МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ «ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра "ПОВТ и АС"

Компоненты JavaBeans

Методические указания к лабораторной работе

по дисциплинам "Программирование", "Объектно-ориентированное программирование"

Ростов-на-Дону 20 г.

Составитель: к.ф.-м.н., доц. Габрельян Б.В.

УДК 512.3

Компоненты JavaBeans: методические указания – Ростов н/Д: Издательский центр ДГТУ, 20 . – 8 с.

В методических указаниях рассматривается технология компонентов JavaBeans: создание компонентов, их использование, обработка сообщений, связанных с компонентами, компоненты AWT и Swing. Даны задания по выполнению лабораторной работы. Методические указания предназначены для студентов направлений 090304 "Программная инженерия", 020303 "Математическое обеспечение и администрирование информационных систем".

Ответственный редактор:

Издательский центр ДГТУ, 20

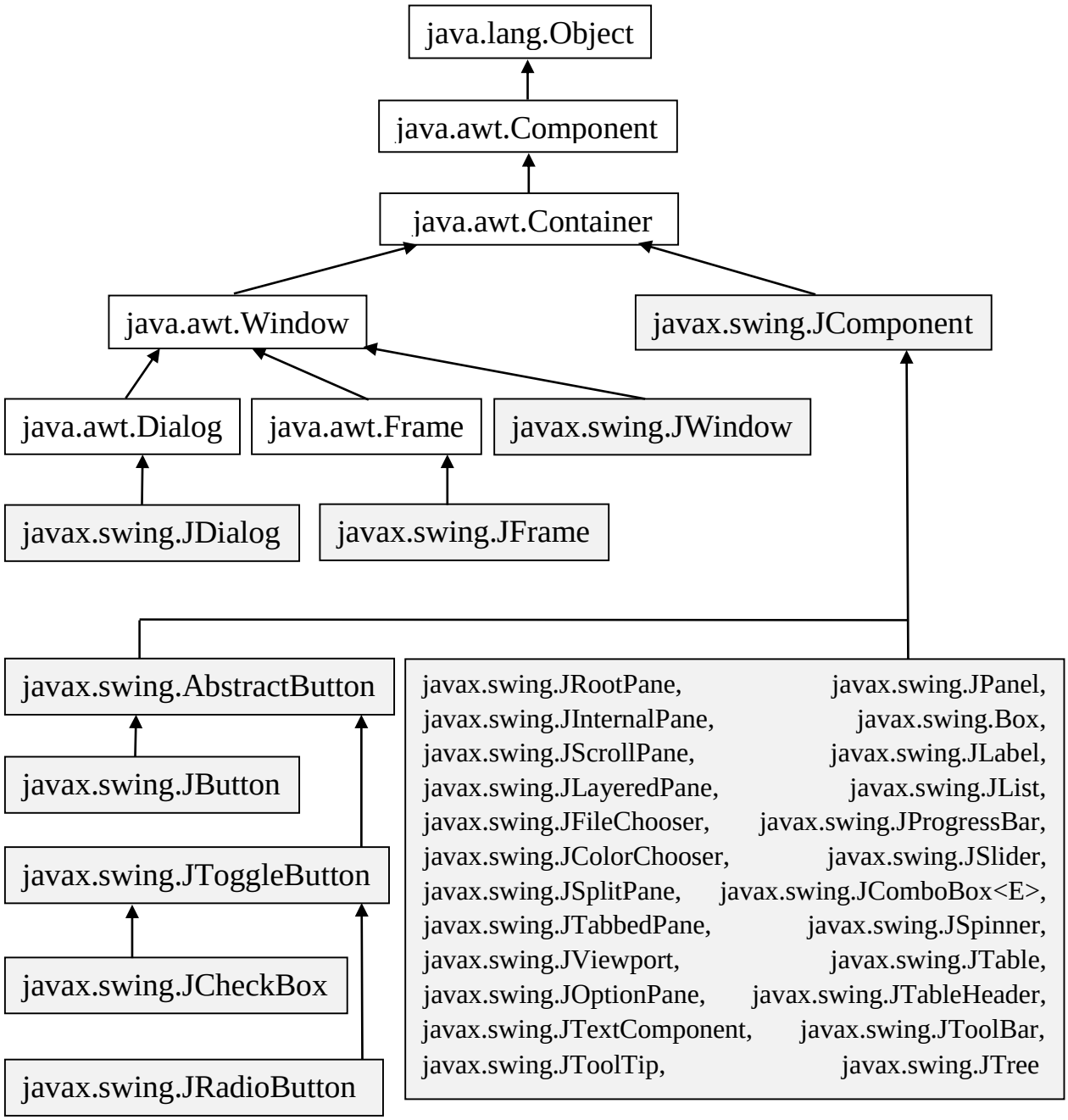
JavaBeans - это модель компонентов, создаваемых на языке Java. Соответственно, компоненты, создаваемые по этой технологии, часто называют "зернами" (Beans). Компоненты представляют собой законченные программные единицы, используемые не на уровне исходного текста, но в откомпилированном, готовом к выполнению виде. Т.к. один и тот же компонент, собственный или приобретенный, может быть "строительным блоком" различных приложений, разработчик получает возможность повторно использовать прежде написанный код. Компоненты строятся на основе классов, для этого к абстракциям объектной модели добавляются новые - интроспекция, свойства, события. Для поддержки новых абстракций могут использоваться дополнительные (по сравнению с обычными классами) языковые конструкции (как, например, в C#), либо, как в Java, некоторые соглашения (протоколы) по организации класса, представляющего компоненты, которые не требуют расширения синтаксиса классов.

Для использования компонентов - откомпилированных модулей, необходимо, во-первых, найти их в системе, во-вторых, получить информацию об их возможностях. В разных моделях компонентов для этого используются различные схемы. Например, компоненты, созданные по технологии COM фирмы Microsoft, должны быть зарегистрированы в системе (доступ к ним осуществляется по уникальному идентификатору) и должны поддерживать помимо собственных, специфичных интерфейсов, некоторые обязательные одинаковые базовые интерфейсы. В технологии JavaBeans используется другой подход, с одной стороны, компонент, как и обычный класс, хранится в файле, имя которого совпадает с именем класса компонента и поиск компонента сводится к поиску файла, с другой стороны, для анализа возможностей компонента Java поддерживает так называемый механизм интроспекции. Этот механизм позволяет проводить либо автоматический анализ компонента (определять его свойства, открытые методы), либо использовать предоставленную разработчиком компонента в виде дополнительного класса (мета класса) подробную информацию о компоненте. Мета класс должен иметь специальное имя и должен реализовывать стандартный интерфейс java.beans.BeanInfo (см. ниже). Помимо технологии JavaBeans в Java реализована еще одна собственная модель компонентов - Enterprise JavaBeans. Хотя последнюю модель можно считать расширением JavaBeans, две технологии различаются областями своего применения. JavaBeans ориентирована на клиентские приложения, чаще всего обладающие графическим пользовательским интерфейсом. Например, классы AWT и Swing, представляющие элементы управления пользовательским интерфейсом (списки, кнопки и т.д.), реализованы по технологии JavaBeans. Напротив, Enterprise JavaBeans - это технология компонентов, содержащих деловую

логику приложения и выполняющихся на сервере, тем самым не предполагающих графического представления. В дальнейшем рассматривается только технология JavaBeans.

1. Стандартные компоненты Swing

Все стандартные компоненты пакетов AWT и Swing построены по технологии JavaBeans. Следующая схема представляет иерархию некоторых компонентов Swing.



Базовый функционал компонентов AWT и Swing объявлен в абстрактном классе java.awt.Component. Это, во-первых, поддержка определения и задания (в случае, если это разрешено менеджером компоновки) размеров и положения компонента в окне родительского компонента для подчиненных окон и относительно верхнего левого угла экрана для окон верхнего уровня (например, JFrame). Её обеспечивают методы

Dimension getSize()
void setSize(Dimension d)
void setSize(int width, int height) - возвращает/задает размер компонента
Dimension getMaximumSize()
void setMaximumSize(Dimension d) - возвращает/задает максимально возможный размер компонента
Dimension getMinimumSize()
void setMinimumSize(Dimension d) - возвращает/задает минимально возможный размер компонента
Dimension getPreferredSize()
void setPreferredSize(Dimension d) - возвращает/задает предпочитаемый размер компонента
int getHeight() - возвращает высоту компонента
int getWidth() - возвращает ширину компонента
Rectangle getBounds() - возвращает позицию и размеры компонента
void setLocation(int x, int y) - задает положение верхнего левого угла компонента
void setBounds(int x, y, width, height)
void setBounds(Rectangle r)

Класс Dimension содержит два поля int width, height; представляющих соответственно ширину и высоту компонента в пикселях. В классе Rectangle объявлены четыре поля int x, y, width, height; - координаты левого верхнего угла компонента, его ширина и высота.

Во-вторых, Component задает основные методы для рисования компонента:

boolean isVisible() - true, если компонент отображается в окне
void setVisible(boolean flag) - отображает/скрывает компонент
void paint(Graphics g) - вызывается всякий раз, когда компонент должен быть перерисован
void repaint() - принудительно вызывает перерисовку компонента

Graphics getGraphics() - возвращает графический контекст, позволяющий рисовать в окне компонента вне метода paint, результаты рисования исчезнут после вызова paint (если paint не выполнит такое же точно рисование)
Color getBackground()
void setBackgound(Color c) - возвращает/задает цвет фона компонента
Color getForeground()
void setForeground(Color c) - возвращает/задает цвет надписей внутри компонента
void setFont(Font f) - шрифт надписей

В-третьих, Component определяет методы, позволяющие добавлять код, который должен вызываться тогда, когда происходит определенное событие, связанное с компонентом. В терминологии Java этот код называется слушателем события (Listener). Например, если пользователь щелкает мышкой по области экрана, занятой компонентом возникает событие от мыши (MouseEvent) причем источником события считается компонент, а обработчиком события может быть объект любого класса, реализующего предопределенный интерфейс слушателей событий от мыши - MouseListener. Если какой-то код (внешний по отношению к компоненту и реализующий соответствующий интерфейс *XXX*Listener, где *XXX* - имя события) должен вызываться для организации отклика на это событие, то его нужно зарегистрировать в компоненте (источнике события). Для этого компонент - источник события должен предоставить метод void add*XXX*Listener(*XXX*Listener listener). В классе Component определены методы

void addComponentListener(ComponentListener cl)
void addFocusListener(FocusListener fl)
void addKeyListener(KeyListener kl)
void addMouseListener(MouseListener ml)
void addMouseMotionListener(MouseMotionListener mml)
void addMouseWheelListener(MouseWheelListener mwl)
void propertyChangeListener(PropertyChangeListener pcl)

boolean isEnabled()
void setEnabled(boolean flag) - позволяет получить/установить режим реакции на события, если false, компонент не будет реагировать на события (и будет отображаться более тусклым цветом)

Некоторые стандартные компоненты Swing:

JLabel - представляет компонент, который содержит текстовую метку (String), изображение (Icon) или то и другое вместе. Не обрабатывает сообщения от мыши и клавиатуры. И текст, и изображение можно менять в процессе использования компонента.
Конструкторы
JLabel(String text)
JLabel(Icon image)
JLabel(String text, Icon image, int horizontalAlignment)

Основные методы
String getText()
void setText(String text)
Icon getIcon()
void setIcon(Icon image)
Текст для метки можно задавать, используя html-разметку. Например, метка содержащая двухстрочный текст, в первой строке "X" курсивом синим цветом, во второй "min", курсивом зеленым цветом
JLabel xmin = new JLabel("<html><font color=blue>X
<font color=green>min");

JButton - компонент, представляющий кнопку на которую можно нажимать. После того, как на кнопку нажали, она возвращается в исходное состояние (Swing предлагает и другой компонент - JToggleButton, кнопку способную находиться в одном из двух состояний - нажата, отпущена, после того как на такую кнопку нажали она остается в нажатом состоянии, если нажать еще раз она переходит в состояние "не нажата"). Кнопка может содержать текстовую метку (String), изображение (Icon) или то и другое вместе.
Конструкторы
JButton(String text)
JButton(Icon image)
JButton(String text, Icon image)

Основные методы
String getText()
void setText(String text)
Icon getIcon()
void setIcon(Icon image)
void addActionListener(ActionListener al) - регистрация кода (слушателя) вызываемого при нажатии на кнопку
В родительском классе AbstractButton объявлен также метод void

setMnemonic(int m) для быстрой активации кнопки с помощью клавиатуры. Активация (нажатие на кнопку) происходит, если нажата клавиша, заданная в методе setMnemonic вместе с управляющей клавишей (зависит от операционной системы, обычно Alt). При этом компонент содержащий кнопку должен быть активен (в фокусе). Клавиша активации задается по ее коду. Удобно использовать предопределенные константы из класса KeyEvent. Например,

```
JButton colorBtn = new JButton("Color");
colorBtn.setMnemonic(KeyEvent.VK_C);
```

Клавишей активации будет клавиша с буквой C, причем регистр символа не учитывается так как код относится к клавише, а не к символу. Надпись на кнопке будет иметь вид "Color", то есть символ C будет подчеркнут. Теперь, если нажать Alt+C произойдет событие "нажатие на кнопку colorBtn".

JTextField - компонент для редактирования однострочного текста. Можно задавать цвет текста (void setForeground(Color c)) и цвет фона (void setBackground(Color c)), шрифт (void setFont(Font f)), но одни и те же цвет и шрифт будут использованы для всего текста. В окне JTextField может быть видна лишь часть текста, тогда щелкнув по тексту мышкой можно перемещаться по тексту с помощью клавиш со стрелками на клавиатуре.

Конструкторы

JTextField(String text)

JTextField (int columns) - columns, это число символов текста, которые будут видны в окне компонента, на самом деле видимая область компонента будет достаточно большой, чтобы отобразить columns символов некоторой "средней" для используемого шрифта ширины

JTextField (String text, int columns)

Основные методы

String getText()

void setText(String text)

void select(int start, int end) - позволяет из программы выделить часть текста, а именно символы от позиции start до позиции end

void selectAll() - выделяет весь текст

void getSelectedText() - возвращает выделенный текст

void setEditable(boolean flag) - позволяет/запрещает редактировать текст компонента

void addActionListener(ActionListener al) - регистрация кода (слушателя) вызываемого при завершении редактирования текста (нажатии на кнопку Enter)

JTextArea - область для отображения многострочного текста в остальном подобен JTextField, но помимо числа символов в строке (columns) определено и число строк в области (rows), rows и columns влияют только на предпочтительный размер компонента, текст рассматривается как непрерывная последовательность символов.
Конструкторы
Конструкторы
JTextArea(String text)
JTextArea (int rows, int columns)
JTextArea (String text, int rows, int columns)

Основные методы
String getText()
void setText(String text)
void select(int start, int end) - позволяет из программы выделить часть текста, а именно символы от позиции start до позиции end
void selectAll() - выделяет весь текст
void getSelectedText() - возвращает выделенный текст
void append(String text) - добавляет указанный текст в конец документа
void insert(String text, int position) - вставляет текст в указанную позицию документа
void setEditable(boolean flag) - позволяет/запрещает редактировать текст компонента
Для работы с текстом существуют и другие компоненты, в частности JEditorPane и его наследник JTextPane. Они в принципе позволяют работать с текстом с произвольной разметкой. Предусмотрены форматы HTML и RTF.

JList<E> - обобщенный класс, отображающий список объектов с возможностью выбора одного или нескольких из них.
Конструкторы
JList()
JList(E[] data)
Например,

```
String[] str = {"veni", "vidi", "vici"};
JList<String> list = new JList<>(str);
```

Основные методы
int getSelectedIndex() - возвращает индекс выбранного элемента списка (индексирование от нуля)
void setSelectedIndex(int index) - выделяет элемент списка с указанным

индексом
int[] getSelectedIndexes() - возвращает массив индексов выбранных элементов списка
void setSelectedIndexes(int[] index) - выделяет элементы списка с указанными индексами
E getSelectedValue() - возвращает выбранный объект
List<E> getSelectedValuesList() - возвращает список выбранных объектов
int getSelectionMode() - возвращает режим выделения элементов. Возможные значения определяются константами ListSelectionModel.SINGLE_SELECTION - может быть выбран только один элемент, ListSelectionModel.SINGLE_INTERVAL_SELECTION - могут быть выбраны несколько подряд расположенных элементов, ListSelectionModel.MULTIPLE_INTERVAL_SELECTION - могут быть выбраны произвольные элементы
void setSelectionMode(int mode) - устанавливает режим выделения элементов
void addListSelectionListener(ListSelectionListener lsl) - регистрация кода (слушателя) вызываемого при выборе элемента списка

JComboBox<E> - обобщенный компонент, представляющий собой комбинацию кнопки или текстового поля ввода и выпадающего списка. Позволяет выбирать элемент списка, а если сделать JComboBox редактируемым, то позволяет вводить данные в редактируемое поле ввода.
Конструкторы
JComboBox()
JComboBox(E[] items)

Основные методы
int getSelectedIndex() - возвращает индекс выбранного элемента списка (индексирование от нуля)
void setSelectedIndex(int index) - выделяет элемент списка с указанным индексом
Object getSelectedItem() - возвращает выбранный объект
void setSelectedItem(Object o) - выделяет указанный элемент
void setEditable(boolean flag)
void addItem(E item)
void insertItemAt(E item, int index)
void removeItem(Object item)
void removeItemAt(int index)
void removeAllItems()
void addActionListener(ActionListener al) - регистрация кода (слушателя)

вызываемого при выполнении действия (выбора элемента, нажатии на кнопку Enter, двойном щелчке мышью для редактируемого компонента)
void addItemListener(ItemListener al) - регистрация кода (слушателя) вызываемого при изменении состояния элемента списка (выбран, не выбран)

JCheckBox - флажок с независимой фиксацией, может быть отмечен (выбран) или нет. Если используется группа таких флажков, каждый из них может быть выбран или нет независимо от состояния других флажков группы.
Конструкторы
JCheckBox(String text)
JCheckBox(String text, boolean selected)
JCheckBox(Icon icon)
JCheckBox(Icon icon, boolean selected)
JCheckBox(String text, Icon icon, boolean selected)

Основные методы
boolean isSelected() - флажок выбран или нет
void setSelected(boolean flag) - сделать флажок выбранным или не выбранным
void addItemListener(ItemListener al) - регистрация кода (слушателя) вызываемого при изменении состояния элемента списка (выбран, не выбран)

JRadioButton - флажок с зависимой фиксацией, может быть отмечен (выбран) или нет. Если используется группа таких флажков, то выбранным в ней может быть только один из флажков. Группа - это объект класса ButtonGroup. В ButtonGroup есть метод void add(AbstractButton ab).
Конструкторы
JRadioButton(String text)
JRadioButton (String text, boolean selected)
JRadioButton (Icon icon)
JRadioButton (Icon icon, boolean selected)
JRadioButton (String text, Icon icon, boolean selected)

Основные методы
boolean isSelected() - флажок выбран или нет
void setSelected(boolean flag) - сделать флажок выбранным или не выбранным
void addActionListener(ActionListener al) - регистрация кода (слушателя) вызываемого при выполнении действия

Swing поддерживает также другие, относительно простые (JProgressBar, JSlider, JSpiner) и более сложные (JTree, JTable) компоненты.

JTable - компонент отображающий и, в общем случае, позволяющий редактировать данные организованные как двумерная таблица. Обычно таблица помещается в панель JScrollPane, предоставляющую при необходимости горизонтальную и/или вертикальную полосу прокрутки. Кроме того, помещение таблицы в эту панель позволяет автоматически отображаться заголовку таблицы. Заголовок таблицы обычно содержит названия ее столбцов. Ячейки разных столбцов таблицы могут содержать объекты разных классов, но для прорисовки таких нестандартных объектов должны предоставляться специальные прорисовщики (renderers), а для их редактирования - редакторы (editors). В простейшем случае таблицы, содержащей строки, можно использовать данные в виде двумерного массива строк и предоставить массив строк с именами столбцов таблицы, например

```
String[] columnNames = {"A", "B", "C"};
String[][] data = {
        {"a1", "b1", "c1"},
        {"a2", "b2", "c2"},
        {"a3", "b3", "c3"},
        {"a4", "b4", "c4"},
        {"a5", "b5", "c5"},
        {"a6", "b6", "c6"}
};
JTable table = new JTable(data, columnNames);
JScrollPane scroll = new JScrollPane(table);
```

Основные методы
int getSelectedRow()
Object getValueAt(int row, int col)
void setValueAt(Object value, int row, int col)

1.1. Контейнеры и менеджеры компоновки

Базовые возможности контейнеров, компонентов способных содержать в себе другие компоненты представляет абстрактный класс java.awt.Container, который сам является компонентом (наследником класса java.awt.Component). Все легковесные компоненты Swing построены на основе абстрактного класса JComponent, который, в свою очередь является наследником Container. Есть контейнеры верхнего уровня (JFrame, JDialog) и контейнеры, которые размещаются внутри других контейнеров (JPanel, JTabbedPane, JSplitPane,

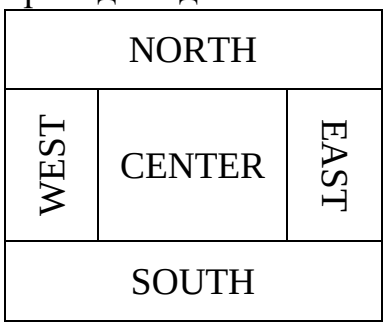
JScrollPane, …). Контейнер обеспечивает, во-первых, добавление/удаление компонентов и, во-вторых, возможность устанавливать менеджер компоновки, который будет управлять размещением компонентов в окне контейнера.
Component add(Component c)
Component add(String name, Component c) - с добавляемым компонентом можно связать имя
void add(Component c, Object restrictions) - ограничения влияют на расположение компонента в окне контейнера
void remove(Component c)
void removeAll()
LayoutManager getLayout() - получить объект класса (менеджера компоновки), который управляет размещением компонентов в окне контейнера
void setLayout(LayoutManager lm) - задать объект класса (менеджера компоновки), который будет управлять размещением компонентов в окне контейнера
Если в контейнере размещаются какие-то компоненты, необходимо определить, в каких позициях окна контейнера они должны размещаться и какие у них должны быть размеры. Это можно делать вручную с помощью методов, объявленных в классе Component: setLocation, setSize, setBounds. Для этого нужно для контейнера вызвать метод setLayout и передать ему пустую ссылку. Например,

```
JFrame frame = new JFrame("Absolute Layout");
frame.setLayout(null);
frame.setSize(500,500);
JButton button = new JButton("Button");
button.setBounds(50,50,120,80);
```

Но теперь, при каждом изменении размеров окна контейнера для красивого расположения всех включенных в него компонентов нужно переопределять параметры их размещения - рассчитывать заново их позиции и быть может размеры. Для того, чтобы автоматизировать этот процесс AWT и Swing предлагают специальные классы - классы менеджеров размещения (компоновки). Разные менеджеры используют разные стратегии размещения компонентов, поэтому их комбинация позволяет реализовать как простые, так и достаточно сложные варианты размещений.

С контейнером верхнего уровня JFrame по умолчанию связан менеджер компоновки, представленный стандартным классом java.awt.BorderLayout. Этот менеджер компоновки разбивает (логически) всю область окна контейнера на пять областей: северную (NORTH), южную (SOUTH), западную (WEST), восточную (EAST) и центральную (CENTER). При добавлении компонента в контейнер нужно явно или неявно (по умолчанию добавление

производится в центральную часть) указать в какую из этих областей проводить добавление.



Для этого используется метод add контейнера с ограничением. Ограничение представляется предопределенной в классе BorderLayout статической константой: NORTH, north, SOUTH, south, WEST, ewst, EAST, east, CENTER, center.

Например,

```
JFrame frame = new JFrame("BorderLayout");
frame.setSize(500, 500);
JPanel northPanel = new JPanel();
northPanel.setPreferredSize(new Dimension(10, 100));
JPanel westPanel = new JPanel();
westPanel.setPreferredSize(new Dimension(100, 10));
JPanel centerPanel = new JPanel();
centerPanel.setPreferredSize(new Dimension(10, 10));
frame.add(northPanel, BorderLayout.NORTH);
frame.add(westPanel, BorderLayout.WEST);
frame.add(centerPanel, BorderLayout.CENTER);
// frame.add(centerPanel); // то же самое, в центр по умолчанию
```

Как видно из примера, не обязательно заполнять все пять областей, можно использовать любое их количество (от одной до пяти). Кроме того, BorderLayout учитывает для компонентов, помещаемых на запад или на восток только горизонтальный предпочитаемый размер (ширину), а для компонентов, помещаемых на север или на юг только вертикальный предпочитаемый размер (высоту). При этом компоненты растягиваются по высоте (запад, восток) или ширине (север, юг) так чтобы заполнить окно контейнера. Компонент, помещаемый в центр растягивается как по ширине, так и по высоте. Такое поведение сохраняется при изменениях размеров окна контейнера. Если в одну и ту же область добавить несколько компонентов, последний добавленный закроет собой все компоненты, добавленные в нее ранее.

Для подчиненного контейнера JPanel по умолчанию задается другой менеджер компоновки - java.awt.FlowLayout. Этот менеджер учитывает предпочитаемые размеры добавляемых компонентов и размещает их в окне контейнера последовательно, слева-направо и сверху-вниз. Если очередной добавляемый компонент не помещается в текущей строке, он переносится в следующую строку.

Можно отказаться от умолчания и назначить контейнеру менеджер компоновки явно

```
JFrame frame = new JFrame("Layout");
```

```
frame.setLayout(new FlowLayout());
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());
```

Существуют другие менеджеры компоновки: java.awt.CardLayout - добавленные компоненты как в стопке игральных карт размещаются друг над другом и доступен только верхний компонент, но, менеджер компоновки позволяет делать видимым следующий компонент и таким образом можно задать дисциплину работы с компонентами похожую на работу с элементами стека; java.awt.GridLayout - поверхность окна контейнера логически представляется как сетка из заданного числа ячеек одинакового размера и добавляемые компоненты последовательно заполняют эти ячейки (по умолчанию слева-направо и сверху-вниз).
Например,


```
class InfoPanel extends JPanel {
  @Override
  public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.setColor(Color.WHITE);
        g.drawString("w="+getWidth()+
              " h="+getHeight());
  }
}
…
JFrame frame = new JFrame("GridLayout");
frame.setLayout(new GridLayout(3, 2));
frame.setSize(500, 500);
frame.add(new JButton("1"));
…
frame.add(new JButton("5"));
JPanel panel = new InfoPanel();
panel.setBackground(Color.RED);
frame.add(panel);
…
```

Менеджер компоновки GridBagLayout позволяет реализовывать более сложное расположение компонентов. Фактически он дает возможность в сетке подобной GridLayout объединять рядом расположенные ячейки и полученную область отдавать добавляемому компоненту. Кроме того, можно использовать javax.swing.Box для вертикального или горизонтального последовательного размещения компонентов. Есть сложный менеджер компоновки

javax.swing.SpringLayout, который предназначен для средств визуального создания программ. В нем можно использовать "распорки", "пружины", размещаемые между компонентами для создания сложных компоновок.
Считается, что можно, не используя сложные менеджеры, но комбинируя вложенные панели с разными простыми менеджерами размещения получать в целом почти любые сложные компоновки.

1.2. События, связанные со стандартными компонентами

Так как все компоненты GUI представлены классами, производными от Component, все они в принципе могут быть источниками событий от мыши и клавиатуры. Помимо этого, компоненты могут генерировать и другие события. Классы представляющие такие предопределенные события размещены в пакетах java.awt.event, javax.swing.event: ActionEvent, TextEvent, CaretEvent, FocusEvent, ListSelectionEvent, ItemEvent. Блоки прослушивания для этих событий должны реализовывать интерфейсы: ActionListener, TextListener и т.д.

С компонентами JButton, JTextField, JComboBox, JRadioButton связано событие Action. Оно возникает, если над компонентом выполнено действие - нажата кнопка (с помощью мыши или при нажатии на клавишу пробел на клавиатуре), нажата клавиша Enter при вводе текста (или двойной щелчок мышью) и т.д. Блок прослушивания для этого действия - **ActionListener** определяет один абстрактный метод void actionPerformed(ActionEvent ae). Например, кнопка, при нажатии на которую ее метка меняется с Yes на No и наоборот

```
import java.awt.*; // JButton
import java.awt.event.*; // ActionEvent, ActionListener

JButton ynBtn = new JButton("Yes");
ynBtn.addActionListener(new ActionListener() {
	@Override
	public void actionPerformed(ActionEvent ae) {
		if(ynBtn.getText() == "Yes") ynBtn.setText("No");
		else ynBtn.setText("Yes");
	}
});
```

С компонентами JComboBox, JCheckBox связано событие Item, возникающее при выборе элемента. Блок прослушивания для этого действия - **ItemListener** определяет один абстрактный метод void itemStateChanged(ItemEvent ae). Например, выбор элемента в JComboBox

```
…
String[] items = {"A", "B", "C"};
JComboBox combo = new JComboBox(items);
combo.addItemListener(new ItemListener() {
	@Override
	public void itemStateChanged(ItemEvent ie) {
		int index = combo.getSelectedIndex();
		System.out.println(combo.getItemAt(index));
	}
});
…
```

При выборе, например, пункта "B" отобразится

```
B
B
```

Так происходит потому, что с точки зрения системы вначале неактивный пункт стал невыбранным, а затем выбранным. Чтобы различить эти состояния нужно вызвать метод int getStateChange() класса ItemEvent. В этом классе объявлены также статические константы SELECTED и DESELECTED.

```
…
		if(e.getStateChange() == ItemEvent.SELECTED)
			System.out.println(combo.getItemAt(index));
…
```

Так как JComboBox генерирует также событие Action можно добавить к нему блок прослушивания этого события.

```
…
combo.addActionListener(new ActionListener() {
	@Override
	public void actionPerformed(ActionEvent ae) {
		int index = combo.getSelectedIndex();
		System.out.println(combo.getItemAt(index));
	}
});
…
```

При выборе элемента JComboBox произойдет и событие Action, но оно произойдет только один раз. Кроме того, если снова выбрать уже выбранный элемент, то событие Item не произойдет, а событие Action произойдет.

1.3. Подключаемые вид и поведение

Программы на Java являются кросс-платформенными. Вид стандартных элементов управления в разных операционных системах и даже в разных

библиотеках компонентов разный. Swing позволяет изменять вид и поведение (Look And Feel - LAF) сразу всех (или только некоторых) легковесных элементов графического пользовательского интерфейса (GUI) в конкретной Java-программе. Можно создавать свои подключаемые вид и поведение (Pluggable Look And Feel - PLAF). Swing содержит несколько предопределенных LAF. Можно получить информацию обо всех зарегистрированных в системе LAF с помощью статического метода getInstalledLookAndFeels() класса UIManager. Метод возвращает массив ссылок на объекты класса UIManager.LookAndFeelInfo для каждого установленного LAF. Основные методы класса UIManager.LookAndFeelInfo это методы String getClassName() и String getName(). Первый возвращает имя класса LAF, которое нужно использовать для переключения на этот вид, а второй - короткое имя для ссылок на соответствующий LAF. Например, для LAF по умолчанию getClassName() вернет строку javax.swing.plaf.metal.MetalLookAndFeel, а getName() строку Metal.

```
UIManager.LookAndFeelInfo[] lfi = UIManager.getInstalledLookAndFeels();
System.out.println("class name: "+lfi[0].getClassName()+"\tname: "+lfi[0].getName());
```

Установить нужный LAF позволяет статический метод UIManager.setLookAndFeel(String className). При вызове метода могут генерироваться различные исключения, поэтому его вызов должен выполняться в блоке try.

```
try {
    UIManager.setLookAndFeel("javax.swing.plaf.metal.MetalLookAndFeel");
}
catch(Exception e) {
    …
}
```

Если компоненты пользовательского интерфейса (User Interface – UI) еще не созданы на момент вызова setLookAndFeel, то самого вызова достаточно для смены LAF. В противном случае необходимо вызвать метод updateComponentTreeUI(Component c) класса SwingUtilities. Этому методу можно передать ссылку на главное окно программы, тогда изменится вид всех компонентов, расположенных на главном окне программы - все дерево размещенных в окне компонентов UI.

```
JFrame mainFrame = new JFrame("Main");
try {
   UIManager.setLookAndFeel("javax.swing.plaf.nimbus.NimbusLookAndFeel");
}
```

```
catch(Exception e) {
        …
}
SwingUtilities.updateComponentTreeUI(mainFrame);
```

Можно передать ссылку на отдельный компонент, тогда LAF изменится для этого компонента и всех компонентов, расположенных в нем.

2. Создание и использование собственных простых компонентов JavaBeans

Чтобы создать компоненты, поддерживающие автоматическую интроспекцию, необходимо определить класс:

1) реализующий интерфейс Serializable,
2) имеющий конструктор по умолчанию,
3) поддерживающий "правила имен" для свойств
4) и, если обрабатываются сообщения, модель делегирования событий.

"Правила имен" позволяют определять свойства. Если в классе определены открытые методы get*Имя* и set*Имя*, то механизм автоматической интроспекции определяет наличие у класса свойства *имя* (с маленькой буквы!). Среда быстрой разработки программ (например, NetBeans), используя интроспекцию, находит все свойства компонента и отображает их имена в окне редактора свойств. В этом редакторе можно задать новое значение для свойства. При этом NetBeans (или другая IDE) добавит в текст программы вызов метода set*Имя*. А для того, чтобы отобразить значение свойства среда обращается к методу get*Имя*. Если предусмотрен лишь один открытый метод get*Имя*, то свойство доступно только для чтения, а если лишь метод set*Имя* - только для изменения (записи). Если тип свойства boolean, вместо метода с именем get*Имя* обычно используют метод с именем is*Имя*. Свойства могут быть и индексированными.

После компиляции класс компонента должен быть упакован в архив (jar-файл) вместе с текстовым файлом описания (манифестом), в котором для класса компонента должны быть указаны стандартные учетные записи. Например, если имя класса компонента MyBeanClass, в манифесте должны быть строки (некоторые среды разработки, например, NeIBeans 4.x, требуют, чтобы перед Name была пустая строка):

```
Name: MyBeanClass.class
Java-Bean: True
```

Эти строчки нужно поместить в текстовый файл, например, manif, затем вызвать утилиту jar:

```
jar -cmf manif mbean.jar MyBeanClass.lass
```

После чего будет создан новый java-архив mbean.jar, содержащий файл MyBeanClass.class и подкаталог META-INF с файлом манифеста MANIFEST.MF в который будет записано содержимое файла manif.

Чтобы компонент стал доступен из интегрированной среды разработки (IDE), необходимо занести его на палитру инструментов. В NetBeans для этого нужно выбрать пункт меню Tools/Install new Java Bean … и затем указать jar-файл с компонентом. Для новых компонентов предназначена панель Beans. Теперь можно добавить его на форму и настраивать так же, как и любой другой известный NetBeans компонент.

NetBeans содержит шаблоны для создания компонентов (в том числе и визуального). При этом сами компоненты автоматически не упаковываются в jar-файлы (чтобы упаковать, необходимо создать отдельный "узел" для архива). Такие компоненты можно добавить в палитру, щелкнув правой кнопкой в области окна редактирования и выбрав Tools/Add to Component Palette...

3. Обработка событий, связанных с компонентом

Чаще всего визуальный компонент управляется внешними событиями. Тогда разработчик компонента должен использовать стандартную схему делегирования сообщений. Для этого компонент должен определять и регистрировать в источниках событий собственные блоки прослушивания. Например, если компонент реагирует на щелчок мыши:

```
…
public MyBean extends JComponent {
        …
        public MyBean() {
                …
                addMouseListener( new MouseAdapter(){
                        public void mouseClicked(MouseEvent me) {
                                if(!enabled) return; // если в неактивном состоянии
                                …
                        }
                        …
                });
                …
        }
        …
}
…
```

С другой стороны, компонент может выступать не только в роли потребителя,

но и в качестве источника событий, чаще всего, событие, связанное с компонентом, это изменение значения какого-нибудь из его свойств. Изменение значения свойства - это предопределенное в Java событие PropertyChangeEvent. Соответствующий интерфейс блока прослушивания - это PropertyChangeListener (пакет java.beans). В интерфейсе объявлен один метод propertyChange(PropertyChangeEvent pce). Обычно событие генерируют в методе set соответствующего компонента. Для генерации используется метод firePropertyChange(String propertyName, Type oldValue, Type newValue), где Type - это либо Object, либо примитивный тип. В обработчике события (методе propertyChange()) можно получить переданные при генерации (методом firePropertyChange()) параметры с помощью методов класса PeopertyChangeEvent: Object getNewValue(), Object gelOldValue() и String getPropertyName(). Например,

```
// Компонент
import javax.swing.*;
import java.bean.*;
…
class SuperBean extends JComponent (
        …
        // свойство
        private int value;
        …
        public int gelValue() { return value; }
        public void setValue(int v) {
                int old = value;
                value = v;
                // генерация события
                firePropertyChange(
                        "SuperValue", // имя не обязано совпадать с именем свойства
                        old,
                        value);
                …
        }
}
…
// Класс, использующий компонент SuperBean

class BeanOvner extends JFrame implements PropertyChangeListener {
…
```

```
        private SuperBean sbean = new SuperBean();
…
        public BeanOvner (){
                …
// создать и присоединить блок прослушивания события PropenyChangeEvent
                sbean.addPropertyChangeListener( this );
                …
        }
        …
        public void propertyChange(PropertyChangeEvent pce) {
                String pName = pce.getPropertyName();
                if(pName.compareTo("SuperValue") == 0) {
                        int oldV = ((Integer)getOldValue()).intValue():
                        int newV = ((Integer)getNewValue()).intValue();
                        …
                }
                …
        }
…

}
```

Свойство, изменение значения которого приводит к генерации события PropertyChangeEvent, называется связанным (bound).

4. Полный контроль над компонентами - интерфейс JavaBeans

Если программа строится из набора компонентов, то ее создание с помощью средств визуального программирования или быстрого создания программ выглядит как добавление известных этим средствам компонентов на форму и настройка их взаимодействия. Известные среде разработки компоненты размещаются на какой-либо панели инструментов (например, панель AWT, панель Swing, панель Beans) и представлены на ней какими-нибудь, в идеале уникальными, значками - иконками. Если значки уникальные и хорошо отражают сущность компонента, использовать их легко. Если же несколько компонентов представлены одинаковым значком, различить их можно только по имени класса, представляющего компонент. Значок для компонента должен предоставить разработчик последнего. Если же такого значка нет, как у компонентов из задания1 и задания2, среда разработки предоставит для них один и тот же зарезервированный значок. Кроме того, в редакторе свойств отображаются все свойства компонента, как собственные,

так и унаследованные. Чаще всего, большинство или даже все унаследованные свойства нежелательно настраивать в редакторе свойств напрямую, т.е. необходимо обеспечить отображение только некоторых, но не всех свойств компонента. То же самое может относиться и к методам и событиям, связанным с компонентом. Для такого "тонкого" управления информацией, отображаемой и настраиваемой в среде быстрой разработки программ, необходимо использовать класс, описывающий возможности компонента и не имеющий самостоятельного смысла (мета класс). Для компонента с именем Имя мета классом будет класс, реализующий специальный интерфейс BeanInfo и имеющий имя ИмяBeanInfo. В интерфейсе BeanInfo объявлены 4 константы и 8 методов (см. документацию по Java). Константы: ICON_COLOR_16X16, ICON_COLOR_32X32, ICON_NO_16X16, ICON_MONO_32X32 определяют тип иконки, представляющей компонент на панели инструментов среды разработки. Иконки - это графические файлы в формате GIF, цветные или монохромные, размером 16x16 или 32x32 пикселя Т.е. с компонентом можно связать до четырех изображений. Какое из них использовать, определяет среда разработки, но фирма Sun рекомендовала представлять, по крайней мере, цветную иконку размером 16x16 пикселей. Иконки должны быть размещены в архиве компонента.
Ниже представлены некоторые методы интерфейса Beanlnfo:

Image getIcon(int iconType)

PropertyDescripto[] getPropertyDescriptors()

MethodDescriptor[] getMethodDescriptors()

EventSetDescriplo[] getEventSetDescriptors()

Чтобы получить иконку, представляющую компонент, среда разработки вызывает метод getIcon() мета класса. Соответственно, чтобы получить список объектов, содержащих описания свойств, отображаемых в редакторе свойств, вызывается метод мета класса getPropertyDescriptors(). Для методов getMethodDescriptors(), а для свойств - getEventSetDescriptors(). Описания классов ProptyDescriptor, MethodDescriptor, EventSetDescriptor см. в документации к Java.

В пакете java.beans объявлен также класс SimpleBeanInfo выполняющий роль адаптера для классов, реализующих интерфейс BeanInfo. В этом классе, помимо пустых реализаций методов интерфейса определен также метод Image loadImage(String imgName) для загрузки файла с иконкой для компонента.

Например,

```
…
class SuperBeanBeanInfo extends SimpleBean {
        private Image icon;
        private static PropertyDescriptor[] propDescr = {
                new PropenyDescriptor("value", SuperBean.class) };

        public SuperBeanBeanlnfo() {
                if(icon == null) loadImage("super.gif"); /* путь относительно .class-
файла */
        }
        public Image getImage(int type) (
                if(type == ICON_COLOR_16x16) return icon;
                return null;
        }
        PropertyDescriptor[] getPropertyDescriptors() {
                return propDcscr;
        }
}
```

Когда новые компоненты созданы и зарегистрированы в NetBeans (или другой IDE), можно использовать их для создания реальных приложений.

Документация (на английском языке) по визуальным редакторам в IntelliJ Idea, NetBeans и Eclipse доступна по ссылкам [6,7,8] соответственно. Для Eclipse нужно установить дополнительный модуль WindowBuilder (Help-Install New Software… В Work with: выбрать --All available sites--, дождаться окончания поиска доступных расширений и выбрать WindowBuilder).

ЗАДАНИЕ 1. Создайте приложение как объект класса-наследника JFrame. Разместите в главном окне комбобокс (JComboBox), метку (JLabel), кнопку (JButton), флажок с независимой фиксацией (JCheckBox), флажок с зависимой фиксацией (JRadioButton) и таблицу (JTable). Задайте для таблицы заголовки "Язык", "Автор" и "Год". Поместите в нее следующие данные: Си, Деннис Ритчи, 1972; C++, Бьерн Страуструп, 1983; Python, Гвидо ван Россум, 1991; Java, Джеймс Гослинг, 1995; JavaScript, Брендон Айк, 1995; C#, Андерс Хейлсберг, 2001; Scala, Мартин Одерски, 2003. Запросите у системы доступные LAF. Поместите их названия в комбобокс. Создайте обработчик событий

для комбобокс так, чтобы при выборе названия LAF оформление программы переключалось на выбранный Look and Feel.

ЗАДАНИЕ 2. Создайте приложение с элементами управления для заданий 5 и 6 из лабораторной работы "Рисование и обработка событий в пакетах AWT и Swing". Подберите подходящие элементы управления.

ЗАДАНИЕ 3. Создайте приложение с элементами управления для заданий 7 и 8 из лабораторной работы "Рисование и обработка событий в пакетах AWT и Swing". Должны поддерживаться следующие возможности: остановка, запуск анимации, задание максимально возможного числа шариков. В бегущей строке должна отображаться информация о количестве двигающихся в данное время шариков и о максимально возможном их количестве.

ЗАДАНИЕ 4. Создайте приложение с элементами управления для задания 3 из лабораторной работы "Рисование и обработка событий в пакетах AWT и Swing". Главное окно должно содержать две панели - панель прорисовки графика функции и панель управления, которая должна содержать текстовые поля ввода для задания интервала изменения аргумента функции, комбобокс для выбора одной из трех функций: sin(x), sin(x*x) + cos(x*x), 2*sin(x)+cos(2*x), кнопку для выбора цвета кривой на графике. Размеры панелей должны автоматически подстраиваться под размер главного окна программы при изменении его размеров. Для получения цвета кривой используйте JColorChooser.

ЗАДАНИЕ 5. Создайте компонент, представляющий собой панель с тремя метками. Первая метка (upperLabel) занимает верхнюю часть панели. Вторая (downLabel) - нижнюю часть. Третья (centerLabel) - центр панели. Текст меток upperLabel и downLabel имеет следующие атрибуты: name - "dialog", type - Funt.BOLD | Font.Italic, size - 10, а метки centerLabel - "dialog”, Bald, 18 (см. метод setFont(java.awt.Font f)) и конструктор класса Font - Font(String name, int type, int size)). Для текста первой метки должно быть задано горизонтальное выравнивание (см. метод setHorizontlalAlignment(int a)) влево, для второй — вправо, а для третьей - по центру. Предусмотрите такие методы, чтобы можно было задавать цвет фона панели и текст тобой метки (все эти параметры - свойства компонента). Если фон окна закрашивается в методе paint(), а не paitComponent(), не забудьте вызвать paint() базового класса для прорисовки дочерних компонентов (в данном случае меток),

размещенных на панели. Если есть такая возможность, используйте NetBeans (или другую IDE) для тестирования (и, быть может, создания) компонента.

ЗАДАНИЕ 6. Создайте компонент "игральная кость", выглядящий как квадрат, внутри которого могут размещаться от одного до шести закрашенных кружков. Цвет кружков должен быть отдельным свойством, цвет фона (области внутри рамки) - другим свойством, а число кружков - третьим. Компонент должен реагировать на щелчок мышью, меняя (случайным образом) значение свойства "число кружков". Кроме того, "кость" должна иметь два состояния: активное и неактивное. В первом событие "щелчок мышкой" обрабатывается, во втором - игнорируется. Если есть такая возможность, используйте NetBeans (или другую IDE) для тестирования (и, быть может создания) компонента.

ЗАДАНИЕ 7. Создайте для компонентов из задания 5 и задания 6 соответствующие мета классы и простенькие изображения (иконки). Если есть такая возможность, добавьте компоненты на панель инструментов в NetBeans (или в другой IDE). Тогда нужно добиться, чтобы в редакторе свойств NetBeans (или в другой IDE) отображались только свойства самих компонентов, но не свойства, доставшиеся им от базовых классов.

ЗАДАНИЕ 8. Создайте игру (если есть такая возможность, то с помощью NetBeans или другой IDE) со следующими правилами:
- играют двое;
- каждый из них по очереди бросает свою собственную игральную кость;
- выброшенное значение определяет число клеток игрового поля, на которое перемещается игрок;
- с каждой клеткой поля связано положительное или отрицательное значение (очки), либо одно из следующих действий - повтор хода, потеря всех набранных очков, окончание игры;
- при каждом перемещении игрока по полю, он получает или теряет соответствующее число очков;
- игрок, первым достигший финишной клетки, получает дополнительно 50 очков;
- игра продолжается до тех пор, пока один из игроков не попадет на финишную клетку;
- победил тот, у кого на момент окончания игры больше очков (о чем должно появиться соответствующее сообщение).

Для создания игры используйте компоненты из заданий 1 и 2. Внешний вид игры должен оказаться подобным следующему:

Литература

1. И.Портянкин "Swing: Эффективные пользовательские интерфейсы, изд. 2". – М.: Лори. – 2011, 591с.
2. М.Холл, Л.Браун "Программирование для Web. Библиотека профессионала". – М.: Вильямс. – 2002, 1264с.
3. М.Морган "Java 2. Руководство разработчика". - М.: Вильямс. - 2000, 720 с.
4. R.Englander. Developing Java Beans. - O'Reilly & Assoc., Inc. - 1997. - 231 p.
5. Java Beans API specification. - Sun Microsystems, Inc. - 1997. - 114 p.
6. https://www.jetbrains.com/help/idea/gui-designer-basics.html
7. https://netbeans.apache.org/kb/docs/java/quickstart-gui_ru.html
8. https://help.eclipse.org/2021-03/index.jsp?topic=%2Forg.eclipse.wb.doc.user%2Fhtml%2Findex.html

Редактор А.А. Литвинова

ЛР № 04779 от 18.05.01.	В набор	В печать
Объем 0,5 усл.п.л., уч.-изд.л.	Офсет.	Формат 60х84/16.
Бумага тип №3.	Заказ №	Тираж 140. Цена

Издательский центр ДГТУ

Адрес университета и полиграфического предприятия:

344010, г. Ростов-на-Дону, пл. Гагарина, 1.