Программирование на Java
Программирование GUI с использованием Swing

Основные визуальные компоненты Swing

- Класс JComponent
- Методы:
  - `setEnabled(boolean enabled) // используется для // управления активностью компонента`
  - `setVisible(boolean visible) // управляет // видимостью компонента`
  - `setBackground(Color color) // изменяет цвет // заднего фона компонента`
  - `setOpaque(boolean opaque) // устанавливает // непрозрачность`
  - `getBackground() и isOpaque() // возвращают // текущий цвет заднего фона и непрозрачность // компонента`

Swing

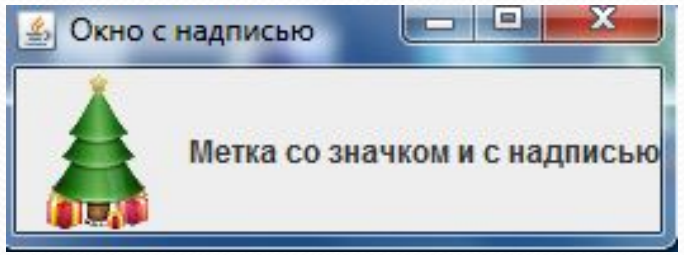
- Все визуальные компоненты библиотеки Swing унаследованы от класса JComponent. Сам этот класс является абстрактными и непосредственно не используется, но все визуальные компоненты наследуют его методы. Рассмотрим наиболее полезные из них.
- *setEnabled(boolean enabled)* используется для управления активностью компонента. При вызове этого метода с параметром **false** компонент переходит в неактивное состояние. Для каждого наследника JComponent эта «неактивность» может быть переопределена по-разному. Например, неактивная кнопка не нажимается, не реагирует на наводящуюся мышь и отображается монохромным серым цветом. Метод isEnabled() возвращает **true**, если элемент активен и **false** в противном случае.
- *setVisible(boolean visible)* управляет видимостью компонента. Мы уже использовали его для отображения окнаJFrame. Большинство элементов управления, в отличие от окна, по умолчанию являются видимыми. Метод isVisible() возвращает **false**, если элемент невидим и true в противном случае.
- С помощью метода *setBackground(Color color)* можно изменить цвет заднего фона компонента. Однако эффект будет иметь место лишь в том случае, если компонент непрозрачен (некоторые компоненты, например метка JLabel по умолчанию являются прозрачными). Непрозрачность устанавливается методом setOpaque(boolean opaque) с параметром **true**.
- Методы *getBackground() и isOpaque()* возвращают текущий цвет заднего фона и непрозрачность компонента.

Основные визуальные компоненты Swing

- **Метка JLabel**
- **Конструкторы:**
  - `JLabel(String text)` `// создает метку с надписью`
  - `JLabel(Icon image)` `// создает метку со значком`

  `// создает метку с надписью text и значком image`
  - `JLabel(String text, Icon image, int align)`
- **Пример . Конструктор класса SimpleWindow**

```
SimpleWindow()
{super("Окно с надписью");
  setDefaultCloseOperation(EXIT_ON_CLOSE);
 JLabel label = new JLabel("Метка со значком и с
надписью", new
 ImageIcon("tree.png"),SwingConstants.RIGHT);
 getContentPane().add(label);
 pack();
}
```



Swing

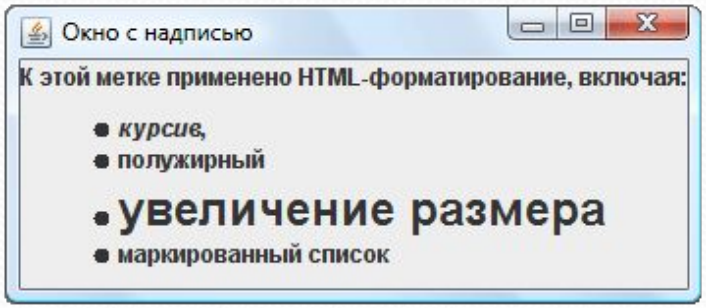
- В большинстве визуальных библиотек *метка* — один из самых простейших компонентов. Она представляет собой обычный текст, который выводится в заданном месте окна и используется для вывода вспомогательной текстовой информации: подписи к другим элементам, инструкции и предупреждения для пользователя. В Swing метка позволяет достичь более интересных эффектов. Во-первых, помимо текста можно использовать значок. Во-вторых, с ее помощью можно выводить отформатированный текст.
- Текст и значок метки можно задать в ее конструкторе. У нее есть несколько конструкторов с различными параметрами (см. слайд). Третий параметр у третьего конструктора задает выравнивание текста вместе со значком. В качестве него может быть использована одна из констант, описанных в интерфейсе SwingConstants: LEFT, RIGHT, CENTER.
- Для примера создадим окно с меткой, созданной при помощи третьего конструктора. Как и на прошлой лекции, мы будем использовать два класса, один из которых назовем SimpleWindow и унаследуем его от класса окна JFrame. В его конструкторе будут создаваться и размещаться все элементы окна. Второй класс будет создавать это окно и отображать его на экране (код будет таким же, как в примерах предыдущей лекции).
- Чтобы убедиться, что выравнивание по правому краю работает, необходимо немного растянуть окно, чтобы ширина метки стала больше оптимальной.

Основные визуальные компоненты Swing

- **Метка JLabel**
- **Пример :**

```
JLabel label = new JLabel("<html>К этой метке применено " +
"HTML-форматирование, включая: <ul><li> <i>курсив</i>," +
"<li><b>полужирный</b> <li><font size = +2> увеличение размера
</font>" + "<li>маркированный список </ul>");
```

- **Методы :**
  - getText()
  - setText(String text)
  - getIcon()
  - setIcon(Icon image)
  - getVerticalAlignment(),
  - setVerticalAlignment(int align), getHorizontalAlignment(),
  - setHorizontalAlignment(int align)
  - getIconTextGap(), setIconTextGap(int gap)



Swing

- В библиотеке Swing метка (и не только она) может быть настроена для отображения отформатированного текста в формате HTML. Для этого необходимо, чтобы строка, устанавливаемая в качестве надписи метки, начиналась с тега <html>. После этого можно использовать в строке любые теги языка HTML версии 3.2, и они будут преобразовываться в соответствующие атрибуты форматирования. В этом легко убедиться, изменив в предыдущем примере строку с вызовом конструктора (см. слайд).
- Основные методы класса JLabel:
  - *getText()* — возвращает текущий текст надписи метки
  - *setText(String text)* — задает новый текст надписи
  - *getIcon()* — возвращает значок метки
  - *setIcon(Icon image)* — устанавливает новый значок. В качестве значка обычно используется объект уже знакомого нам простого класса ImageIcon (как в вышеприведенном примере).
  - *getVerticalAlignment(), setVerticalAlignment(int align), getHorizontalAlignment(), setHorizontalAlignment(int align)* — эти четыре метода позволяют получить текущее или установить новое выравнивание (по горизонтали и вертикали) метки относительно ее границ. Возможные положения описаны в интерфейсе SwingConstants.
  - getVerticalTextPosition(), setVerticalTextPosition(int align), getHorizontalTextPosition(),setHorizontalTextPosition(int align) — эти четыре метода позволяют получить текущее или установить новое выравнивание текста относительно значка. Возможные положения описаны в интерфейсе SwingConstants.
  - *getIconTextGap(), setIconTextGap(int gap)* — позволяет получить или задать расстояние между текстом и значком метки в пикселах.

Основные визуальные компоненты Swing

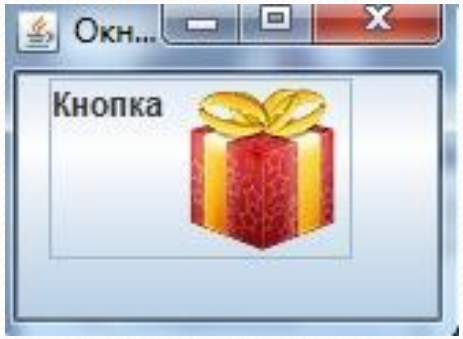
- **Метка JButton**
- **Конструкторы :**
  - `JButton(), JButton(String text),`
  - `JButton(Icon icon), JButton(String text, Icon icon)`
- **Методы :**
  - `setRolloverIcon(Icon icon)`
  - `setPressedIcon(Icon icon)`
  - `setDisableIcon(Icon icon)`
  - `setMargin(Insets margin)`
  - `setBorderPainted(boolean borderPainted),`
  - `setFocusPainted(boolean focusPainted),`
  - `setContentAreaFilled(boolean contentAreaFilled)`

- *Кнопка* — это прямоугольник с текстом (и/или значком), по которому пользователь щелкает, когда хочет выполнить какое-то действие (или о чем-то сигнализировать).
- Кнопка создается одним из пяти конструкторов, в частности *JButton(), JButton(String text), JButton(Icon icon), JButton(String text, Icon icon)*, параметры которых говорят сами за себя. Пятый конструктор мы рассмотрим в следующей лекции.
- Кроме обычного значка можно назначить кнопке еще несколько — для различных состояний.
- Метод *setRolloverIcon(Icon icon)* позволяет задать значок, который будет появляться при наведении на кнопку мыши, *setPressedIcon(Icon icon)* — значок для кнопки в нажатом состоянии, *setDisableIcon(Icon icon)* — значок для неактивной кнопки. Каждому из этих методов соответствует метод get.
- Метод *setMargin(Insets margin)* позволяет задать величину отступов от текста надписи на кнопке до ее полей. Объект класса Insets, который передается в этот метод, может быть создан конструктором с четырьмя целочисленными параметрами, задающими величину отступов: Insets(int top, int left, int bottom, int right). Метод getMargin() возвращает величину текущих отступов в виде объекта того же класса.
- Все методы класса JLabel, описанные в предыдущем разделе, присутствуют и в классе JButton. С помощью этих методов можно изменять значок и текст надписи на кнопке, а также управлять их взаимным расположением друг относительно друга и относительно края кнопки (с учетом отступов).
- Посредством методов *setBorderPainted(boolean borderPainted), setFocusPainted(boolean focusPainted), setContentAreaFilled(boolean contentAreaFilled)* можно отключать (параметром **false**) и включать обратно (параметром**true**) прорисовку рамки, прорисовку фокуса (кнопка, на которой находится фокус, выделяется пунктирным прямоугольником) и закраску кнопки в нажатом состоянии.
- На следующем слайде представлен пример создания кнопки со значком и с надписью. Изменим ее отступы и расположение текста относительно значка (текст будет выровнен влево и вверх относительно значка).

Основные визуальные компоненты Swing

- **Пример .**
- **Добавить import java.awt.*; для класса Insets**

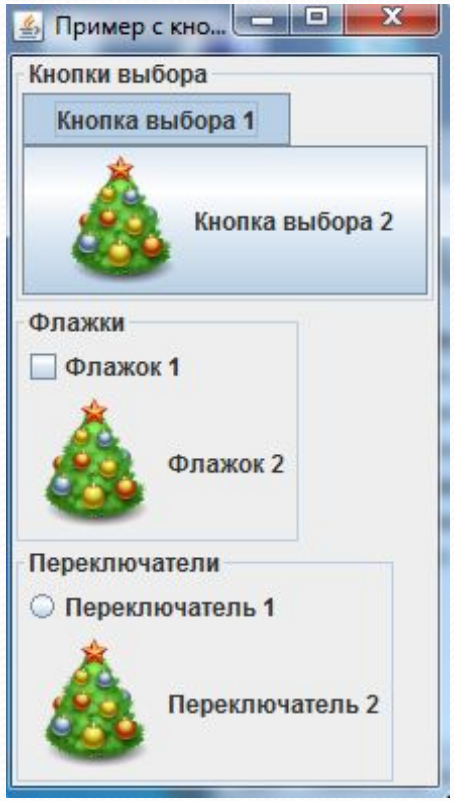
```
SimpleWindow()
{ super("Окно с кнопкой");
 setDefaultCloseOperation(EXIT_ON_CLOSE);
JButton button = new JButton("Кнопка", new
  ImageIcon("2.png"));
 button.setMargin(new Insets(0, 10, 20, 30));
 button.setVerticalTextPosition(SwingConstants.TOP);
 button.setHorizontalTextPosition(SwingConstants.LEFT);
 getContentPane().add(button);
 pack();
}
```



- Компоненты JToggleButton, JCheckBox, JRadioButton

- Конструктор:
  - `JToggleButton(String text, Icon icon, boolean selected)`

- Методы:
  - `add(AbstractButton button)`
  - `getElements()`



- Пример . Добавить import javax.swing.border.*;

```
SimpleWindow()
{super("Пример с кнопками выбора, флажками и переключателями");
 setDefaultCloseOperation(EXIT_ON_CLOSE);
 ImageIcon icon = new ImageIcon("1.png");
 Box mainBox = Box.createVerticalBox();
```

Swing

- Компонент *JToggleButton* представляет собой кнопку, которая может находиться в двух состояниях: нажатом и отпущенном. Когда пользователь щелкает мышкой по такой кнопке, она изменяет свое состояние. Именно таким образом ведут себя кнопки форматирования на инструментальной панели текстового редактора. Кнопка не только устанавливает или убирает курсивное начертание в выделенном тексте, но и сигнализирует о его наличии или отсутствии.
- Основной конструктор — *JToggleButton(String text, Icon icon, boolean selected)* создает кнопку с заданными надписью, значком и текущим состоянием. Кнопку можно перевести в требуемое состояние программным путем, вызвав метод *setSelected(boolean selected).*
  Метод isSelected() возвращает **true**, если кнопка выбрана (т.е. находится в нажатом состоянии) и **false** в противном случает.
- От класса *JToggleButton* унаследован класс *JCheckBox* — флажок. Этот класс имеет точно такой же набор конструкторов и методов, т.е. не расширяет функциональность предка. Единственное различие между ними — во внешнем виде: JCheckBox выглядит не как кнопка, а как небольшой квадратик, в котором можно поставить или убрать галочку.
- Аналогичным образом ведет себя класс *JRadioButton* — переключатель или радиокнопка, внешне выглядящая как пустой кружок, когда она не выделена и кружок с точкой в выделенном состоянии.
- Несмотря на то, что классы *JCheckBox* и *JRadioButton* ведут себя абсолютно одинаково (и аналогично их общему предку *JToggleButton)*, их принято использовать в различных ситуациях. В частности, JRadioButton предполагает выбор единственной альтернативы из нескольких возможных: несколько таких объектов объединяются в одну группу (чаще всего эта группа визуально обозначается рамкой) и при выборе одного из элементов группы предыдущий выбранный элемент переходит в состояние «не выбран».

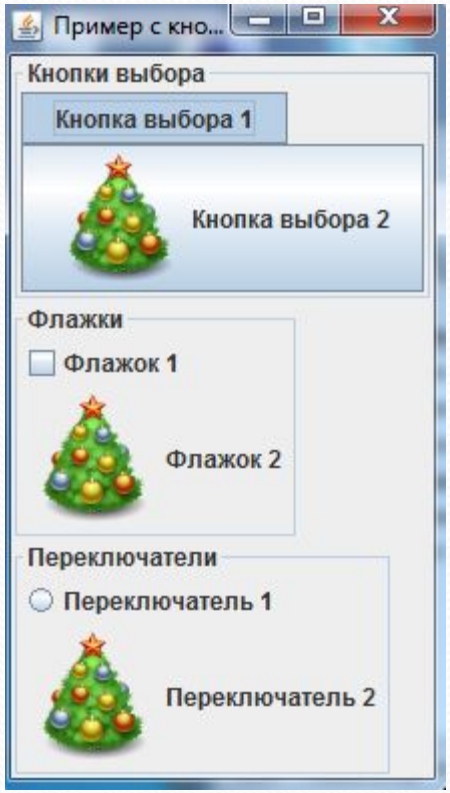
- Для того, чтобы получить такое поведение, используется специальный контейнер *ButtonGroup* — взаимоисключающая группа (создается конструктором без параметров). Если добавить в один такой контейнер несколько элементов *JRadioButton,* то выбранным всегда будет только один из них.
- В принципе, в ButtonGroup могут быть добавлены не только переключатели, но и флажки, и кнопки выбора (и даже обычные кнопки). Но при разработке интерфейса следует следовать устоявшемуся подходу, согласно которому во взаимоисключающую группу следует объединять объекты JRadioButton (и, в некоторых случаях JToggleButton), но не JCheckBox.
- Метод *add(AbstractButton button)* добавляет элемент в группу. Метод *getElements()* возвращает все ее элементы в виде коллекции Enumeration. По коллекции можно пройтись итератором и найти выделенный элемент.
- Рассмотри на следующих слайдах пример, в котором создаются две кнопки выбора, два флажка и два переключателя. Кнопки выбора и переключатели объединены в группы ButtonGroup. Для того, чтобы обвести каждую пару элементов рамкой, необходимо расположить каждую пару элементов на отдельной панели.

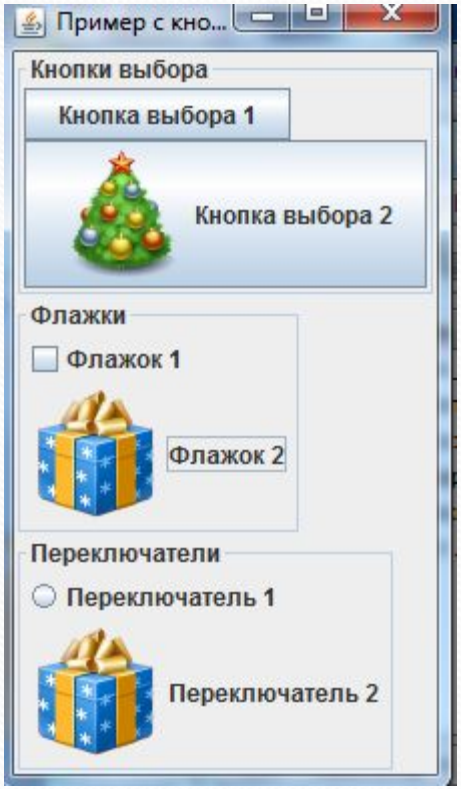
Основные визуальные компоненты Swing

```
Box box1 = Box.createVerticalBox();
JToggleButton tButton1 = new JToggleButton("Кнопка выбора 1");
JToggleButton tButton2 = new JToggleButton("Кнопка выбора 2",
 icon);
ButtonGroup bg = new ButtonGroup(); // создаем группу
                                    // взаимного исключения
bg.add(tButton1);
bg.add(tButton2);   // сделали кнопки tButton1 и tButton2
                    // взаимоисключающими
box1.add(tButton1);
box1.add(tButton2); // добавили кнопки tButton1 и tButton2
                    // на панель box1
box1.setBorder(new TitledBorder("Кнопки выбора"));
```

Основные визуальные компоненты Swing

```
Box box2 = Box.createVerticalBox();
JCheckBox check1 = new JCheckBox("Флажок 1");
JCheckBox check2 = new JCheckBox("Флажок 2", icon);
// добавим флажки
// на панель box2
box2.add(check1);
box2.add(check2);
```






```
check2.setSelectedIcon(new ImageIcon("2.png"));
box2.setBorder(new TitledBorder("Флажки"));
```

Основные визуальные компоненты Swing

```
Box box3 = Box.createVerticalBox();
JRadioButton rButton1 = new JRadioButton("Переключатель 1");
JRadioButton rButton2 = new JRadioButton("Переключатель 2",
  icon);
bg = new ButtonGroup(); // создаем группу взаимного исключения
bg.add(rButton1);
bg.add(rButton2); // сделали радиокнопки взаимоисключающими
box3.add(rButton1);
box3.add(rButton2); // добавили радиокнопки на панель box3

rButton2.setSelectedIcon(new ImageIcon("2.png"));

box3.setBorder(new TitledBorder("Переключатели"));
mainBox.add(box1);
mainBox.add(box2);
mainBox.add(box3);
setContentPane(mainBox);
pack(); }
```

- Запустив пример, мы можем пронаблюдать особенности работы кнопок выбора, флажков и переключателей. В частности, мы видим, что у флажков или переключателей рисунок заменяет элемент выделения. Но рисунок не показывает, выбран ли данный объект, что может сбить пользователя с толку. Необходимо установить отдельный рисунок для выделенного состояния, что достигается методом *setSelectedIcon(Icon icon).*
- Добавьте в нужные места команды:

```
check2.setSelectedIcon(new ImageIcon("2.gif"));      и
rButton2.setSelectedIcon(new ImageIcon("2.gif"));
```

- Пронаблюдайте произведенный эффект. Не забудьте, что файл 2.gif, равно как и файл 1.gif должны находиться в доступном для программы месте: в директории вашего проекта.

Текстовое поле JTextField

Конструкторы:

- `JTextField(int columns)`
- `JTextField(String text)`
- `JTextField(String text, int columns)`

Методы:

- `setText(String text)`
- `getText(), getText(int offset, int length)`
- `getSelectedText()`
- `replaceSelection(String content)`
- `getSelectionStart(), getSelectionEnd()`
- `setSelectionStart(int start), setSelectionEnd(int end)`
- `getCaretPosition(), setCaretPosition(int position)`
- `setCaretColor(Color color)`
- `setHorizontalAlignment(int align)`

Swing

- *Текстовое поле* — простой и часто используемый компонент, предназначенный для ввода небольших по объему (записываемых в одну строку) текстовых данных. Для создания текстового поля чаще всего используются конструкторы:
- *JTextField(int columns)* — создает пустое текстовое поле, ширина которого достаточна для размещения columns символов. При этом пользователь может вводить в текстовое поле строку какой угодно длины: она просто будет прокручиваться.
- *JTextField(String text)* — создает текстовое поле с начальным текстом text.
- *JTextField(String text, int columns)* — устанавливает и ширину и начальный текст.
- Занести текст в поле можно методом *setText(String text)*. Метод *getText()* возвращает содержимое текстового поля целиком, а *getText(int offset, int length)* — фрагмент содержимого длины length, начиная с символа offset.
- Часть текста в поле может выделяться (как программным путем, так и в результате действий пользователя). Метод *getSelectedText()* позволяет получить выделенную часть текста. Заменить выделенный текст другим можно с помощью метода *replaceSelection(String content)*. Методы *getSelectionStart() u getSelectionEnd()* возвращают границы выделенного участка, а методы *setSelectionStart(int start) u setSelectionEnd(int end)* изменяют их.
- Метод *getCaretPosition()* возвращает позицию курсора (каретки) в текстовом поле, а метод *setCaretPosition(int position)* позволяет задать ее программно. Методом *setCaretColor(Color color)* можно изменить цвет курсора.
- По умолчанию текст в поле прижимается к левому краю. Изменить это можно методом *setHorizontalAlignment(int align)*, в качестве параметра передается одна из констант выравнивания, определенных в этом же классе JTextField:LEFT, CENTER, RIGHT.

- Область для ввода текста JTextArea
- Конструктор:
  - `JTextArea(int rows, int columns)`
- Методы:
  - `setWrapStyleWord(boolean wrapStyle)`
  - `setLineWrap(boolean lineWrap)`
  - `append(String text)`
  - `insert(String text, int position)`

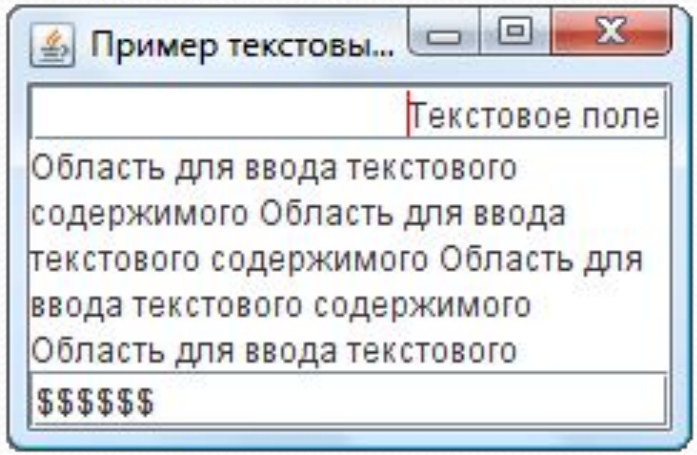
- **JTextArea** является потомком **JTextField** и наследует все его методы. В отличие от текстового поля область для ввода текста позволяет ввести не одну строку, а несколько. В связи с этим JTextArea предлагает несколько дополнительных функций.
- *Во-первых*, это способность переносить слова на соседнюю строку целиком, которой управляет метод *setWrapStyleWord(boolean wrapStyle)*. Если вызвать этот метод с параметром true, то слова не будут разрываться в том месте, где они «натыкаются» на границу компонента, а будут целиком перенесены на новую строку.
- *Во-вторых*, это способность переносить текст (то есть длинные строки будут укладываться в несколько строк вместо одной, уходящей за границы компонента. Этой способностью управляет метод *setLineWrap(boolean lineWrap).* Методы *isWrapStyleWord()* и *isLineWrap()* возвращают текущее состояние данных способностей (**true** — активирована и **false** — деактивирована).
- При создании *JTextArea* чаще всего используют конструктор *JTextArea(int rows, int columns)*, устанавливающий высоту (количество строк) и ширину (количество символов) компонента.
- Для работы со своим содержимым JTextArea дополнительно предлагает два удобных метода. Метод *append(String text)* добавляет строку text в конец уже имеющегося текста, а метод *insert(String text, int position)*вставляет ее в позицию position.
- Пронаблюдаем эти три компонента на наглядном примере. Создадим простое окно, в котором разместим их с помощью менеджера BorderLayout.
- Для того, чтобы лучше понять особенности работы текстовой области, замените по очереди **true** на **false** в вызовах методов *setLineWrap()* и *setWrapStyleWord().* Пронаблюдайте за изменением работы компонента. Изменяйте размеры окна, чтобы видеть, каким образом текст перестраивается под доступное ему пространство.

- **Пример . Добавить import java.awt.*;**

```
SimpleWindow()
{ super("Пример текстовых компонентов");
 setDefaultCloseOperation(EXIT_ON_CLOSE);
 JTextField textField = new JTextField("Текстовое поле", 20);
 textField.setCaretColor(Color.RED);
 textField.setHorizontalAlignment(JTextField.RIGHT);
 JPasswordField passwordField = new JPasswordField(20);
 passwordField.setEchoChar('$');
 passwordField.setText("пароль");
 JTextArea textArea = new JTextArea(5, 20);
 textArea.setLineWrap(true);
 textArea.setWrapStyleWord(true);
```

Основные визуальные компоненты Swing

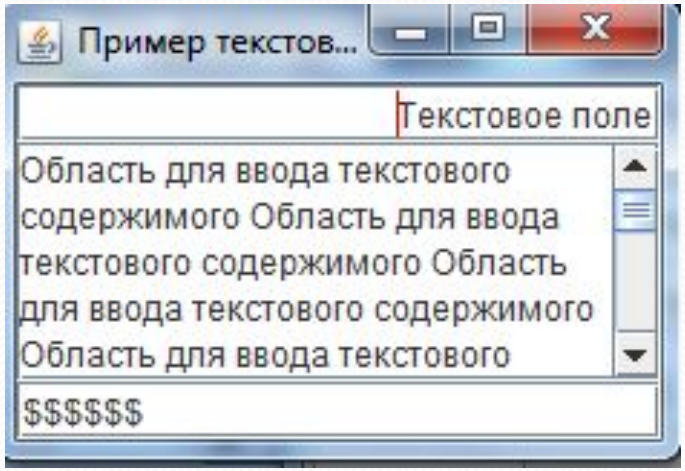
```
 for (int i = 0; i <= 20; i++)
  textArea.append("Область для ввода текстового содержимого ");
 getContentPane().add(textField, BorderLayout.NORTH);
 getContentPane().add(textArea);
 getContentPane().add(passwordField, BorderLayout.SOUTH);
 pack();
}
```



Панель прокрутки JScrollPane

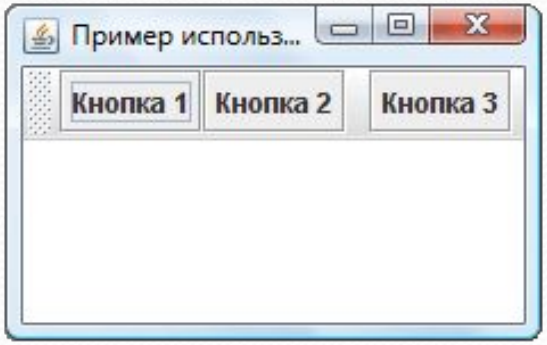
```
getContentPane().add(textArea); ->
    getContentPane().add(new JScrollPane(textArea));
```

- **Методы:**
  - `setHorizontalScrollBarPolicy(int policy)`
  - `setVerticalScrollBarPolicy(int policy)`



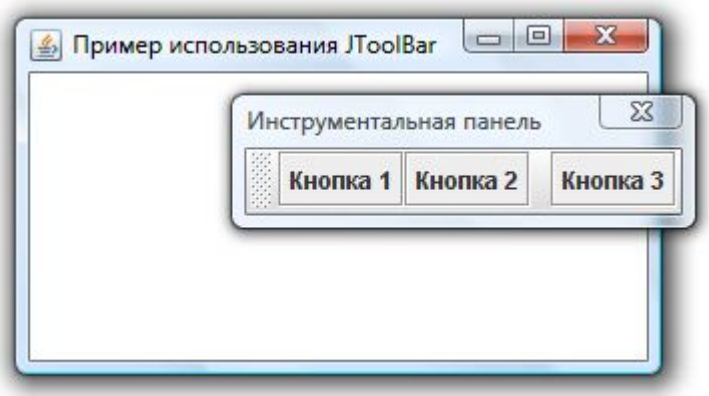
- Наблюдая за поведением компонента JTextArea в предыдущем примере, легко можно обнаружить проблемы, которые возникают, когда тексту становится «тесно» в рамках отведенного места. В зависимости от используемого менеджера расположения текст либо обрезается, уходя за границы компонента, либо раздвигает эти границы (но в любом случае остается ограничен размером окна). В таких случаях типично использование полос прокрутки, но в Swing полосы прокрутки сами собой не появляются.
- К счастью, добавить к компоненту полосы прокрутки на самом деле очень просто. Для этого служит компонент *JScrollPane* — панель прокрутки. Чаще всего она просто «надевается» на требуемый объект посредством собственного конструктора, принимающего этот объект в качестве параметра. Например, чтобы текстовая область textArea из предыдущего примера обрела полосы прокрутки, необходимо заменить команду *getContentPane().add(textArea);* на команду *getContentPane().add(**new** JScrollPane(textArea));*
- В этой команде создается панель с полосами прокрутки, в нее помещается объект textArea, а сама панель добавляется в панель содержимого окна. Теперь текст свободно прокручивается. А в случае применения менеджера FlowLayout илиBoxLayout компонент JTextArea не будет подстраиваться под свое содержимое (будет иметь предпочтительный размер, соответствующий параметрам конструктора) и, при необходимости, отображать полоски прокрутки.
- Полезными методами JScrollPane являются:
- *setHorizontalScrollBarPolicy(int policy)* — позволяет задать стратегию работы с горизонтальной полосой прокрутки. Возможные значения представлены константами HORIZONTAL_SCROLLBAR_ALWAYS (отображать всегда),HORIZONTAL_SCROLLBAR_AS_NEEDED (отображать при необходимости) и HORIZONTAL_SCROLLBAR_NEVER (не отображать никогда). Данные константы определены в интерфейсе ScrollPaneConstants.
- *setVerticalScrollBarPolicy(int policy)* позволяет задать стратегию работы с вертикальной полосой прокрутки посредством констант VERTICAL_SCROLLBAR_ALWAYS,
- VERTICAL_SCROLLBAR_AS_NEEDED иVERTICAL_SCROLLBAR_NEVER.

Пример. Инструментальная панель JToolBar




```
SimpleWindow()
{ super("Пример использования JToolBar");
 setDefaultCloseOperation(EXIT_ON_CLOSE);
 JTextArea textArea = new JTextArea(5, 20);
 getContentPane().add(textArea);
 JToolBar toolBar = new JToolBar("Инструментальная панель");
 toolBar.add(new JButton("Кнопка 1"));
 toolBar.add(new JButton("Кнопка 2"));
 toolBar.addSeparator();
 toolBar.add(new JButton("Кнопка 3"));


getContentPane().add(toolBar, BorderLayout.NORTH);
pack();
}
```

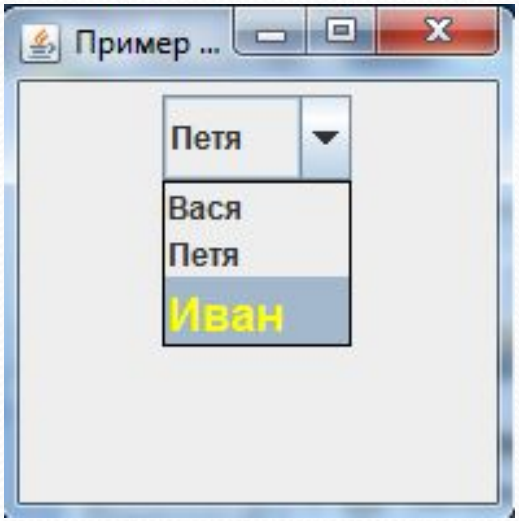


Swing

- В Swing для инструментальных панелей разработан визуальный компонент JToolBar.
- Создадим окно с менеджером расположения BorderLayout, разместим по центру область для ввода текста JTextArea, а к верхней границе прикрепим инструментальную панель с тремя кнопками и одним разделителем.
- Если запустить пример, то можно поэкспериментировать с инструментальной панелью. Попробуйте отсоединить ее от верхней границы окна и прикрепить к какой-либо другой. Отсоедините ее от границ окна так, чтобы панель стала самостоятельным окном. При этом панель всегда отображается над родительским окном, даже если именно оно, а не панель является активным. Если закрыть самостоятельную панель кнопкой с крестиком, она вернется в свое окно, в то место, где она была закреплена последний раз.
- Полезные методы JToolBar:
- Конструктор *JToolBar(String title)* создает горизонтальную панель с заданным заголовком. Горизонтальная панель предназначена для прикрепления к верхней либо нижней границе родительской панели (имеющей расположение BorderLayout). Для создания вертикальной панели используется конструктор *JToolBar(String title, int orientation),* где параметр orientation задается константой VERTICAL из интерфейса SwingConstants. Также доступны конструкторы *JToolBar()* и *JToolBar(int orientation),* создающие панель без заголовка.
- *setFloatable(boolean floatable)* — разрешает либо запрещает (по умолчанию разрешает) пользователю откреплять панель от места ее начального расположения. Ему соответствует метод *isFloatable()* возвращающий **true**, если откреплять панель разрешено.
- *add(Component component)* — добавляет на инструментальную панель новый элемент управления. Взаимосвязанные группы элементов управления принято разделять с помощью линии или пустого пространства. Метод *addSeparator()*добавляет такой разделитель.

Пример. Выпадающий список JComboBox

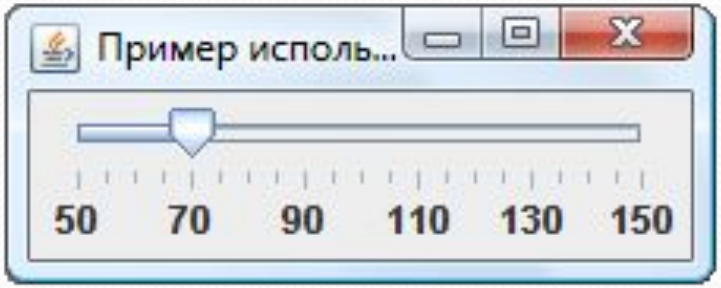
```
SimpleWindow()
{ super("Пример использования JComboBox");
  setDefaultCloseOperation(EXIT_ON_CLOSE);
  String[] elements = new String[] {"Вася", "Петя",
"<html><font size = +1 color = yellow>Иван</font>"};
 JComboBox combo = new JComboBox(elements);
 combo.setSelectedIndex(1);
 JPanel panel = new JPanel();
 panel.add(combo);
 setContentPane(panel);
 setSize(200,200);
}
```



- Выпадающий список — весьма распространенный элемент управления. Он содержит множество вариантов, из которых пользователь может выбрать один и только один, либо (если выпадающий список это позволяет) ввести свой собственный.
- Создать выпадающий список можно конструктором по умолчанию *JComboBox(),* после чего добавлять в него элементы методом *addItem(Object item)*, добавляющим новый элемент в конец списка, или методом *insertItemAt(Object item, int index),* позволяющим уточнить позицию, в которую требуется вставить элемент. Однако проще использовать конструктор, в котором сразу указываются все элементы выпадающего списка. Таких конструкторов два: *JComboBox(Object[] elements) и JComboBox(Vector elements)*. Работают они одинаково, так что это вопрос удобства разработчика: использовать массив или вектор.
- Чаще всего в выпадающий список добавляют строки, однако, как это следует из сигнатур описанных выше методов, он может содержать вообще любые объекты. Любой объект преобразуется к строке методом *toString(),* именно эта строка и будет представлять его в выпадающем списке.
- Метод *getItemAt(int index)* позволяет обратиться к произвольному элементу.
- Метод *removeAllItems()* удаляет из JComboBox все элементы, а метод *removeItem(Object item)* — конкретный элемент (при условии, что он содержался в списке).
- Метод *getSelectedIndex()* позволяет получить индекс выбранного пользователем элемента (элементы нумеруются начиная с нуля), а метод *getSelectedItem()*возвращает сам выбранный объект. Сделать конкретный элемент выбранным можно и программно, воспользовавшись методом *setSelectedIndex(int index)* или *setSelectedItem(Object item).*
- Чтобы пользователь мог ввести свой вариант, который не присутствует в списке, должен быть вызван метод *setEditable(boolean editable)* с параметром **true**. Ему соответствует метод *isEditable().*
- Рассмотрим пример, в котором создается выпадающий список из 3 элементов и выбирается 2-й.

Пример. Ползунок JSlider

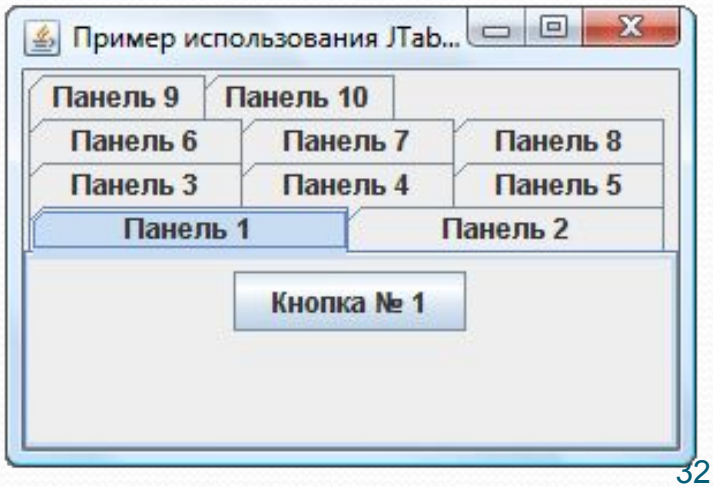
```
SimpleWindow()
{ super("Пример использования JSlider");
  setDefaultCloseOperation(EXIT_ON_CLOSE);
 JSlider slider = new JSlider(JSlider.HORIZONTAL, 50, 150, 70);
 slider.setMajorTickSpacing(20);
 slider.setMinorTickSpacing(5);
 slider.setPaintTicks(true);
 slider.setPaintLabels(true);
 slider.setSnapToTicks(true);
 JPanel panel = new JPanel();
 panel.add(slider);
 setContentPane(panel);
 pack();
}
```



- Ползунок позволяет пользователю выбрать некоторое число из диапазона доступных значений, наглядно представив этот диапазон. Против наглядности у ползунка есть один недостаток: он занимает достаточно много места.
- Основной конструктор ползунка: *JSlider(int orientation, int min, int max, int value).* Первый параметр — ориентация ползунка (HORIZONTAL или VERTICAL). Остальные параметры указывают соответственно минимальное, максимальное и текущее значение. Изменить эти значения позволяют методы *setOrientation(int), setMinimum(int min), setMaximum(int max), setValue(int value*), а получить текущие — соответствующие им методы get. Чаще всего, конечно, используется метод *getValue()* — чтобы определить, какое значение выбрал при помощи ползунка пользователь.
- Шкала ползунка может быть украшена делениями. Метод *setMajorTickSpacing(int spacing)* позволяет задать расстояние, через которое будут выводиться большие деления, а метод *setMinorTickSpacing(int spacing*) — расстояние, через которые будут выводиться маленькие деления.
  Метод *setPaintTicks(boolean paint)* включает или отключает прорисовку этих делений.
  Метод *setSnapToTicks(boolean snap)* включает или отключает «прилипание» ползунка к делениям: если вызвать этот метод с параметром **true**, пользователь сможет выбрать при помощи ползунка только значения, соответствующие делениям. Наконец, метод *setPaintLabels(boolean paint)* включает или отключает прорисовку меток под большими делениями.

Пример. Панель со вкладками JTabbedPane

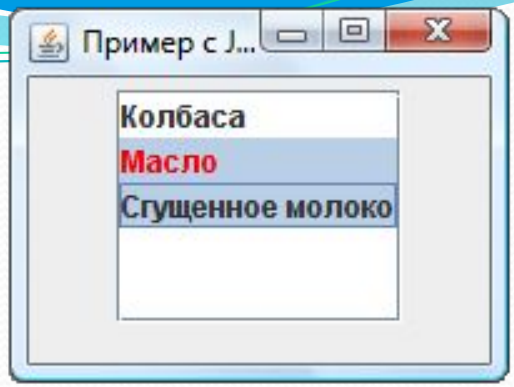
```
SimpleWindow()
{ super("Пример использования JTabbedPane");
  setDefaultCloseOperation(EXIT_ON_CLOSE);
  JTabbedPane tabbedPane = new JTabbedPane(JTabbedPane.TOP,
  JTabbedPane.WRAP_TAB_LAYOUT);
 for (int i = 1; i <= 10; i++)
  { JPanel panel = new JPanel();
    panel.add(new JButton("Кнопка № " + i));
    tabbedPane.add("Панель " + i, panel);
  }
getContentPane().add(tabbedPane);
setSize(300,200);
}
```



- Создать панель со вкладками можно простым конструктором, в котором определяется только месторасположение ярлычков (LEFT, RIGHT, TOP или BOTTOM). Но иногда бывает полезен конструктор *JTabbedPane(int orientation, int layout)*, где второй параметр принимает значения, соответствующие константам SCROLL_TAB_LAYOUT (если все ярлычки не помещаются, появляется полоса прокрутки) или WRAP_TAB_LAYOUT (ярлычки могут располагаться в несколько рядов).
- После этого можно добавлять вкладки методом *addTab()*, имеющим несколько вариантов. В частности, метод *addTab(String title, Component tab)* добавляет закладку с указанием текста ярлычка, а метод *addTab(String title, Icon icon, Component tab)*позволяет задать также и значок к ярлычку. В качестве вкладки обычно служит панель с размещенными на ней элементами управления.
- Создадим панель с десятью вкладками, на каждой из которых поместим по кнопке.

- Измените пример, чтобы вкладки располагались не в несколько рядов, а прокручивались и, кроме того, добавьте к ярлыку четвертой вкладки значок.

Пример. Список JList




```
SimpleWindow()
{ super("Пример с JList");
 setDefaultCloseOperation(EXIT_ON_CLOSE);
 Object[] elements = new Object[] {"Колбаса", "<html><font
  color = red>Масло", "Сгущенное молоко"};
 JList list = new JList(elements);
 list.setVisibleRowCount(5);
list.setSelectionMode(ListSelectionModel.SINGLE_INTERVAL_SELECT
  ION);
 list.setSelectedIndices(new int[] {1,2});
 getContentPane().setLayout(new FlowLayout());
 getContentPane().add(new JScrollPane(list));
 setSize(200,150);
}
```

Swing

- Список содержит группу элементов, аналогично выпадающему списку JComboBox, но обладает двумя отличительными особенностями. Во-первых, на экране видны одновременно несколько элементов списка. Во-вторых, пользователь может выбрать в списке не один элемент, а несколько (если установлен соответствующий режим выделения).
- Создать список можно с помощью конструктора, работающего на основе массива Object[] или вектора Vector(аналогично JComboBox). Метод *setVisibleRowCount(int count)* устанавливает количество видимых элементов списка. Остальные элементы будут уходить за его пределы или прокручиваться, если поместить список в JScrollPane(что рекомендуется).
- По умолчанию пользователь может выбрать в списке любое число элементов, держа нажатой клавишу Ctrl. Это можно изменить, вызвав метод *setSelectionMode(int mode),* где параметр задается одной из констант класса *ListSelectionModel*:
  - SINGLE_SELECTION — может быть выделен только один элемент,
  - SINGLE_INTERVAL_SELECTION — может быть выделено несколько элементов, но составляющих непрерывный интервал,
  - MULTIPLE_INTERVAL_SELECTION — может быть выделено произвольное количество смежных и несмежных элементов.
- Выделенный элемент списка (если он один) можно получить методом *getSelectedValue().* Если таких несколько, метод вернет первый из них. Метод getSelectedValues() возвращает все выделенные элементы списка в виде массиваObject[]. Аналогично работают методы *getSelectedIndex() и getSelectedIndices(),* только возвращают они не сами выделенные элементы, а их индексы. Всем этим методам соответствуют методы set, так что выделить элементы списка можно и программно.