Факультет "Информатика и системы управления"

Методические указания к лабораторной работе

по курсу "Распределенные системы обработки информации"

Программирование графики и обработка событий

Москва 2004 г.

**Оглавление**

[Цель работы 3](#_Toc82613923)

[Задание для домашней подготовки 3](#_Toc82613924)

[Задания к лабораторной работе 3](#_Toc82613925)

[Задание 1 3](#_Toc82613926)

[Задание 2 3](#_Toc82613927)

[Содержание отчета 3](#_Toc82613928)

[Контрольные вопросы. 4](#_Toc82613929)

[Литература. 5](#_Toc82613930)

[Приложение 1. Спецификация класса Calc. 6](#_Toc82613931)

[Приложение 2. Иерархия классов AWT. 9](#_Toc82613932)

[Приложение 3. Основные компоненты. 10](#_Toc82613933)

[Компонент Label 10](#_Toc82613934)

[Компонент Button 10](#_Toc82613935)

[Компонент Checkbox 11](#_Toc82613936)

[Класс CheckboxGroup 11](#_Toc82613937)

[Компонент Choice 12](#_Toc82613938)

[Компонент TextField 13](#_Toc82613939)

[Контейнер Framе 14](#_Toc82613940)

[Приложение 4. Менеджеры размещения. 16](#_Toc82613941)

[Менеджер абсолютного размещения 16](#_Toc82613942)

[Менеджер FlowLayout 18](#_Toc82613943)

[Менеджер BorderLayout 18](#_Toc82613944)

[Менеджер GridLayout 19](#_Toc82613945)

[Менеджер CardLayout 20](#_Toc82613946)

[Менеджер GridBagLayout 21](#_Toc82613947)

[Приложение 5. Обработка событий. 25](#_Toc82613948)

# Цель работы

1. Познакомиться с библиотекой java.awt.
2. Научится обрабатывать события.
3. Освоить работу с управляющими компонентами и менеджерами размещения.
4. Применить полученные знания на практике

# Задание для домашней подготовки

Ознакомиться с материалом, предоставленным в приложениях к данным методическим указаниям. Ознакомиться с текстом задания к лабораторной работе и написать программу.

# Задания к лабораторной работе

# Задание 1

Написать программу обработки щелчков на любую из трех кнопок, расположенных на панели. Каждый раз, когда пользователь щелкает мышью на какой-нибудь из трех кнопок, должен изменяться цвет панели (желтый, синий, красный).

# Задание 2

* Написать программу–калькулятор, реализующую графический интерфейс.

# Содержание отчета

Отчет должен содержать:

1. Постановку задачи, решаемой отлаженной программой.
2. Руководство пользователя отлаженной программы, содержащее описание интерфейсов всех функций программы.
3. Листинг программы с необходимыми комментариями.

# 

# Контрольные вопросы

* + - 1. Почему в программе необходимо записать обработчик события закрытия окна? Как закрыть окно в случае, если программа запущена без этого обработчика?
      2. Какие менеджеры размещения компонентов вы знаете? В чем особенность каждого из них?
      3. Измените программу так, чтобы метод getRez() вызывался при:
         1. вводе текста в одно из текстовых полей
         2. изменении состояния флажка (переключателей)
         3. выборе одного из пунктов выпадающего списка
      4. Как изменится работа программы, если внутри блока switch(){} не записывать операторы break.
      5. Назовите два способа, которыми можно задать заголовок (title) окна.

# Литература

Официальные источники:

1. Кен Арнольд, Джеймс Гослинг, Дэвид Холмс. Язык программирования Java™.
2. Официальный сайт Java — <http://java.sun.com/> (есть раздел на русском языке с учебником).
3. Java™ 2 SDK, Standard Edition Documentation — <http://java.sun.com/products/jdk/1.5/index.html>.
4. Джеймс Гослинг, Билл Джой, Гай Стил. Спецификация языка Java (The Java Language Specification — <http://www.javasoft.com/docs/books/jls/>). Перевод на русский язык — <http://www.uni-vologda.ac.ru/java/jls/index.html>
5. Официальный сайт проекта Eclipse — <http://www.eclipse.org/>.

Другое:

1. Дмитрий Рамодин. Начинаем программировать на языке Java.
2. Николай Смирнов. Java 2: Учебное пособие.
3. Картузов А. В. Программирование на языке Java.
4. Вязовик Н.А. Программирование на Java.
5. Алексей Литвинюк. Введение в интегрированную среду разработки Eclipse — <http://lib.juga.ru/article/articleview/174/1/69/>.

# Приложение 1. Спецификация класса Calc.

import java.awt.\*; //подключение пакета с классами графических компонентов

import java.awt.event.\*; // подключение пакета с интерфейсами – “слушателями”

class Calc extends Frame implements ActionListener

//наследуем класс Frame (окно) и интерфейс “слушателя” события нажатия кнопки

{

TextField t1=new TextField(8); //размер поля – 8 символов

//… другие текстовые поля

Label l1=new Label("(double)"); //метка по умолчанию содержит текст «(double)»

//… другие метки

Button b = new Button("Получить результат"); //кнопка

Checkbox cb=new Checkbox("+20%");//флажок с названием

Choice ch;//выпадающий список

public Calc(){//конструктор класса

super();

setSize(400,200);//размер окна

ch = new Choice();//конструктор выпадающего списка

ch.addItem("+");//добавляем пункты списка

//… другие пункты списка

setLayout(new FlowLayout()); //назначаем менеджер размещения компонентов (компоновщик)

add(l1);//выводим созданные компоненты на экран

add(t1);

add(ch);

//… и другие

addWindowListener(new WindowAdapter(){ //добавляем обработчик кнопки закрытия окна

public void windowClosing(WindowEvent e){

dispose(); System.exit(0);

}

}

);

b.addActionListener(this);//добавляем обработчик нажатия на кнопку

show();//делает окно видимым

}//конец конструктора

public void getRez(){//метод, вычисляющий результат

double x, //значение из первого текстового поля

Rez=0; //результат

//… вторая переменная – значение второго текстового поля

try{

x=new Double(t1.getText()).doubleValue();//извлечение числа из первого текстового поля

//…

switch(ch.getSelectedIndex()){//какой пункт выпадающего списка выбран? (номер)

case 0: Rez = x + /\*вторая переменная\*/; break;

case 1: //…

//… остальные операторы

};

if (cb.getState()) Rez\*=1.2;//если флажок нажат, умножаем результат на 1.2

//записываем текст в поле (метку) результата с помощью метода setText(String.valueOf(Rez));

} //окончание блока try

catch(Exception ex){

String s=new String(ex.getMessage());

//записываем текст в поле (метку) сообщения об ощибке с помощью метода setText("Ошибка "+s);

}// окончание блока catch

}// окончание метода getRez()

public void actionPerformed(ActionEvent e){getRez();}//реализация метода actionPerformed

//(метод вызывается нажатием на кнопку b), см. выше

public static void main(String []a){Calc clc=new Calc();} //создаем только экземпляр класса Calc и все.

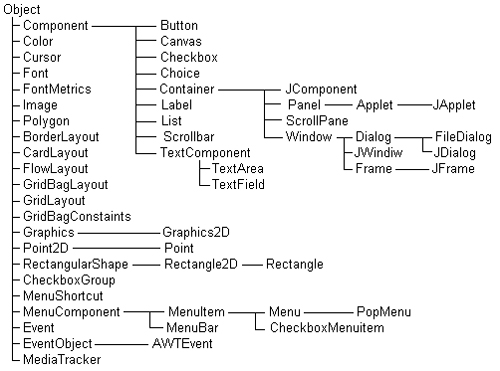
} //окончание класса Calc

# Приложение 2. Иерархия классов AWT

На рис. 1 показана иерархия основных классов AWT. Основу ее составляют готовые компоненты: Button, Canvas, Checkbox, Choice, Container, Label, List, Scrollbar, TextArea, TextField, Menubar, Menu, PopupMenu, Menultem, CheckboxMenuItem. Если этого набора не хватает, то от класса Canvas можно породить собственные "тяжелые" компоненты, а от класса Component — "легкие" компоненты.

Основные контейнеры — это классы Panel, ScrollPane, Window, Frame, Dialog, FileDialog. Свои "тяжелые" контейнеры можно породить от класса Panel, а "легкие" — от класса Сontainer.

На рис. 1 также показаны и начальные классы иерархии библиотеки Swing — классы JComponent, JWindow, JFrame, JDialog, JApplet.



**Рис. 1** Иерархия основных классов AWT

# Приложение 3. Основные компоненты

### 

### Компонент Label

Компонент Label — это просто строка текста, оформленная как графический компонент для размещения в контейнере. Текст можно поменять только методом доступа setText(string text), но не вводом пользователя с клавиатуры или с помощью мыши.

Создается объект этого класса одним из трех конструкторов:

* Label () — пустой объект без текста;
* Label (string text) — объект с текстом text, который прижимается клевому краю компонента;
* Label (String text, int alignment) — объект с текстом text и определенным размещением в компоненте текста, задаваемого одной из трех констант: CENTER, LEFT, RIGHT.

Размещение можно изменить методом доступа setAlignment(int alignment).

Остальные методы позволяют получить текст getText() и размещение getAlignment ().

#### События

В классе Label происходят события классов Component: ComponentEvent, FocusEvent, KeyEvent, MouseEvent.

### Компонент Button

Компонент Button — это кнопка стандартного для данной графической системы вида с надписью, умеющая реагировать на щелчок кнопки мыши — при нажатии она "вдавливается" в плоскость контейнера, при отпускании — становится "выпуклой".

Два конструктора Button () и Button (String label) создают кнопку без надписи и с надписью label соответственно.

Методы доступа getLabel() и setLabel (String label) позволяют получить и изменить надпись на кнопке.

Главная функция кнопки — реагировать на щелчки мыши, и прочие методы класса обрабатывают эти действия.

#### События

Кроме событий класса Component: ComponentEvent, FocusEvent, KeyEvent, MouseEvent, при воздействии на кнопку происходит событие ActionEvent.

### Компонент Checkbox

Компонент Checkbox — это надпись справа от небольшого квадратика, в котором в некоторых графических системах появляется галочка после щелчка кнопкой мыши — компонент переходит в состояние on. После следующего щелчка галочка пропадает — это состояние off. В других графических системах состояние on отмечается "вдавливанием" квадратика. В компоненте Checkbox состояния on/off отмечаются логическими значениями true/false соответственно.

Три конструктора Checkbox (), Checkbox (String label), Checkbox (String label, boolean state) создают компонент без надписи, с надписью label в состоянии off, и в заданном состоянии state.

Методы доступа getLabel(), setLabel (String label), getState(), setState (boolean state) возвращают и изменяют эти параметры компонента.

#### События

В классе Checkbox происходят события класса Component: ComponentEvent, FocusEvent, KeyEvent, MouseEvent, а при изменении состояния кнопки возникает событие ItemEvent.

В библиотеке AWT радиокнопки не образуют отдельный компонент. Вместо этого несколько компонентов Checkbox объединяются в группу с помощью объекта класса CheckboxGroup.

### Класс CheckboxGroup

Класс CheckboxGroup очень мал, поскольку его задача — просто дать общее имя всем объектам Checkbox, образующим одну группу. В него входит один конструктор по умолчанию CheckboxGroup() и два метода доступа:

* getSelectedCheckbox(), возвращающий выбранный объект Checkbox;
* setSelectedCheckbox (Checkbox box), задающий выбор.

#### Как создать группу радиокнопок

Чтобы организовать группу радиокнопок, надо сначала сформировать объект класса CheckboxGroup, а затем создавать кнопки конструкторами

Checkbox(String label, CheckboxGroup group, boolean state)

Checkbox(String label, boolean state, CheckboxGroup group)

Эти конструкторы идентичны, просто при записи конструктора можно не думать о порядке следования его аргументов.

Только одна радиокнопка в группе может иметь состояние state = true.

### Компонент Choice

Компонент Сhoice — это раскрывающийся список, один, выбранный, пункт (item) которого виден в поле, а другие появляются при щелчке кнопкой мыши на небольшой кнопке справа от поля компонента.

Вначале конструктором Choice() создается пустой список.

Затем, методом add (String text), в список добавляются новые пункты с текстом text. Они располагаются в порядке написания методов add() и нумеруются от нуля.

Вставить новый пункт в нужное место можно методом

insert (string text, int position).

Выбор пункта можно произвести из программы методом select (String text) или select(int position).

Удалить один пункт из списка можно методом remove (String text) или remove (int position), а все пункты сразу — методом removeAll().

Число пунктов в списке можно узнать методом getItemCount ().

Выяснить, какой пункт находится в позиции pos можно методом getItem(int pos), возвращающим строку.

Наконец, определение выбранного пункта производится методом getSelectedIndex(), возвращающим позицию этого пункта, или методом getSelectedItem(), возвращающим выделенную строку.

#### События

В классе Choice происходят события класса Component: ComponentEvent, FocusEvent, KeyEvent, MouseEvent, а при выборе пункта возникает событие ItemEvent.

Если надо показать на экране несколько пунктов списка, то в этом случае используется объект класса List.

### Компонент TextField

Компонент TextField — это поле для ввода одной строки текста. Ширина поля измеряется в колонках (column). Ширина колонки — это средняя ширина символа в шрифте, которым вводится текст. Нажатие клавиши <Enter> заканчивает ввод и служит сигналом к началу обработки введенного текста, т. е. при этом происходит событие ActionEvent.

В классе четыре конструктора:

* TextField() — создает пустое поле шириной в одну колонку;
* TextField(int columns) — создает пустое поле с числом колонок columns;
* TextField(String text) — создает поле с текстом text;
* TextField(String text, int columns) — создает поле с текстом text и числом колонок columns.

Интересная разновидность поля ввода — поле для ввода пароля. В таком поле вместо вводимых символов появляется какой-нибудь особый эхо-символ, чаще всего звездочка, чтобы пароль никто не подсмотрел через плечо.

Данное поле ввода получается выполнением метода setEchoChar(char echo). Аргумент echo — это символ, который будет появляться в поле. Проверить, установлен ли эхо-символ, можно логическим методом echoCharisSet(), получить эхо-символ — методом getEchoChar ().

Чтобы вернуть поле ввода в обычное состояние, достаточно выполнить метод setEchoChar(0).

#### События

Кроме событий класса Component: ComponentEvent, FocusEvent, KeyEvent, MouseEvent, при изменении текста пользователем происходит событие TextEvent, а при нажатии на клавишу <Enter> — событие ActionEvent.

### Контейнер Framе

Контейнер Frame — это полноценное готовое окно со строкой заголовка, в которую помещены кнопки контекстного меню, сворачивания окна в ярлык и разворачивания во весь экран и кнопка закрытия приложения. Заголовок окна записывается в конструкторе или методом setTitle(String title).

На кнопке контекстного меню в левой части строки заголовка изображена дымящаяся чашечка кофе — логотип Java. Вы можете установить там другое изображение методом setIconImage(image icon), создав предварительно изображение icon в виде объекта класса image.

Все элементы окна Frame вычерчиваются графической оболочкой операционной системы по правилам этой оболочки. Окно Frame автоматически регистрируется в оконном менеджере графической оболочки и может перемещаться, менять размеры, сворачиваться в панель задач (task bar) с помощью мыши или клавиатуры, как "родное" окно операционной системы.

Создать окно типа Frame можно следующими конструкторами:

* Frame () — создает окно с пустой строкой заголовка;
* Frame (String title) — записывает аргумент title в строку заголовка.

Методы класса Frame осуществляют доступ к элементам окна, но также класс Frame наследует около двухсот методов классов Component, Container и Window. В частности, наследуется менеджер размещения по умолчанию — BorderLayout.

#### События

Кроме событий класса Component: ComponentEvent, FocusEvent, KeyEvent, MouseEvent, при изменении размеров окна, его перемещении или удалении с экрана, а также показа на экране происходит событие windowEvent.

# Приложение 4. Менеджеры размещения.

Для того чтобы размещение компонентов в окне не нарушалось при движении, переносе и т.д. используются менеджеры размещения или компоновки (layout managers), называемые иногда просто компоновщиками.

Программист может воспользоваться стандартными менеджерами размещения или написать свои собственные, реализовав интерфейсы LayoutManager или LayoutManager2. К стандартным менеджерам размещения относятся:

|  |  |
| --- | --- |
| Класс | Описание |
| null | абсолютное позиционирование при размещении компонентов в контейнере |
| FlowLayout | компоненты размещаются в контейнере один за другим: слева направо, сверху вниз |
| BorderLayout | компоненты размещаются в контейнере в порядке сторон света – “Север”, “Восток”, “Юг”, “Запад”, “Центр” |
| GridLayout | компоненты размещаются в контейнере в клетках таблицы; каждый элемент занимает одну клетку |
| GridBagLayout | компоненты размещаются в контейнере в клетках таблицы; каждый элемент может занимать более одной клетки; является наиболее гибким компоновщиком |
| CardLayout | одновременно доступен один компонент контейнера; идеальный менеджер компоновки для создания составных компонентов, похожих на “записные” книжки в Windows |

Менеджеры размещения могут быть скомбинированы друг с другом.

По умолчанию в панелях и аплетах используется менеджер размещения FlowLayout, а в окнах и диалоговых окнах – BorderLayout.

Установка менеджера размещения для текущего контейнера осуществляется при помощи вызова:

контейнер.setLayout(new конструкторМенеджераРазмещения(параметры))

### Менеджер абсолютного размещения

Объявление того, что контейнер имеет менеджер размещения с абсолютным позиционированием компонентов, имеет вид:

контейнер.setLayout(null)

Установка границ каждого компонента осуществляется при помощи метода:

компонент.setSize(координатаХ, координатаY, ширина, высота)

Компонент добавляется в контейнер при помощи метода:

контейнер.add(компонент)

Приведем пример использования менеджера размещения компонентов с абсолютным позиционированием.

Пример.1. Демонстрация использования менеджера размещения компонентов с абсолютным позиционированием

import java.awt.\*;

public class Absolute extends Frame{

Button b1 = new Button("Cancel"),// Создадим две кнопки

b2 = new Button("OK" );

public void Absolute()

{

setBackground(Color.lightGray); // Установим светло-серый фон

setLayout(null);// Установим для окна менеджер абсолютного расположения

// Установим границы (расположение и размеры) для каждой кнопки

b1.setBounds( 50, 70, 50, 24);

b2.setBounds(110, 70, 50, 24);

// Добавим обе кнопки в окно

add(b1);

add(b2);

}

}

### Менеджер FlowLayout

Наиболее просто поступает менеджер размещения FlowLayout. Он укладывает в контейнер один компонент за другим слева направо как кирпичи, переходя от верхних рядов к нижним. При изменении размера контейнера "кирпичи" перестраиваются. Компоненты поступают в том порядке, в каком они заданы в методах add().

В каждом ряду компоненты могут прижиматься к левому краю, если в конструкторе аргумент align равен FlowLayout.LEFT, к правому краю, если этот аргумент FlowLayout.RIGHT, или собираться в середине ряда, если FlowLayout.CENTER.

Между компонентами можно оставить промежутки (gap) по горизонтали hgap и вертикали vgap. Это задается в конструкторе:

FlowLayout(int align, int hgap, int vgap)

Второй конструктор задает промежутки размером 5 пикселов:

FlowLayout(int align)

Третий конструктор определяет выравнивание по центру и промежутки 5 пикселов:

FlowLayout()

После формирования объекта эти параметры можно изменить методами:

setHgap(int hgap) setVgap(int vgap) setAlignment(int align)

### Менеджер BorderLayout

Менеджер размещения BorderLayout делит контейнер на пять неравных областей, полностью заполняя каждую область одним компонентом. Области получили географические названия NORTH, SOUTH, WEST, EAST и CENTER.

Метод add() в случае применения BorderLayout имеет два аргумента: ссылку на компонент comp и область region, в которую помещается компонент — одну из перечисленных выше констант:

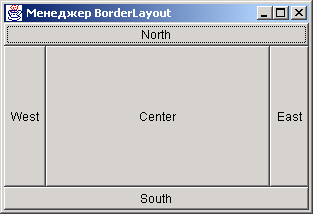
add(Component comp, String region)

Обычный метод add (Component comp) с одним аргументом помещает компонент в область CENTER.

В классе два конструктора:

* BorderLayout () — между областями нет промежутков;
* BorderLayout(int hgap int vgap) — между областями остаются горизонтальные hgap и вертикальные vgap промежутки, задаваемые в пикселах.

Если в контейнер помещается менее пяти компонентов, то некоторые области не используются и не занимают места в контейнере. Если не занята область CENTER, то компоненты прижимаются к границам контейнера.



**Рис. 2.** Области размещения BorderLayout

Менеджер размещения BorderLayout кажется неудобным: он располагает не больше пяти компонентов, последние растекаются по всей области, области имеют странный вид. Но дело в том, что в каждую область можно поместить не компонент, а панель, и размещать компоненты на ней.

### Менеджер GridLayout

Менеджер размещения GridLayout расставляет компоненты в таблицу с заданным в конструкторе числом строк rows и столбцов columns:

GridLayout(int rows, int columns)

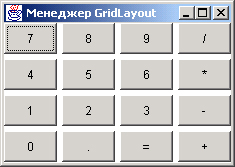
Все компоненты получают одинаковый размер. Промежутков между компонентами нет. Второй конструктор позволяет задать промежутки между компонентами в пикселах по горизонтали hgap и вертикали vgap:

GridLayout(int rows, int columns, int hgap, int vgap)

Конструктор по умолчанию GridLayout() задает таблицу размером 0x0 без промежутков между компонентами. Компоненты будут располагаться в одной строке.

Компоненты размещаются менеджером GridLayout слева направо по строкам созданной таблицы в том порядке, в котором они заданы в методах add().

Нулевое количество строк или столбцов означает, что менеджер сам создаст нужное их число.



**Рис. 3.** Пример размещения кнопок менеджером GridLayout

### 

### Менеджер CardLayout

Менеджер размещения CardLayout своеобразен — он показывает в контейнере только один, первый (first), компонент. Остальные компоненты лежат под первым в определенном порядке как игральные карты в колоде. Их расположение определяется порядком, в котором написаны методы add(). Следующий компонент можно показать методом next(Container с), предыдущий — методом previous(Container с), Последний — методом last(Container с), первый — методом first(Container с). Аргумент этих методов — ссылка на контейнер, в который помещены компоненты, обычно this.

В классе два конструктора:

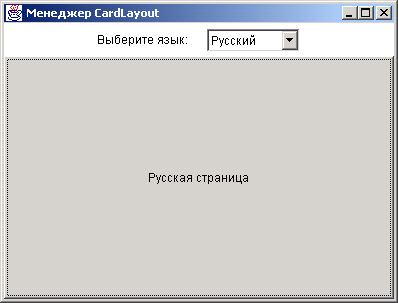
* СardLayout() — не отделяет компонент от границ контейнера;
* CardLayout(int hgap, int vgap) — задает горизонтальные hgap и вертикальные vgap поля.

Менеджер CardLayout позволяет организовать и произвольный доступ к компонентам. Метод add() для менеджера CardLayout имеет своеобразный вид:

add(Component comp, Object constraints)

Здесь аргумент constraints должен иметь тип String и содержать имя компонента. Нужный компонент с именем name можно показать методом:

show(Container parent, String name)



**Рис. 4.** Пример менеджера размещения CardLayout

### 

### Менеджер GridBagLayout

Менеджер размещения GridBagLayout расставляет компоненты наиболее гибко, позволяя задавать размеры и положение каждого компонента. Но он оказался очень сложным и применяется редко.

В классе GridBagLayout есть только один конструктор по умолчанию, без аргументов. Менеджер класса GridBagLayout, в отличие от других менеджеров размещения, не содержит правил размещения. Он играет только организующую роль. Ему передаются ссылка на компонент и правила расположения этого компонента, а сам он помещает данный компонент по указанным правилам в контейнер. Все правила размещения компонентов задаются в объекте другого класса, GridBagConstraints.

Менеджер размещает компоненты в таблице с неопределенным заранее числом строк и столбцов. Один компонент может занимать несколько ячеек этой таблицы, заполнять ячейку целиком, располагаться в ее центре, углу или прижиматься к краю ячейки.

Класс GridBagConstraints содержит одиннадцать полей, определяющих размеры компонентов, их положение в контейнере и взаимное положение, и несколько констант — значений некоторых полей. Они перечислены в табл. 1. Эти параметры определяются конструктором, имеющим одиннадцать аргументов. Второй конструктор — конструктор по умолчанию — присваивает параметрам значения, заданные по умолчанию.

**Таблица 1.** Поля класса GridBagConstraints

|  |  |
| --- | --- |
| Поле | Значение |
| anchor | Направление размещения компонента в контейнере. Константы:  CENTER, NORTH, EAST, NORTHEAST, SOUTHEAST, SOUTH, SOUTHWEST, WEST, и NORTHWEST; no умолчанию CENTER |
| fill | Растяжение компонента для заполнения ячейки. Константы: NONE, HORIZONTAL, VERTICAL, BOTH; ПО умолчанию NONE |
| gridheight | Количество ячеек в колонке, занимаемых компонентом. Целое типа int, по умолчанию 1. Константа REMAINDER означает, что компонент займет остаток колонки, RELATIVE — будет следующим по порядку в колонке |
| gridwidth | Количество ячеек в строке, занимаемых компонентом. Целое типа int, по умолчанию 1. Константа REMAINDER означает, что компонент займет остаток строки, RELATIVE — будет следующим в строке по порядку |
| gridx | Номер ячейки в строке. Самая левая ячейка имеет номер 0. По умолчанию константа RELATIVE, что означает: следующая по порядку |
| gridy | Номер ячейки в столбце. Самая верхняя ячейка имеет номер 0. По умолчанию константа RELATIVE, что означает: следующая по порядку |
| insets | Поля в контейнере. Объект класса insets; по умолчанию объект с нулями |
| ipadx, ipady | Горизонтальные и вертикальные поля вокруг компонентов; по умолчанию 0 |
| weightx,  weighty | Пропорциональное растяжение компонентов при изменении размера контейнера; по умолчанию 0,0 |

Как правило, объект класса GridBagConstraints создается конструктором по умолчанию, затем значения нужных полей меняются простым присваиванием новых значений, например:

GridBagConstraints gbc = new GridBagConstraints();

gbc.weightx = 1.0;

gbc.gridwidth = GridBagConstraints.REMAINDER;

gbc.gridheight =2;

После создания объекта gbc класса GridBagConstraints менеджеру размещения указывается, что при помещении компонента comp в контейнер следует применять правила, занесенные в объект gbc. Для этого применяется метод

add(Component comp, GridBagConstraints gbc)

Итак, схема применения менеджера GridBagLayout такова:

GridBagLayout gbl = new GridBagLayout(); // Создаем менеджер

setLayout(gbl); // Устанавливаем его в контейнер

// Задаем правила размещения по умолчанию

GridBagConstraints с = new GridBagConstraints();

Button b2 = new Button(); // Создаем компонент

c.gridwidth =2; // Меняем правила размещения

add(bl, с); // Помещаем компонент b2 в контейнер по указанным правилам

//размещения с

Button b2 = new Button(); // Создаем следующий компонент

c.gridwidth = 1; // Меняем правила для его размещения

add(b2, с); // Помещаем в контейнер

и т.д.

# Приложение 5. Обработка событий.

Java 2 использует хорошо структурированную, функционально полную и очень логичную модель обработки событий, впервые реализованную в JDK 1.1. Иерархия классов событий имеет вид:

 Object

 EventObject

 AWTEvent

 ActionEvent

 AdjustmentEvent

 ItemEvent

 TextEvent

 ComponentEvent

 ContainerEvent

 FocusEvent

 WindowEvent

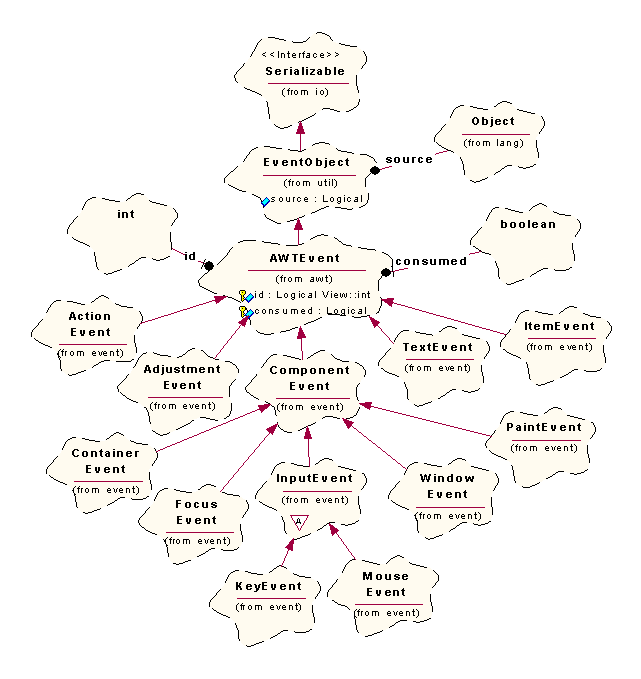
 PaintEvent

 InputEvent

 KeyEvent

 MouseEvent

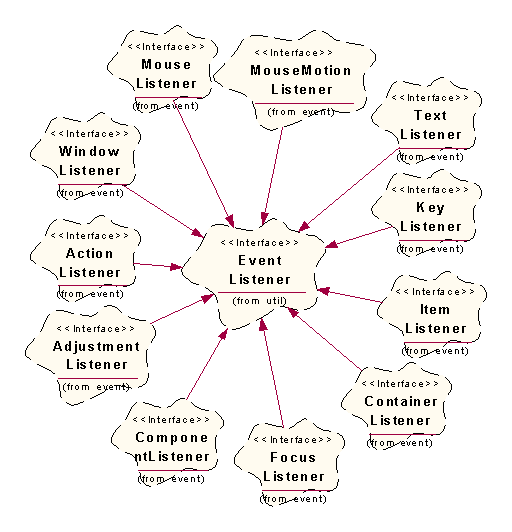
Классы и интерфейсы для работы с делегированными событиями расположены в пакете java.awt.event (см. рисунок 5).



В классах событий существуют открытые методы:

|  |  |  |
| --- | --- | --- |
| Класс | Метод | Возвращаемые данные |
| EventObject | Object getSource() | объект, вызвавший события |
|  | String toString() | строка, описывающая событие |
| AWTEvent | int getID() | идентификатор типа события |
|  | String paramString() | строка параметров события |
| ActionEvent | int getModifiers() | состояние клавиш-модификаторов |
|  | String getActionCommand() | название действия |
| AdjustmеntEvent | int getValue() | текущее значение, полученное из прокрутки |
|  | int getAdjustmentType() | тип установки прокрутки, изменившей значения |
|  | Adjustable getAdjustable() | ссылка на объект прокрутки, установивший значение |
| ItemEvent | Object getItem() | пункт списка, измененный данным событием |
|  | Container getContainer() | объект, вызвавший данное событие |
| FocusEvent | boolean isTemporary() | истину, если фокус ввода данного объекта возобновляется самостоятельно |
| PaintEvent | Graphics getGraphics() | графический контекст для данного события |
| WindowEvent | Window getWindow() | ссылка на окно, которое вызвало данное событие |
| InputEvent | boolean isShiftDown() | истина, если нажата клавиша Shift |
|  | boolean isControlDown() | истина, если нажата клавиша Control |
|  | boolean isMetaDown() | истина, если нажата клавиша Meta |
|  | boolean isAltDown() | истина, если нажата клавиша Alt |
|  | int getWhen() | время в миллисекундах с того времени, как произошло данное событие |
|  | int getModifiers() | состояние всех модификаторов клавиатуры |
|  | boolean isConsumed() | истина, если событие обработано |
| KeyEvent | int getKeyCode() | код клавиши, вызвавшей событие |
|  | char getKeyChar() | символ клавиши, вызвавшей событие |
|  | boolean isActionKey() | истина, если клавиша, вызвавшая событие, является функциональной клавишей |
| MouseEvent | int getX() | текущая абсцисса указателя мыши при генерации данного события |
|  | int getY() | текущая ордината указателя мыши при генерации данного события |
|  | Point getPoint() | положение указателя мыши при генерации данного события |
|  | getClickCount() | число нажатий на кнопку мыши при генерации данного события |
|  | isPopupTrigger() | истина, если данное событие связано с переключателем всплывающего меню для текущей операционной платформы |

Классы, ответственные за обработку событий в JDK 1.1, реализуют интерфейсы “прослушивания” событий (listeners):



|  |  |  |
| --- | --- | --- |
| Интерфейс | Переопределяемые методы | Описание |
| ActionListener | actionPerformed  (ActionEvent e) | произведено некоторое действие |
| AdjustmentListener | adjustementValueChanged (AdjustmentEvent) | изменилось значение в линии прокрутки |
| ComponentListener | componentResized (ComponentEvent e) | компонент изменил размер |
|  | componentMoved  (ComponentEvent e) | компонент изменил расположение |
|  | componentShown  (ComponentEvent e) | компонент отображен |
|  | componentHidden (ComponentEvent e) | компонент спрятан |
| ContainerListener | componentAdded  (ContainerEvent e) | компонент добавлен  в контейнер |
|  | componentRemoved (ContainerEvent e) | компонент убран из контейнера |
| FocusListener | focusGained(FocusEvent e) | компонент получил фокус ввода |
|  | focusLost (FocusEvent e) | компонент потерял фокус ввода |
| ItemListener | ItemStateChaged (ItemEvent e) | изменился выделенный пункт выбора |
| KeyListener | keyTyped(KeyEvent e) | напечатан символ |
|  | keyPressed(KeyEvent e) | нажата клавиша |
|  | keyReleased(KeyEvent e) | нажатая клавиша отпущена |
| MouseListener | mouseClicked(MouseEvent e) | нажата и отпущена кнопка мыши |
|  | mousePressed(MouseEvent e) | нажата кнопка мыши |
|  | mouseReleased(MouseEvent e) | отпущена нажатая кнопка мыши |
|  | mouseEntered(MouseEvent e) | указатель мыши вошел в область компонента |
|  | mouseExited(MouseEvent e) | указатель мыши вышел из области компонента |
| MouseMotionListener | mouseDragged(MouseEvent e) | перемещен указатель мыши при нажатой кнопке мыши |
|  | mouseMoved(MouseEvent e) | перемещен указатель мыши при не нажатой кнопке мыши |
| TextListener | textValueChanged(TextEvent e) | текстовое значение изменилось |
| WindowListener | windowOpened(WindowEvent e) | окно открыто |
|  | windowClosing(WindowEvent e) | окно закрывается |
|  | windowClosed(WindowEvent e) | окно закрыто |
|  | windowIconified(WindowEvent e) | окно минимизировано |
|  | windowDeiconified  (WindowEvent e) | окно восстановило размеры после минимизации |
|  | windowActivated  (WindowEvent e) | окно активировано (является окном ввода информации) |
|  | windowDeactivated  (WindowEvent e) | окно перестало быть активным окном |

Некоторым интерфейсам событий соответствуют открытые абстрактные классы-адаптеры, содержащие пустые методы для обработки событий:

|  |  |
| --- | --- |
| Интерфейс | Класс |
| ComponentListener | ComponentAdapter |
| ContainerListener | ContainerAdapter |
| FocusListener | FocusAdapter |
| KeyListener | KeyAdapter |
| MouseListener | MouseAdapter |
| MouseMotionListener | MouseMotionAdapter |
| MouseInputListener | MouseInputAdapter |
| WindowListener | WindowAdapted |

К действиям относятся события, поступающие от компонентов:

|  |  |
| --- | --- |
| Компонент | Генерируемое событие |
| Button | нажатие на кнопку мыши |
| MenuItem | нажатие на кнопку мыши |
| List | нажатие на кнопку мыши |
|  | нажатие на клавишу возврата каретки |
| TextField | нажатие на клавишу возврата каретки |

Управляющие компоненты регистрируют “прослушивание” событий при помощи методов вида:

void addТипКомпонентаListener(объектРеализующийПрослушивание)

и освобождаются от “прослушивания” событий при помощи соответствующих методов вида:

void removeТипКомпонентаListener(объектРеализующийПрослушивание)

|  |  |
| --- | --- |
| Класс | Метод для добавления “слушателя” события |
| Componenet | addComponentListener(ComponenetListener l)  addFocusListener(FocusListener l)  addKeyListener(KeyListener)  addMouseListener(MouseListener l)  addMouseMotionListener(MouseMotionListener l) |
| Container | addContainerListener(ContainerListener l) |
| Button | addActionListener(ActionListener l) |
| Checkbox | addItemListener(ItemListener l) |
| CheckboxMenuItem | addItemListener(ItemListener l) |
| Choice | addItemListener(ItemListener l) |
| List | addItemListener(ItemListener l)  addActionListener(ActionListener l) |
| MenuItem | addActionListener(ActionListener l) |
| Scrollbar | addAdjustnmentListener(AdjustmentListener l) |
| TextComponent | addTextListener(TextListener l) |
| TextArea | addActionListener(ActionListener l) |
| Window | addWindowListener(WindowListener l) |

Приведем пример приложения, обрабатывающего событие ActionEvent

Пример работы с событием действия (ActionEvent)

import java.awt.\*;

import java.awt.event.\*;

public class FrameActionEvent extends Frame implements ActionListener{

Label l = new Label();// Создадим метку для вывода сообщения

Button b = new Button("OK");// Создадим кнопку, генерирующую сообщение

public void FrameActionEvent()

{

setLayout(null);// Установим абсолютное размещение компонентов

setBackground(Color.lightGray);// Установим светло-серый фон

// Установим границы управляющих компонентов

l.setBounds(20, 10, 40, 20); // метки

b.setBounds(20, 40, 40, 20); // кнопки

// Зарегистрируем для кнопки “прослушивание” событий

b.addActionListener(this);

// Добавим в окно

add(l); // метку

add(b); // кнопку

}

// Реализуем обработку события действия

public void actionPerformed(ActionEvent e)

{

l.setText("I'm OK!");

}

}