

Программирование на Java

10. Модель-представление

Глухих Михаил Игоревич
mailto: glukhikh@mail.ru

Задача–пример

- ▶ Необходимо разработать редактор схем коммивояжера
- ▶ Показывающий города с их названиями
- ▶ И существующие пути между ними (вид транспорта, стоимость, время в пути)

Объектная модель

- ▶ Город (City)
 - название, координаты
- ▶ Путь (Way)
 - начало, конец, стоимость, время
 - тип (kind) – автобус, поезд, самолет
- ▶ Мир (World)
 - города и пути

Класс-город и класс-путь

```
public class City implements Serializable {  
    private String name;  
    private int x, y;  
    public City(String name, int x, int y) {  
        // ...  
    }  
}  
  
public class Way implements Serializable {  
    public enum WayKind {  
        BUS, TRAIN, AIRCRAFT;  
    };  
    private City start, finish;  
    private WayKind kind;  
    private int cost, time;  
}
```

Класс-мир

```
public class World implements Serializable {  
    private List<City> cities;  
    private List<Way> ways;  
    public World() {  
        cities = new ArrayList<City>();  
        ways = new ArrayList<Way>();  
    }  
}
```

Что такое Serializable

- ▶ Поддержка этого интерфейса обеспечивает возможность сохранения объекта в поток (в виде набора байт) и загрузки его из потока
- ▶ Может использоваться
 - для сохранения в файл/загрузки из файла
 - для передачи по сети
 - ...

Сохранение/загрузка

► Сохранение мира (world)

```
ObjectOutputStream outputStream =  
    new ObjectOutputStream(  
        new FileOutputStream(file));  
outputStream.writeObject(world);
```

► Загрузка мира (world)

```
ObjectInputStream inputStream =  
    new ObjectInputStream(  
        new FileInputStream(file));  
world = (World)inputStream.readObject();
```

Способы сохранения/загрузки

- ▶ Способ 1 (самый простой в реализации) – поддержка интерфейса Serializable
- ▶ Обратите внимание – "самый простой в реализации" не означает "самый простой в поддержке"!
- ▶ Достаточно добавить в класс данное-член, или добавить не закрытую функцию, как файлы, сформированные старой версией, перестанут загружаться

Почему возникают проблемы при восстановлении?

- ▶ При сохранении объекта записывается также `SerialVersionUID`
- ▶ Он формируется из всех данных (кроме тех, которые помечены как `transient`), имен функций (кроме `private`), статических членов (кроме `private`)
- ▶ Если мы меняем класс, `SerialVersionUID` меняется

Вывод по Serializable

- ▶ Если вы решаете, что некоторый объект реализует этот интерфейс, тем самым вы соглашаетесь более не добавлять в него членов-данных (кроме **transient**) и методов (кроме **private**)

Способы сохранения/загрузки

- ▶ Способ 2 (несколько более сложный) – поддержка интерфейса Externalizable

- ▶ Необходимо реализовать два метода

```
public void writeExternal(ObjectOutput out)
    throws IOException {}

public void readExternal(ObjectInput in)
    throws IOException, ClassNotFoundException {}
```

- ▶ В первом из них мы определяем, как записать объект в поток
- ▶ А во втором – как прочитать объект из потока

Пример (для класса City)

writeExternal/readExternal

```
public void writeExternal(ObjectOutput out)
    throws IOException {
    out.writeInt(1); // версия
    out.writeUTF(name);
    out.writeInt(x);
    out.writeInt(y);
}

public void readExternal(ObjectInput in)
    throws IOException, ClassNotFoundException {
    final int version = in.readInt();
    if (version > 1) throw new IOException("...");
    name = in.readUTF();
    x = in.readInt();
    y = in.readInt();
}
```

Пример (для класса World)

writeExternal

```
public void writeExternal(ObjectOutput out)
    throws IOException {
    out.writeInt(1);
    out.writeInt(cities.size()); // Города
    for (City city: cities) city.writeExternal(out);
    out.writeInt(ways.size()); // Пути
    for (Way way: ways) {
        way.writeExternal(out);
        final int startIndex = cities.indexOf(way.getStart());
        out.writeInt(startIndex);
        final int finishIndex=cities.indexOf(way.getFinish());
        out.writeInt(finishIndex);
    }
}
```

Пример (для класса World) readExternal

```
public void readExternal(ObjectInput in)
    throws IOException, ClassNotFoundException {
    final int version = in.readInt();
    if (version > 1) throw new IOException("...");
    cities.clear();
    final int cityNum = in.readInt();
    for (int i=0; i<cityNum; i++) {
        final City city = new City("", 0, 0);
        city.readExternal(in);
        cities.add(city);
    }
    // ...
}
```

Пример (для класса World) readExternal

```
public void readExternal(ObjectInput in)
    throws IOException, ClassNotFoundException {
    // ...
    ways.clear();
    final int wayNum = in.readInt();
    for (int i=0; i<wayNum; i++) {
        final Way way = new Way(null, null,
                                WayKind.BUS, 0, 0);

        way.readExternal(in);
        way.setStart(cities.get(in.readInt()));
        way.setFinish(cities.get(in.readInt()));
        ways.add(way);
    }
}
```

Достоинства writeExternal/readExternal

- ▶ Мы сами определяем, как сохранять наши объекты
- ▶ При помощи поля "версия" мы можем следить за изменениями версии объекта (обеспечивается обратная совместимость, то есть более новые версии программы могут прочитать файлы, созданные старыми версиями)

Недостатки writeExternal/readExternal

- ▶ Содержимое сохраненного файла (см. пример) абсолютно непонятно человеку; если произойдет какая-либо ошибка, понять, что произошло, достаточно сложно

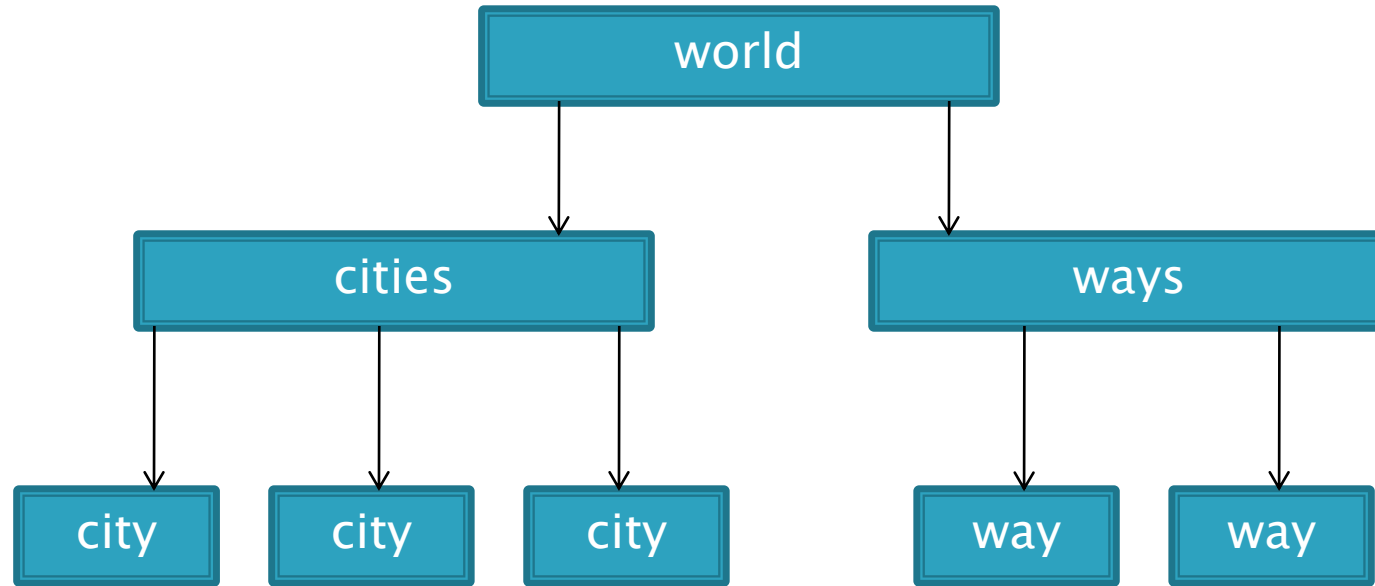
Способы сохранения/загрузки

- ▶ Способ 3 – использование универсальных форматов
- ▶ Примеры таких форматов – XML, JSON, INI
- ▶ Общее их свойство – все они текстовые, то есть читаемые

Пример файла в формате XML (eXtensible Markup Language)

```
<?xml version="1.0" encoding="UTF-8" ?>
<world>
<cities>
  <city name="Питер" x="48" y="98" />
  <city name="Москва" x="286" y="70" />
  <city name="Киев" x="196" y="267" />
</cities><ways>
<way kind="Автобус" time="10" cost="500" start="0" finish="1"/>
<way kind="Поезд" time="5" cost="2000" start="0" finish="1"/>
<way kind="Самолет" time="2" cost="4600" start="0" finish="1"/>
<way kind="Поезд" time="4" cost="2100" start="1" finish="2"/>
<way kind="Самолет" time="4" cost="6000" start="0" finish="2"/>
</ways></world>
```

Структура XML



Как работать с XML на Java?

- ▶ Есть много способов. Один из них – открытая библиотека JDOM (см. www.jdom.org)
- ▶ Основные используемые понятия
 - `org.jdom.Document` – документ XML
 - `org.jdom.Element` – один из элементов документа, например, `<city>...</city>` или `<world>...</world>`
 - `org.jdom.Attribute` – атрибут одного из элементов, например, `cost`

Создание элемента для города

```
public Element getXMLElement() {  
    final Element root = new Element("city");  
    root.setAttribute("name", name);  
    root.setAttribute("x", String.valueOf(x));  
    root.setAttribute("y", String.valueOf(y));  
    return root;  
}
```

Сохранение в XML

```
public void writeXML(final File file) throws IOException {  
    final Element citiesRoot = new Element("cities");  
    final List<Element> cityElements =  
        new LinkedList<Element>();  
    for (City city : cities)  
        cityElements.add(city.getXMLElement());  
    citiesRoot.setContent(cityElements);  
}
```

Сохранение в XML

```
public void writeXML(final File file) throws IOException {  
    final Element waysRoot = new Element("ways");  
    final List<Element> wayElements = new LinkedList<Element>();  
    for (Way way: ways) {  
        final Element wayElement = way.getXMLElement();  
        final int startIndex = cities.indexOf(way.getStart());  
        wayElement.setAttribute("start",  
                                String.valueOf(startIndex));  
        final int finishIndex = cities.indexOf(way.getFinish());  
        wayElement.setAttribute("finish",  
                                String.valueOf(finishIndex));  
        wayElements.add(wayElement);  
    }  
    waysRoot.setContent(wayElements);  
}
```


Сохранение в XML

```
public void writeXML(final File file) throws IOException {  
    final Element root = new Element("world");  
    // ...  
    root.addContent(citiesRoot);  
    root.addContent(waysRoot);  
    final Document document = new Document(root);  
    final XMLOutputter outputter = new XMLOutputter();  
    outputter.output(document, new FileOutputStream(file));  
}
```

Разбор элемента для города

```
public static City readXML(Element element) {  
    return new City(  
        element.getAttributeValue("name"),  
        Integer.parseInt(  
            element.getAttributeValue("x")),  
        Integer.parseInt(  
            element.getAttributeValue("y")));  
}
```

Загрузка из XML

```
public void readXML(final File file)
    throws IOException, JDOMException {
    // SAX - Simple API for XML parsing
    final SAXBuilder builder = new SAXBuilder();
    final Document document = builder.build(file);
    final Element root = document.getRootElement();
    cities.clear();
    final Element citiesRoot = root.getChild("cities");
    for (Object obj: citiesRoot.getChildren()) {
        cities.add(City.readXML((Element) obj));
    }
    // ...
}
```

Загрузка из XML

```
public void readXML(final File file)
    throws IOException, JDOMException {
    ways.clear();
    final Element waysRoot = root.getChild("ways");
    for (Object obj: waysRoot.getChildren()) {
        final Element wayElem = (Element)obj;
        final Way way = Way.readXML(wayElem);
        way.setStart(cities.get(Integer.parseInt(
            wayElem.getAttributeValue("start"))));
        way.setFinish(cities.get(Integer.parseInt(
            wayElem.getAttributeValue("finish"))));
        ways.add(way);
    }
}
```

Интерфейс

- ▶ Меню: сохранение, загрузка, выход, добавление объектов
- ▶ Панель инструментов
- ▶ Главная панель для отображения мира
- ▶ Малая панель для отображения свойств объекта
- ▶ Панель статуса для отображения информационных сообщений

Интерфейс

- ▶ См. внешний вид

Иерархия компонентов

- ▶ VoyagerFrame
 - JMenuBar menuBar
 - JMenu
 - JMenuItem
 - JRadioButtonMenuItem
 - contentPane (BorderLayout)
 - JToolBar toolbar (NORTH)
 - JButton
 - JPanel statusBar (SOUTH)
 - JSplitPane splitPane (CENTER)
 - JScrollPane scrollPane
 - VoyagerPanel mainPanel
 - JPanel infoPanel

Создание меню

```
private void initMenuBar() {  
    menuBar = new JMenuBar();  
    this.setJMenuBar(menuBar);  
    fileMenu = new JMenu("Файл");  
    menuBar.add(fileMenu);  
    openMenu = new JMenuItem("Открыть");  
    openMenu.addActionListener(openListener);  
    fileMenu.add(openMenu);  
    saveMenu = new JMenuItem("Сохранить");  
    saveMenu.addActionListener(saveListener);  
    fileMenu.add(saveMenu);  
    fileMenu.addSeparator();  
    // ...  
}
```


Создание меню

```
private void initMenuBar() {  
    // ...  
    modeMenu = new JMenu("Режим");  
    modeGroup = new ButtonGroup();  
    selectMenu = new JRadioButtonMenuItem("Выбор");  
    selectMenu.setSelected(true);  
    selectMenu.addActionListener(selectListener);  
    modeMenu.add(selectMenu);  
    modeGroup.add(selectMenu);  
    // ...  
    menuBar.add(modeMenu);  
}
```

Создание панели инструментов

- ▶ Традиционно, под панель инструментов выделяется северная часть BorderLayout
- ▶ Хотя она может сдвигаться и к другим границам

Создание панели инструментов

```
private void initToolBar() {  
    toolbar = new JToolBar();  
    toolbar.addSeparator();  
    openButton = new JButton(new  
        ImageIcon("open.png"));  
    openButton.addActionListener(openListener);  
    toolbar.add(openButton);  
    saveButton = new JButton(new  
        ImageIcon("save.png"));  
    saveButton.addActionListener(saveListener);  
    toolbar.add(saveButton);  
    // ...  
    this.add(toolbar, BorderLayout.NORTH);  
}
```

Создание слушателей

```
private void initListeners() {  
    addCityListener = e -> onAddCity();  
    // ...  
    quitListener = e -> onQuit();  
}
```

Обработчики изменения режима

```
private void onSelect() {  
    statusLabel.setText("Режим выбора");  
    mainPanel.chooseSelectMode();  
}
```

```
private void onAddCity() {  
    statusLabel.setText("Режим добавления города");  
    mainPanel.chooseCityMode();  
}
```

```
private void onAddWay() {  
    statusLabel.setText("Режим добавления пути");  
    mainPanel.chooseWayMode();  
}
```

Создание панели статуса

- ▶ В библиотеке Swing для панели статуса нет собственного класса
- ▶ Поэтому мы используем обычный JPanel
- ▶ И разместим на нем JLabel для вывода сообщений

Создание панели статуса

```
private void initStatusBar() {  
    statusBar = new JPanel();  
    statusBar.setPreferredSize(new Dimension(500,  
25));  
    statusBar.setBorder(new  
        BevelBorder(BevelBorder.LOWERED));  
    statusBar.setLayout(new  
        FlowLayout(FlowLayout.LEFT));  
    statusLabel = new JLabel("Режим выбора");  
    statusBar.add(statusLabel);  
    this.add(statusBar, BorderLayout.SOUTH);  
}
```

Создание главной панели

```
private void initMainPanel() {  
    voyagerPanel = new VoyagerPanel();  
    voyagerPanel.setBackground(  
        new Color(0, 0, 64));  
    // Задаем размер 1000 x 1000  
    voyagerPanel.setPreferredSize(  
        new Dimension(1000, 1000));  
    // Задаем «утопленную» рамку  
    voyagerPanel.setBorder(  
        new BevelBorder(  
            BevelBorder.LOWERED));  
}
```


Создание панели прокрутки

- ▶ Панель прокрутки используется, чтобы показать большой контейнер на меньшем участке экрана

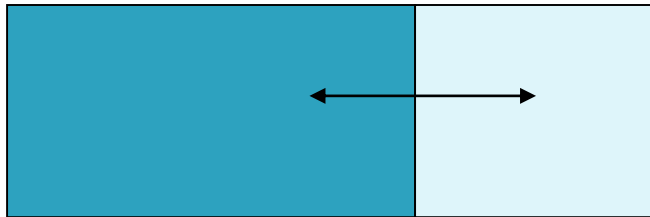
```
JScrollPane scrollPanel =  
    new JScrollPane(voyagerPanel);  
// Минимально допустимый размер  
scrollPanel.setMinimumSize(  
    new Dimension(200, 200));  
// Предпочтительный размер  
scrollPanel.setPreferredSize(  
    new Dimension(500, 500));  
scrollPanel.setVerticalScrollBarPolicy(  
    JScrollPane.VERTICAL_SCROLLBAR_ALWAYS);  
scrollPanel.setHorizontalScrollBarPolicy(  
    JScrollPane.HORIZONTAL_SCROLLBAR_ALWAYS);
```

- ▶ Прокрутка происходит автоматически

Создание сдвоенной панели

```
JSplitPane pane = new JSplitPane(  
    JSplitPane.HORIZONTAL_SPLIT,  
    scrollPanel, infoPanel);
```

- ▶ Пользователь может регулировать размеры соседних панелей



Создание простого диалога

- ▶ Хотим при нажатии на «Выход» уточнить, надо ли выходить
- ▶ Можно создать свой JDialog
- ▶ Или использовать простой готовый диалог

Диалог подтверждения выхода

```
private void onQuit() {  
    String[] vars = { "Да", "Нет" };  
    int result =  
        JOptionPane.showOptionDialog(  
            this, "Действительно выйти?",  
            "", JOptionPane.YES_NO_OPTION,  
            JOptionPane.QUESTION_MESSAGE,  
            null, vars, "Да");  
    if (result==JOptionPane.YES_OPTION)  
        System.exit(0);  
}
```

Подтверждение выхода

- ▶ Можно добиться, чтобы тот же запрос появлялся по нажатию X или ALT-F4:

```
setDefaultCloseOperation(  
    WindowConstants.  
        DO_NOTHING_ON_CLOSE);  
addWindowListener(  
    new WindowAdapter() {  
        public void windowClosing  
            (WindowEvent ev) {  
            onQuit();  
        }  
    }) ;
```

Диалоги открытия и закрытия файла

- ▶ Нет необходимости реализовывать вручную
– существует стандартный диалог
JFileChooser

Сохранение

```
private void onSave() {
    JFileChooser fileChooser =
        new JFileChooser(currentFile);
    int result = fileChooser.showSaveDialog(this);
    if (result==JFileChooser.APPROVE_OPTION) {
        currentFile = fileChooser.getSelectedFile();
        try {
            mainPanel.saveWorldToFile(currentFile);
            this.setTitle("Коммивояжер - " +
                currentFile.getName());
        } catch (IOException ex) {
            JOptionPane.showMessageDialog(this,
                "Ошибка открытия файла");
        }
    }
}
```

Сохранение

```
public class VoyagerPanel extends JPanel {  
    // ...  
    public void saveWorldToFile(File file)  
        throws IOException {  
        ObjectOutputStream outputStream =  
            new ObjectOutputStream(  
                new FileOutputStream(file));  
        outputStream.writeObject(world);  
    }  
}
```


Аналогично, загрузка

```
public class VoyagerPanel extends JPanel {  
    // ...  
    public void openWorldFromFile(File file)  
        throws IOException,  
            ClassNotFoundException {  
        ObjectInputStream inputStream =  
            new ObjectInputStream(  
                new FileInputStream(file));  
        world = (World)inputStream.readObject();  
    }  
}
```

Файловые фильтры

```
fileChooser.addChoosableFileFilter(new FileFilter() {  
    public boolean accept(File f) {  
        if (f != null) {  
            if (f.isDirectory()) return true;  
            String name = f.getName();  
            int i = name.lastIndexOf('.');  
            if (i > 0 && i < name.length() - 1)  
                return name.substring(i+1).  
                    equalsIgnoreCase("world");  
        }  
        return false;  
    }  
    public String getDescription() {  
        return "Файлы коммивояжера (*.world)";  
    }  
});
```

Панель информации

- ▶ Должна позволять задать
 - название города
 - тип транспортного маршрута
 - стоимость маршрута
 - время в пути

Необходимые компоненты

```
cityName = new JTextField(15);
String[] kindNames = {
    WayKind.BUS.toString(),
    WayKind.TRAIN.toString(),
    WayKind.AIRCRAFT.toString()
};
wayKind = new JComboBox<>(kindNames);
wayCost = new JSpinner(
    new SpinnerNumberModel(1000, 100, 10000,
    100));
wayTime = new JSpinner(
    new SpinnerNumberModel(1, 0, 20, 1));
```

Взаимодействие панелей

- ▶ Как связаны между собой главная панель и панель информации?

Взаимодействие панелей

- ▶ Главная панель
 - текущий город
 - текущий путь
- ▶ Информационная панель
 - имя текущего города
 - тип текущего пути
 - стоимость текущего пути
 - протяженность текущего пути

Взаимодействие панелей

- ▶ Панель информации должна «знать» о том, что на главной панели выбран другой город или путь
- ▶ С другой стороны, главная панель должна «знать» о том, что на панели информации сменились какие-либо свойства города или пути
- ▶ Это значит – панели должны знать друг о друге?

Взаимодействие через интерфейсы

- ▶ В главной панели есть **два** основных события:
 - изменение текущего города
 - изменение текущего пути
- ▶ В информационной панели есть **четыре** основных события:
 - изменение имени города
 - изменение типа пути
 - изменение стоимости пути
 - изменения продолжительности пути
- ▶ Создадим интерфейсы для их обработки

Интерфейс currentListener

```
public interface CurrentListener {  
    public void currentCityChanged(City city);  
    public void currentWayChanged(Way way);  
}
```

```
public class VoyagerPanel ... {  
    CurrentListener currentListener;  
}
```

Интерфейс infoListener

```
public interface InfoListener {  
    public void cityNameChanged(String name);  
    public void wayKindChanged(WayKind kind);  
    public void wayCostChanged(int cost);  
    public void wayTimeChanged(int time);  
}  
  
public class InfoPanel ... {  
    InfoListener infoListener;  
    public void setListener(InfoListener listener) {  
        infoListener = listener;  
    }  
}
```

Обработка событий – JTextField

```
public void initListeners() {  
    cityName.addKeyListener(new KeyAdapter() {  
        public void keyReleased(KeyEvent e) {  
            if (infoListener != null)  
                infoListener.cityNameChanged(  
                    cityName.getText());  
        }  
    });  
}
```

Обработка событий – JComboBox

```
public void initListeners() {
    wayKind.addActionListener(e ->
        if (infoListener != null) {
            infoListener.wayKindChanged(getWayKind());
        }
    });
}

public WayKind getWayKind() {
    switch (wayKind.getSelectedIndex()) {
        case 0: return WayKind.BUS;
        case 1: return WayKind.TRAIN;
        default: return WayKind.AIRCRAFT;
    }
}
```

Обработка событий – JSpinner

```
public void initListeners() {  
    wayCost.addChangeListener(e ->  
        if (infoListener != null)  
            infoListener.wayCostChanged(  
                (Integer) wayCost.getValue());  
    });  
}
```

Обработка событий информационной панели

```
class MainPanel extends JPanel
    implements InfoListener {
    public void cityNameChanged(String name) {
        if (currentCity != null) {
            currentCity.setName(name);
            repaint();
        }
    }
    public void wayCostChanged(int cost) {
        if (currentWay != null)
            currentWay.setCost(cost);
    }
}
```

Выбор объектов в главной панели

```
private void onPressSelect(int x, int y) {  
    City city = world.getCityByCoord(x, y);  
    if (city != null) {  
        currentCity = city;  
        currentListener.currentCityChanged(city);  
        repaint();  
    } else {  
        // ...  
    }  
}
```

Обработка событий главной панели

```
public class InfoPanel extends JPanel
                        implements CurrentListener {
    public void currentCityChanged(City city) {
        if (city == null) {
            cityName.setEnabled(false);
        } else {
            this.setCityName(city.getName());
        }
    }
    public void setCityName(String name) {
        cityName.setEnabled(true);
        cityName.setText(name);
    }
}
```


Обработка событий главной панели

```
public class InfoPanel extends JPanel
                                implements CurrentListener {
    public void currentWayChanged(Way way) {
        if (way == null) {
            wayKind.setEnabled(false);
            wayCost.setEnabled(false);
            wayTime.setEnabled(false);
        } else {
            this.setWayKind(way.getKind());
            this.setWayCost(way.getCost());
            this.setWayTime(way.getTime());
        }
    }
}
```

Обработка событий главной панели

```
public class InfoPanel extends JPanel
    implements CurrentListener {
    public void setWayKind(WayKind kind) {
        wayKind.setEnabled(true);
        switch (kind) {
            case BUS:
                wayKind.setSelectedIndex(0);
                break;
            case TRAIN:
                wayKind.setSelectedIndex(1);
                break;
            case AIRCRAFT:
                wayKind.setSelectedIndex(2);
                break;
        }
    }
}
```

Обработка событий главной панели

```
public class InfoPanel extends JPanel
                        implements CurrentListener {
    public void setWayCost(int cost) {
        wayCost.setEnabled(true);
        wayCost.setValue(Integer.valueOf(cost));
    }
    public void setWayTime(int time) {
        wayTime.setEnabled(true);
        wayTime.setValue(Integer.valueOf(time));
    }
}
```

Undo / Redo

- ▶ См. в проекте (UndoManager)
- ▶ Идейно:
 - Одно действие = UndoableEdit
 - Сборщик действий = UndoManager

Итого

► Рассмотрено

- Форматы сохранения / загрузки
- Сложные компоненты
- Готовые диалоги
- Собственные слушатели
- Undo / Redo
- ...

► Далее

- Многопоточное программирование