#) and [Free WordPress real estate themes](#)

