Державний університет телекомунікацій
Навчально-науковий інститут телекомунікацій та інформатизації
Кафедра комутаційних систем

Методичне керівництво для виконання практичної роботи

Бібліотеки Java.

з дисципліни «Програмне забезпечення телекомунікаційних систем».
освітньо-кваліфікаційного рівня магістр

Київ - 2014

Укладачі: проф. каф. КС Кунах Н.І.
доцент. каф. КС Невдачина О.В.
ст. викладач каф.КС Полоневич А.П.

Методичні вказівки обговорені і схвалені
на засіданні кафедри КС

Протокол № _____
від «___» _____ 2014р.

Тема: Бібліотеки Java.

Мета заняття: Вивчення базових бібліотек у мові програмування Java. Отримання основних навичок написання найпростіших програм мовою Java.

Час заняття: 90 хвилин.

Список літератури.

1. Герберт Шилдт "Java. Полное руководство, 8-е издание", 2012

Источник: http://progbook.ru/java/

Зміст заняття.

1. **Ознайомча частина.**

Программы будут разрабатываться быстрее и занимать меньше места в том случае, если большинство стандартных алгоритмов, наиболее применимых типов данных и прочих действий будет реализовано заранее и поставляться вместе с языком программирования. Таким образом, одним из наиболее весомых преимуществ Java заключается в наличии стандартной библиотеки, входящей в состав языка. Стандартная библиотека Java растет от версии к версии, и на данный момент охватывает огромное количество возможных областей применения. С другой стороны, довольно существенная проблема заключается в том, что, поскольку стандартной библиотеке нет альтернативы, встречающиеся в ней ошибки часто кочуют из версии в версию.

Под **библиотекой** в программировании понимают набор готовых классов и интерфейсов, предназначенных для решения определенного круга задач.

Введение в библиотеку Swing

В Java есть несколько библиотек визуальных компонентов для создания графического интерфейса пользователя. Самая ранняя из них называется AWT. Считается, что при ее проектировании был допущен ряд недочетов, вследствие которых с ней довольно сложно работать. Библиотека Swing разработана на базе AWT и заменяет большинство ее компонентов своими, спроектированными более тщательно и удобно.

Полноценный графический интерфейс может быть разработан с ее помощью.

Окно JFrame

В библиотеке Swing описан класс JFrame, представляющий собой окно с рамкой и строкой заголовка (с кнопками «Свернуть», «Во весь экран» и «Закрыть»). Оно может изменять размеры и перемещаться по экрану.

Конструктор JFrame() без параметров создает пустое окно. Конструктор JFrame(String title) создает пустое окно с заголовком title.

Чтобы написать простейшую программу, выводящую на экран пустое окно, нам потребуется еще три метода:

setSize(int width, int height) — устанавливает размеры окна. Если не задать размеры, окно будет иметь нулевую высоту независимо от того, что в нем находится и пользователю после запуска придется растягивать окно вручную. Размеры окна включают не только «рабочую» область, но и границы и строку заголовка.

setDefaultCloseOperation(int operation) — позволяет указать действие, которое необходимо выполнить, когда пользователь закрывает окно нажатием на крестик. Обычно в программе есть одно или несколько окон при закрытии которых программа прекращает работу. Для того, чтобы запрограммировать это поведение, следует в качестве параметра operation передать константу EXIT_ON_CLOSE, описанную в классе JFrame.

setVisible(boolean visible) — когда окно создается, оно по умолчанию невидимо. Чтобы отобразить окно на экране, вызывается данный метод с параметром **true**. Если вызвать его с параметром **false**, окно снова станет невидимым.

Пример 1. Програмка создает пустое окно, с размерами 400 на 300, кнопками: «свернуть», «развернуть», «закрыть»

```
import javax.swing.*;  -  эта строка вставляется всегда, когда мы создаем элемент интерфейса.
```

```
import javax.swing.*;
public class MyClass {

public static void main (String [] args) {

JFrame myWindow = new JFrame("Пробное окно");

myWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

myWindow.setSize(400, 300);

myWindow.setVisible(true);

}
 }
```

Пример 2. Создание нескольких окон. Создаем новый класс с именем «SimpleWindow» и класс с именем «Program».

```
import javax.swing.*;
```

```
public class SimpleWindow  extends JFrame {

SimpleWindow(){
super("Пробное окно");
setDefaultCloseOperation(EXIT_ON_CLOSE);

setSize(250, 100);


            }

                  }
```

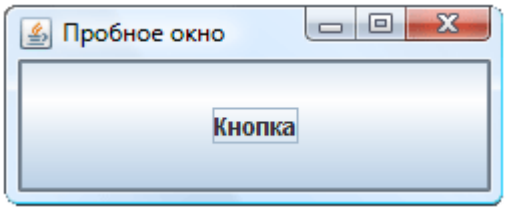
```
import javax.swing.*;
public class Program {

      public static void main (String [] args) {

      JFrame myWindow = new SimpleWindow();

      myWindow.setVisible(true);

      }

            }
```

Панель содержимого

Напрямую в окне элементы управления не размещаются. Для этого служит панель содержимого, занимающая все пространство окна. Обратиться к этой панели можно методом getContentPane() класса JFrame. С помощью метода add(Component component) можно добавить на нее любой элемент управления.

Пока используем только один элемент управления — кнопку. Кнопка описывается классом **JButton** и создается конструктором с параметром типа **String** — надписью.

Добавим кнопку в панель содержимого нашего окна командами:




```
JButton newButton = new JButton();
getContentPane().add(newButton);
```

В результате получим окно с кнопкой. Кнопка занимает всю доступную площадь окна. Такой эффект полезен не во всех программах, поэтому необходимо изучить различные способы расположения элементов на панели.

Наш файл «SimpleWindow» приобретает вид:

```
import javax.swing.*;

public class SimpleWindow  extends JFrame {
```

```
SimpleWindow(){
super("Пробное окно");
JButton newButton = new JButton();
getContentPane().add(newButton);
                  setDefaultCloseOperation(EXIT_ON_CLOSE);
setSize(250, 100);

                        }

            }
```

Класс Container (контейнер)

Элементы, которые содержат другие элементы, называются контейнерами. Все они являются потомками класса Container и наследуют от него ряд полезных методов:

add(Component component) — добавляет в контейнер элемент component;

remove(Component component) — удаляет из контейнера элемент component;

removeAll() — удаляет все элементы контейнера;

getComponentCount() — возвращает число элементов контейнера.

Кроме хранения элементов контейнер занимается их пространственным расположением и прорисовкой.

Класс JPanel (панель)

Панель **JPanel** — это элемент управления, представляющий собой прямоугольное пространство, на котором можно размещать другие элементы. Элементы добавляются и удаляются методами, унаследованными от класса Container.

В примере с кнопкой мы наблюдали, как добавленная на панель содержимого кнопка заняла все ее пространство. Это происходит не всегда. На самом деле у каждой панели есть так называемый *менеджер размещения*, который определяет стратегию взаимного расположения элементов, добавляемых на панель. Его можно изменить методом **setLayout(LayoutManager manager**). Но чтобы передать в этот метод нужный параметр, необходимо знать, какими бывают менеджеры.

Менеджер последовательного размещения FlowLayout

Самый простой менеджер размещения — FlowLayout. Он размещает добавляемые на панель компоненты строго по очереди, строка за строкой, в зависимости от размеров панели. Как только очередной элемент не помещается в текущей строке, он переносится на следующую.

Пример. Изменим конструктор класса SimpleWindow следующим образом:

```
import java.awt.FlowLayout;

import javax.swing.*;

public class SimpleWindow  extends JFrame {

SimpleWindow(){
super("Пробное окно");
setDefaultCloseOperation(EXIT_ON_CLOSE);

JPanel panel = new JPanel();

panel.setLayout(new FlowLayout());

panel.add(new JButton("Кнопка"));

panel.add(new JButton("+"));

panel.add(new JButton("-"));

panel.add(new JButton("Кнопка с длинной надписью"));

setContentPane(panel);

setSize(250, 100);

}

      }
```

Менеджеры расположения описаны в пакете java.awt. Не забывайте импортировать нужные классы.

Проанализируем текст примера. Новый менеджер расположения FlowLayout создается конструктором без параметров. Обратите внимание, в программе не используется промежуточная переменная. То есть вместо двух команд:

FlowLayout newLayout = *new* FlowLayout(); panel.setLayout(newLayout);

Мы используем одну:

panel.setLayout(*new* FlowLayout());

Менеджер граничного размещения BorderLayout

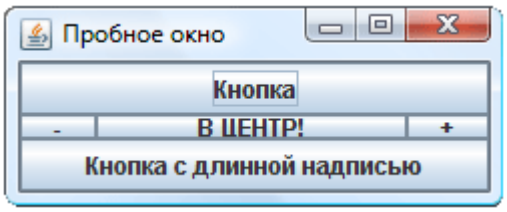
Менеджер размещения **BorderLayout** разделяет панель на пять областей: центральную, верхнюю, нижнюю, правую и левую. В каждую из этих областей можно добавить ровно по одному компоненту, причем компонент будет занимать всю отведенную для него область. Компоненты, добавленные в верхнюю и нижнюю области, будут растянуты по ширине,

добавленные в правую и левую — по высоте, а компонент, добавленный в центр, будет растянут так, чтобы полностью заполнить оставшееся пространство панели.

При добавлении элемента на панель с менеджером размещения BorderLayout, необходимо дополнительно указывать в методе add(), какая из областей имеется в виду. Для этого служат строки с названиями сторон света: `"North"`, `"South"`, `"East"`, `"West"` и `"Center"`. Но вместо них рекомендуется использовать константы, определенные в классе BorderLayout: NORTH, SOUTH, EAST, WEST и CENTER (поскольку в строке можно допустить ошибку и не заметить этого, а при попытке написать неправильно имя константы компилятор выдаст предупреждение). Если же использовать метод add() как обычно, с одним параметром, элемент будет добавлен в центр.

Панель содержимого имеет именно такое расположение, именно поэтому кнопка и занимала все окно целиком (она была добавлена в центральную область). Чтобы пронаблюдать эффект BorderLayout, добавим кнопки во все пять областей:

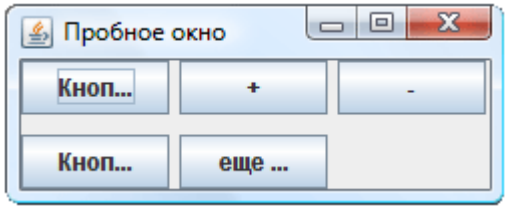
```
SimpleWindow()
{ super("Пробное окно");
 setDefaultCloseOperation(EXIT_ON_CLOSE);
getContentPane().add(new JButton("Кнопка"), BorderLayout.NORTH);
getContentPane().add(new JButton("+"), BorderLayout.EAST);
getContentPane().add(new JButton("-"), BorderLayout.WEST);
getContentPane().add(new JButton("Кнопка с длинной надписью"),
BorderLayout.SOUTH); getContentPane().add(new JButton("В ЦЕНТР!"));
setSize(250, 100); }
```



Менеджер табличного размещения GridLayout

GridLayout разбивает панель на ячейки одинаковой ширины и высоты (таким образом окно становится похожим на таблицу). Каждый элемент, добавляемый на панель с таким расположением, целиком занимает одну ячейку. Ячейки заполняются элементами по очереди, начиная с левой верхней.

Этот менеджер, в отличие от рассмотренных ранее, создается конструктором с параметрами (четыре целых числа). Необходимо указать количество столбцов, строк и расстояние между ячейками по горизонтали и по вертикали. Выполните следующий пример и пронаблюдайте эффект.




```
SimpleWindow(){
super("Пробное окно");
setDefaultCloseOperation(EXIT_ON_CLOSE);

JPanel panel = new JPanel();

panel.setLayout(new GridLayout(2,3,5,10));

panel.add(new JButton("Кнопка"));

panel.add(new JButton("+"));

panel.add(new JButton("-"));

panel.add(new JButton("Кнопка с длинной надписью"));

panel.add(new JButton("еще кнопка"));

setContentPane(panel);

setSize(250, 100);

}}
```

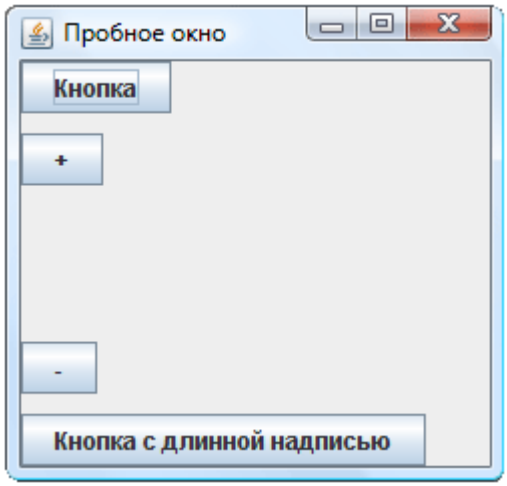
Менеджер блочного размещения BoxLayout и класс Box

Менеджер BoxLayout размещает элементы на панели в строку или в столбец.

Элементы, добавленные на панель с блочным размещением, выстраиваются один за другим. Расстояние между элементами по умолчанию нулевое. Однако вместо компонента можно добавить невидимую «распорку», единственная задача которой — раздвигать соседние элементы, обеспечивая между ними заданное расстояние. Горизонтальная распорка создается статическим методом createHorizontalStrut(int width), а вертикальная — методом createVerticalStrut(int height). Оба метода определены в классе Box, а целочисленный параметр в каждом из них определяет размер распорки.

Кроме того, на такую панель можно добавить еще один специальный элемент — своеобразную «пружину». Если размер панели будет больше, чем необходимо для оптимального размещения всех элементов, те из них, которые способны растягиваться, будут стараться заполнить дополнительное пространство собой. Если же разместить среди элементов одну или несколько «пружин», дополнительное свободное пространство будет распределяться и в эти промежутки между элементами. Горизонтальная и вертикальная пружины создаются соответственно методами createHorizontalGlue() и createVerticalGlue().

Пример: расположим четыре кнопки вертикально, поставив между двумя центральными «пружину», а между остальными — распорки в 10 пикселей.



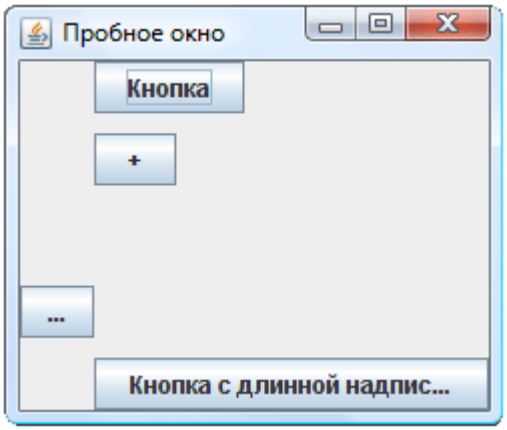

```
SimpleWindow(){ super("Пробное окно");
setDefaultCloseOperation(EXIT_ON_CLOSE);
Box box = Box.createVerticalBox();
box.add(new JButton("Кнопка"));
box.add(Box.createVerticalStrut(10));
box.add(new JButton("+"));
box.add(Box.createVerticalGlue());
box.add(new JButton("-"));
box.add(Box.createVerticalStrut(10));
box.add(new JButton("Кнопка с длинной
надписью")); setContentPane(box);
setSize(250, 100); }
```

Особенности выравнивания элементов

В примере с вертикальной панелью все кнопки оказались выровнены по левому краю. Такое выравнивание по горизонтали принято по умолчанию.

Однако при разработке окна программы может понадобиться, чтобы какие-то элементы были выровнены иначе, например, по правому краю или по центру. Для того, чтобы установить выравнивание любого визуального компонента (например, кнопки или панели), используются методы **setAlignmentX(float alignment)** — выравнивание по горизонтали и **setAlignmentY(float alignment)** — выравнивание по вертикали. В качестве параметра проще всего использовать константы, определенные в классе JComponent. Для выравнивания по горизонтали служат константы **LEFT_ALIGNMENT** (по левому краю), **RIGHT_ALIGNMENT** (по правому краю) и **CENTER_ALIGNMENT** (по центру). Для выравнивания по вертикали — **BOTTOM_ALIGNMENT** (по нижнему краю), **TOP_ALIGNMENT** (по верхнему краю) и **CENTER_ALIGNMENT** (по центру).

Однако выравнивание работает несколько иначе, чем ожидается. Чтобы это обнаружить, изменим предыдущий пример, выровняв третью кнопку по правому краю.



Для этого заменим строку:

```
box.add(new JButton("-"));
```

На три других:

```
JButton rightButton = new JButton("-");
rightButton.setAlignmentX(JComponent.RIGHT_ALIGNMENT); box.add(rightButton);
```

Ручное размещение элементов

Если в качестве менеджера размещения панели установить ***null,*** элементы не будут расставляться автоматически. Координаты каждого элемента необходимо в этом случае указать явно, при этом они никак не зависят от размеров панели и от координат других элементов. По умолчанию координаты равны нулю (т.е. элемент расположен в левом верхнем углу панели). Размер элемента также необходимо задавать явно (в противном случае его ширина и высота будут равны нулю и элемент отображаться не будет).

Координаты элемента можно задать одним из следующих методов:

setLocation(int x, int y),

setLocation(Point point)

Эти методы работают аналогично, устанавливая левый верхний угол элемента в точку с заданными координатами. Разница в способе задания точки. Можно представить точку двумя целыми числами, а можно объектом класса Point. Класс Point по сути представляет собой ту же пару чисел, его конструктор имеет вид Point(int x, int y). Получить доступ к отдельной координате можно методами getX() и getY().

Пример: Предположим, нам нужно поместить элемент b в точности в то место, которое занимает элемент a. Этого легко добиться одной строкой:

```
b.setLocation(a.getLocation());
```

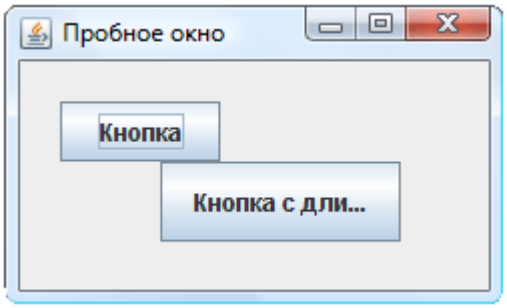
Размер элемента задается одним из двух методов:

```
setSize(int width, int height),

setSize(Dimension size)
```

Элемент b можно сделать точно такого же размера, как элемент a, выполнив команду:

```
b.setSize(a.getSize());
```

Пример: Создадим панель, с которой не будет связано никакого менеджера размещения и вручную разместим на ней две кнопки:




```
SimpleWindow(){ super("Пробное окно");
setDefaultCloseOperation(EXIT_ON_CLOSE);
JPanel panel = new JPanel();
panel.setLayout(null); JButton button = new
JButton("Кнопка"); button.setSize(80, 30);
button.setLocation(20,20); panel.add(button);
button = new JButton("Кнопка с длинной
надписью"); button.setSize(120, 40);
button.setLocation(70,50); panel.add(button); setContentPane(panel);
setSize(250, 150); }
```

Мы используем одну и ту же переменную button для обращения к обеим кнопкам (причем, второй раз ее описывать не нужно). В самом деле, осуществив все необходимые операции с первой кнопкой и зная, что обращаться к ней нам больше не понадобится, мы используем «освободившуюся» переменную для манипуляций со второй.

«Упаковка» окна

Если вместо явного указания размеров окна, вызвать метод pack(), они будут подобраны оптимальным образом с учетом предпочтений всех элементов, размещенных в этом окне.

Пример: Оцените работу этого метода, заменив в каждом из вышеприведенных примеров команду

```
setSize(250, 100);
```

на команду

```
pack();
```

Заметьте, что когда панель не имеет метода размещения, эта команда не работает (поскольку панель не имеет алгоритма для вычисления своего предпочтительного размера).

Рамки

Когда панели служат не просто для размещения элементов в соответствии с алгоритмом некоторого менеджера, а для визуального отделения их друг от друга, они оформляются с помощью рамок.

Рамка панели устанавливается методом setBorder(Border border). Параметром метода выступает рамка — объект класса Border. Это абстрактный класс, поэтому для создания рамки используются его наследники:

EmptyBorder — пустая рамка, позволяет создать отступы вокруг панели. Размеры отступов задаются в конструкторе четырьмя целыми числами.

TitledBorder — рамка с заголовком. Простейший конструктор имеет один параметр типа String (текст заголовка). Заголовок может размещаться вдоль любой стороны рамки, иметь различные начертания.

EtchedBorder — рамка с тиснением. Может быть вогнутой или выпуклой.

BevelBorder — объемная рамка (выпуклая или вогнутая). Можно настроить цвета, требуемые для получения объемных эффектов.

SoftBevelBorder — то же самое, что BevelBorder, но позволяет дополнительно скруглить углы.

LineBorder — простая рамка, нарисованная сплошной линией. Можно выбирать цвет и толщину линии, скруглить углы.

MatteBorder — рамка из повторяющегося рисунка.

CompoundBorder — объединяет две рамки, передаваемые в качестве параметров конструктору в одну новую рамку.

Все перечисленные классы описаны в пакете javax.swing.border.

Пример. В этом примере мы создадим пять панелей с различными рамками и разместим их в виде таблицы. Чтобы не описывать пять раз процедуру создания новой панели, вынесем ее в отдельный метод:

```
private JPanel createPanel(Border border, String text) { JPanel panel = new
JPanel();

 panel.setLayout(new BorderLayout());

panel.add(new JButton(text));

panel.setBorder(new CompoundBorder(new EmptyBorder(12,12,12,12), border));
return panel; }
```

Метод createPanel() создает панель с кнопкой во весь свой размер. В качестве параметра передается надпись на кнопке и рамка, которую необходимо добавить к панели. Рамка добавляется не напрямую, а путем композиции с пустой рамкой. Этот прием часто используется, чтобы рамка не прилипала к краю панели.

Теперь пять раз воспользуемся этим методом в конструкторе окна программы. Вся программа имеет вид:

```
public class SimpleWindow  extends JFrame {
      private JPanel createPanel(Border border, String text) { JPanel panel =
new JPanel();
       panel.setLayout(new BorderLayout());
      panel.add(new JButton(text));
      panel.setBorder(new CompoundBorder(new EmptyBorder(12,12,12,12),
border)); return panel; }
      SimpleWindow(){ super("Пробное окно");
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      JPanel panel = new JPanel();
      panel.setLayout(new GridLayout(2,3,5,10));
      panel.add(createPanel(new TitledBorder("Рамка с заголовком"),
"TitledBorder"));
      panel.add(createPanel(new EtchedBorder(), "EtchedBorder"));
      panel.add(createPanel(new BevelBorder(BevelBorder.LOWERED),
"BevelBorder"));
      panel.add(createPanel(new SoftBevelBorder(BevelBorder.RAISED),
"SoftBevelBorder"));
      panel.add(createPanel(new LineBorder(Color.ORANGE, 4), "LineBorder"));

      setContentPane(panel);
      pack();} }
```

Этот пример показывает, с помощью каких конструкторов создаются различные рамки и как они выглядят. В нем использованы новый класс: Color.

Класс Color предназначен для работы с цветом. В нем есть несколько констант, описывающих наиболее распространенные цвета. В частности, к таковым относится Color.ORANGE.

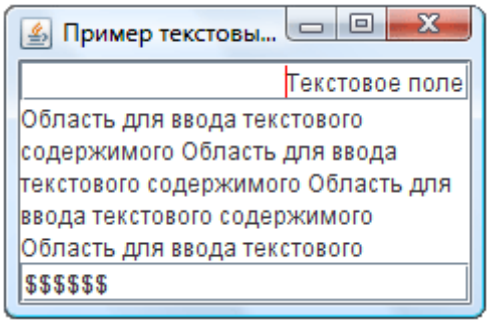
Область для ввода текста JTextArea

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.

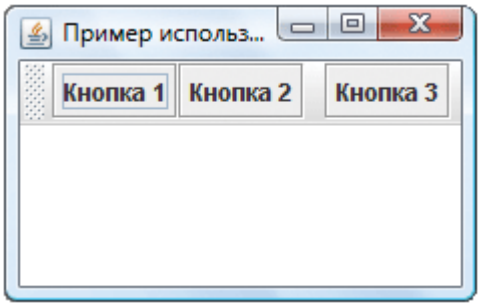


Пример. Создадим простое окно, в котором разместим их с помощью менеджера BorderLayout.

```
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);
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();} }
```

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.

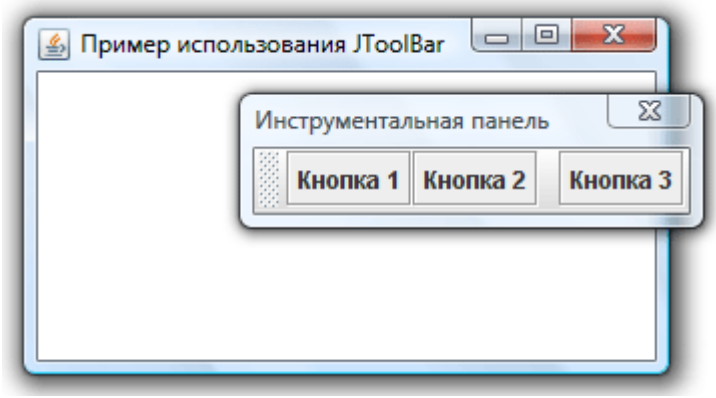
Инструментальная панель JToolBar



Большинство программных продуктов предоставляют удобные инструментальные панели, расположенные вдоль границ окна программы и содержащие кнопки, выпадающие списки и другие элементы управления, обычно соответствующие командам меню. В Swing для инструментальных панелей разработан визуальный компонент JToolBar, в котором заложена просто потрясающая функциональность.

Пример. 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:

```
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(); }
```



Запустите пример и поэкспериментируйте с инструментальной панелью. Попробуйте отсоединить ее от верхней границы окна и прикрепить к какой-либо другой.

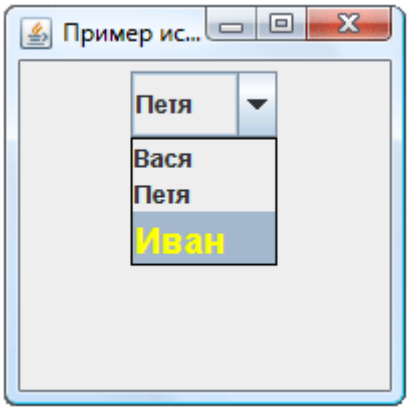
Выпадающий список JComboBox

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).

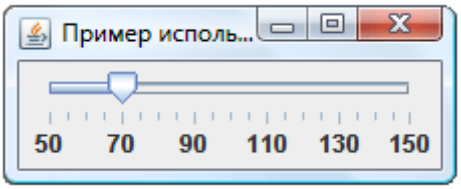
Метод getSelectedIndex() позволяет получить индекс выбранного пользователем элемента (элементы нумеруются начиная с нуля), а метод getSelectedItem() возвращает сам выбранный объект. Сделать конкретный элемент выбранным можно и программно, воспользовавшись методом setSelectedIndex(int index) или setSelectedItem(Object 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().

Пример. Создаем выпадающий список из 3 элементов и выбирается 2-й. Строка, представляющая третий элемент, использует HTML-теги. Как показывает результат, они работают не только в метках.

```
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); }
```

Ползунок JSlider



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.

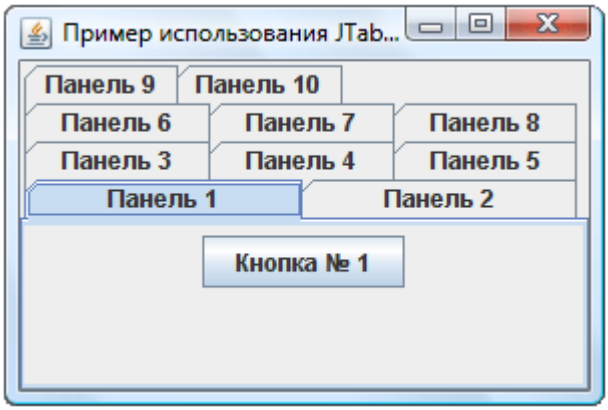
Шкала ползунка может быть украшена делениями. Метод setMajorTickSpacing(int spacing) позволяет задать расстояние, через которое будут выводиться большие деления, а метод setMinorTickSpacing(int spacing) — расстояние, через которые будут выводиться маленькие деления. Метод setPaintTicks(boolean paint) 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. использования перечисленных методов:

```
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(); }
```

Панель со вкладками JTabbedPane



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. Все эти вкладки создадим в цикле for, чтобы не писать много кода.

```
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); }

);}}
```

Питання для самоконтролю.

1. Що таке бібліотеки в мові програмування Java?
2. Які основні бібліотеки Ви знаєте?
3. Призначення бібліотек.
4. Основні можливості бібліотеки Swing.