GUI программирование. AWT

Лекция 5

Цели

- Дать определение GUI
- Описать пакет Abstract Window Toolkit (AWT)
- Описать различные контейнеры и компоненты
- Определить события, генерируемые компонентами
- Рассмотреть Java AWT-приложение

Graphical User Interface (GUI)

- **Graphical User Interface (GUI)** – графический интерфейс пользователя – используется для приёма от пользователя входных данных в дружественной и понятной форме.
- Любой элемент графического интерфейса создается с использованием следующей процедуры:
  - Создать элемент. Например, кнопку или надпись.
  - Определить его начальный внешний вид.
  - Определить, должен ли он занимать заданное явно положение или располагаться в любом положении, принятом по умолчанию.
  - Добавить элемент к общему экрану (или окну) интерфейса.

- Окно должно перемещаться по экрану, изменять размеры, реагировать на действия мыши и клавиатуры. В окне должны быть, как минимум, следующие стандартные компоненты:
  - Строка заголовка (**title bar**), с левой стороны которой необходимо разместить кнопку контекстного меню, а с правой — кнопки сворачивания и разворачивания окна и кнопку закрытия приложения.
  - Необязательная строка меню (**menu bar**) с выпадающими пунктами меню.
  - Горизонтальная и вертикальная полосы прокрутки (**scrollbars**).
  - Окно должно быть окружено рамкой (**border**), реагирующей на действия мыши.

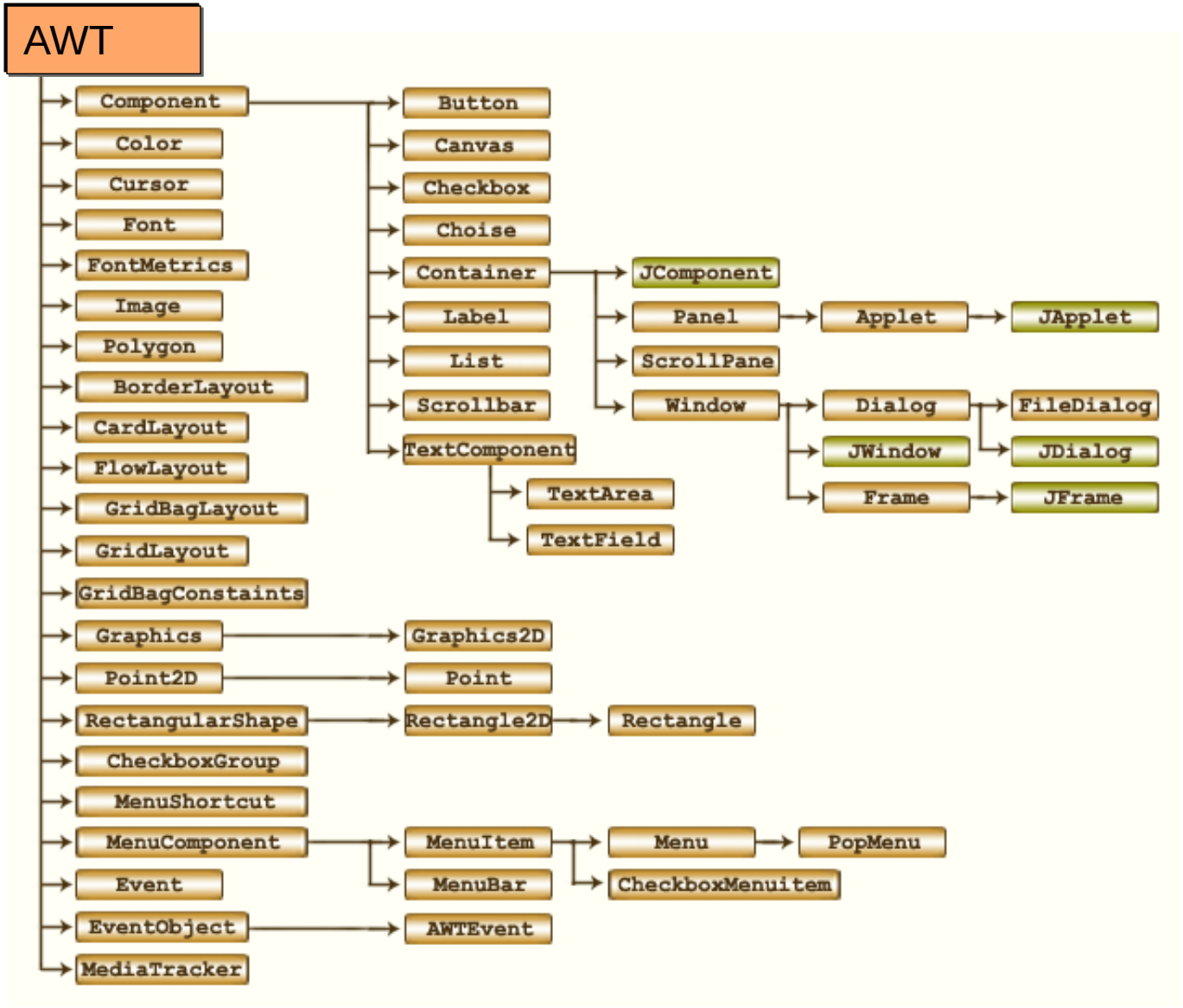
Пакет Abstract Window Toolkit

- **Abstract Window Toolkit (AWT)** – набор классов Java, которые позволяют создать графический интерфейс пользователя GUI и принимать от пользователя данные, вводимые с клавиатуры и/или мыши.
- Пакет AWT предоставляет элементы, которые позволяют создать привлекательный и эффективный интерфейс GUI.
- Основное понятие графического интерфейса пользователя – **компонент (component)** графической системы - отдельный, полностью определенный элемент, который можно использовать независимо от других элементов.
- Например, это поле ввода, кнопка, строка меню, полоса прокрутки, радиокнопка. Само окно приложения — тоже его компонент.
- Компоненты могут быть и невидимыми, например, панель, объединяющая компоненты, тоже является компонентом.

AWT

- Пакет AWT содержит:
  - **Контейнеры** (container) – это компоненты, которые могут содержать другие AWT – компоненты. Контейнер отвечает за размещение всех компонентов, которые он содержит.
  - **Компонент** – это объект, который имеет графическое представление и может быть выведен на экран. Компоненты обеспечивают взаимодействие с пользователем.
  - **Менеджеры размещения (макета)** (Layout managers). Определяют местоположение и размеры компонентов интерфейса GUI.
  - **Функциональные возможности вывода графики и рисования.** Пакет AWT поддерживает обширный набор методов вывода (отображения) графики. Все графические элементы прорисовываются относительно окна.
  - **Шрифты.** Для выбора нового шрифта должен быть создан объект Font, который будет описывать этот шрифт.
  - **События** (event) – это объект, который описывает изменение состояния источника. Событие может быть сгенерировано как последовательность взаимодействий человека с элементами графического интерфейса пользователя.

AWT


AWT
Component
Color
Cursor
Font
FontMetrics
Image
Polygon
BorderLayout
CardLayout
FlowLayout
GridBagLayout
GridLayout
GridBagConstaints
Graphics
Point2D
RectangularShape
CheckboxGroup
MenuShortcut
MenuComponent
Event
EventObject
MediaTracker
Button
Canvas
Checkbox
Choise
Container
Label
List
Scrollbar
TextComponent
TextArea
TextField
JComponent
Panel
ScrollPane
Window
Applet
JApplet
Dialog
FileDialog
JWindow
JDialog
Frame
JFrame
Graphics2D
Point
Rectangle2D
Rectangle
MenuItem
MenuBar
Menu
PopMenu
CheckboxMenuitem
AWTEvent

Контейнеры

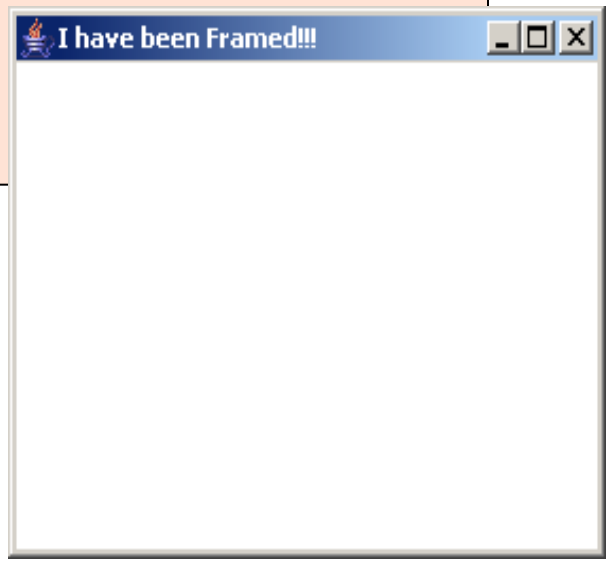
- **Контейнер** - область, которая может содержать элементы.
- Существует класс `Container` в пакете `java.awt`, от которого наследуются напрямую или косвенно два широко используемых контейнера – `Frame` и `Panel`.
  - `Frame` – это отдельное окно и его границы (рамка).
  - `Panel` – это область без визуально обозначенных границ (без рамки), которая содержится внутри окна, предоставленного браузером или средством просмотра апплетов (appletviewer).

Контейнеры – Frame (окно)

- Frame - окно, которое не зависит ни от апплета, ни от браузера.
- Является подклассом класса Window и имеет панель заголовка (**title bar**), панель меню (**menu bar**), границы (**borders**) и угловые маркеры изменения размера (**resizing corners**).
- Может работать, как компонент или как контейнер.
- Создаётся с использованием следующих конструкторов:
  - **`Frame()`**
    - Создаёт окно (Frame), которое является невидимым
  - **`Frame(String Title)`**
    - Создаёт невидимое окно с заданным заголовком
- Чтобы сделать окно `Frame` видимым, используется метод:
  - **`setVisible();`**

Пример

```
import java.awt.*;
class FrameDemo extends Frame
{
   public FrameDemo (String title){
      super(title);
   }
   public static void main (String args[]){
      FrameDemo objFr = new FrameDemo("I have been Framed!!!");
      objFr.setSize(500,500);
      objFr.setVisible(true);
   }
}
```



Контейнеры – Panel (панель)

- Panel (панель) используется для объединения нескольких компонентов в группу.
- Самый простой способ создания панели – вызов её конструктора `Panel()` .
- Так как панель не может быть выведена сама по себе, её необходимо добавить в окно (frame).
- Само окно (frame) будет видимым только после вызова двух методов установки – `setSize()` и `setVisible()`.

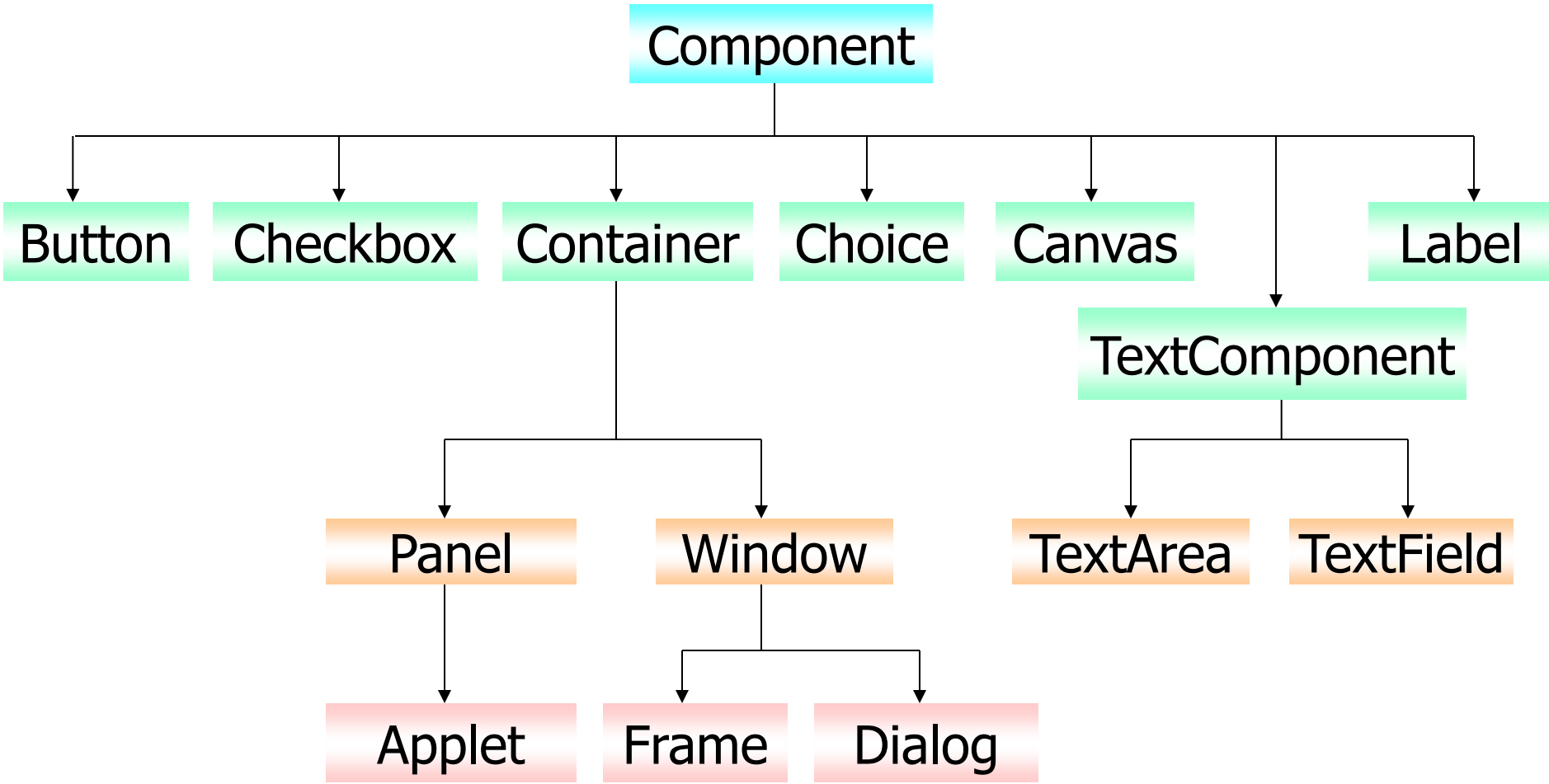
Пример

```
import java.awt.*;
class PanelTest extends Panel
{
   public static void main(String args[])
   {
      PanelTest objPanel = new PanelTest();
      Frame objFr = new Frame("Testing a Panel!");
      objFr.add(objPanel);
      objFr.setSize(400,400);
      objFr.setVisible(true);
   }
   }
```

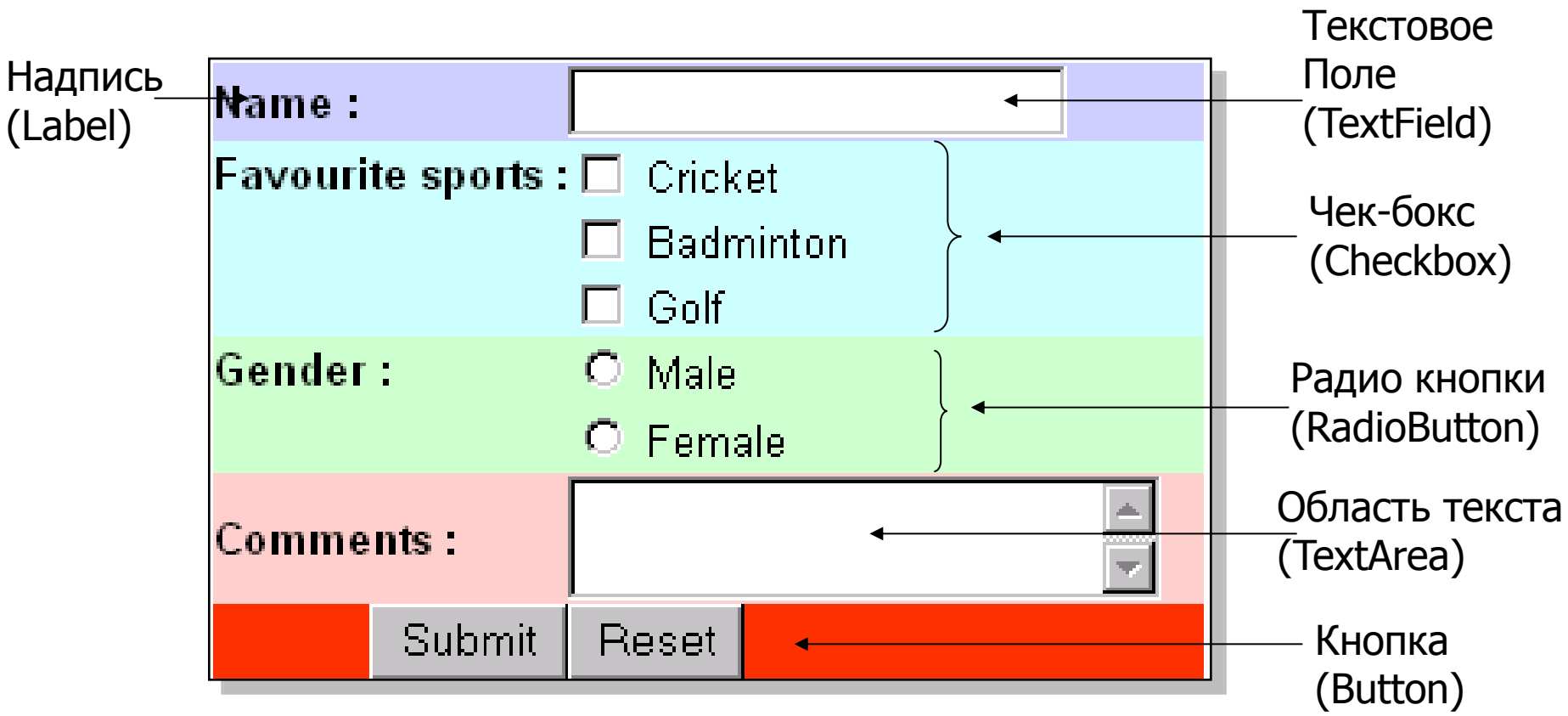
Компонент

- **Компонент** –это элемент, который может быть помещен в GUI.
- Текстовые поля (Textfields), надписи (Labels), чек-боксы (Checkboxes), области текста (Textareas) являются примерами компонентов.
- Некоторые усовершенствованные компоненты включают полосы прокрутки (scrollbars), скроллируемые панели (scrollpanes) и диалоговые элементы (dialogs).

Иерархия классов в языке Java



Различные компоненты



Надпись (Label)

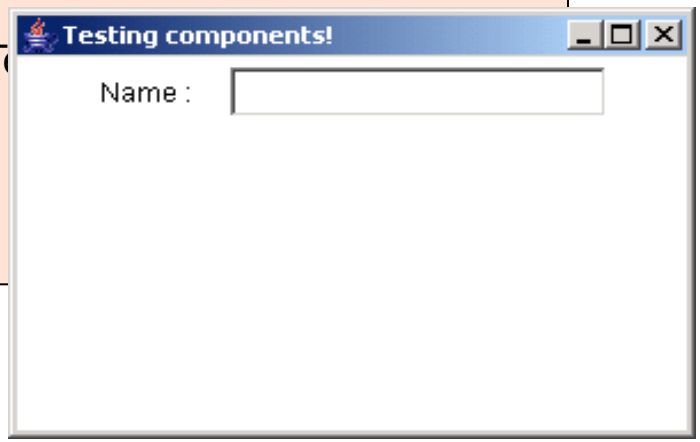
- **Надпись (Label)** главным образом используются для описания функций элемента.
- Не могут редактироваться пользователем.
- Создаются с использованием одного из следующих конструкторов:
  - **`Label( )`**
    Создаёт пустую надпись
  - **`Label(String labeltext)`**
    Создаёт надпись с заданным текстом
  - **`Label(String labeltext, int alignment)`**
    Создаёт надпись с текстом, который выравнивается заданным образом, где способ выравнивания alignment может быть `Label.LEFT`, `Label.RIGHT` или `Label.CENTER`

Текстовое поле (TextField)

- **TextField** - элемент GUI, используемый для ввода текста.
- Главным образом принимает одну строку ввода.
- Создаётся с использованием одного из следующих конструкторов:
  - `TextField()`
    Создаёт новое текстовое поле
  - `TextField(int columns)`
    Создаёт новое текстовое поле с заданным количеством столбцов
  - `TextField(String s)`
    Создаёт новое текстовое поле с заданной строкой
  - `TextField(String s, int columns)`
    Создаёт новое текстовое поле с заданной строкой и с заданным количеством столбцов

Пример

```
import java.awt.*;
class AcceptName extends Frame
{
    TextField txtName = new TextField(20);
    Label lblName = new Label("Name :");
    public AcceptName (String title)
    {
        super(title);
        setLayout(new FlowLayout());
        add(lblName);
        add(txtName);
    }
    public static void main(String args[])
    {
        AcceptName objAccName = new AcceptName ("T
        objAccName.setSize(300,200);
        objAccName.show();
    }
}
```

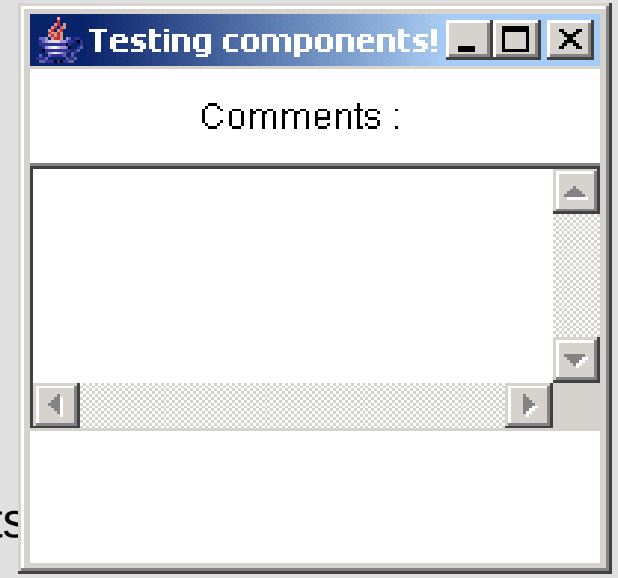


Область текста (TextArea)

- Используется, когда текст должен приниматься в виде двух или большего количества строк.
- Включает полосу прокрутки (scrollbar).
- Область текста TextArea может быть создана с использованием одного из следующих конструкторов:
  - **`TextArea()`**
    Создаёт новую область текста.
  - **`TextArea(int rows, int cols)`**
    Создаёт новую область текста с заданным количеством строк и столбцов
  - **`TextArea(String text)`**
    Создаёт новую область текста с заданной строкой.
  - **`TextArea(String text, int rows, int cols)`**
    Создаёт новую область текста с заданной строкой, с заданным количеством
    строк и столбцов.
  - **`TextArea (String text, int rows, int cols, int scrollbars)`**
    Создаёт новую область текста с заданной строкой, с заданным количеством
    строк и столбцов, а также с видимой полосой прокрутки.

Пример

```
import java.awt.*;
import java.awt.event.*;
class TextComments extends Frame
{
   TextArea txtComment = new TextArea(5,25);
   Label lblCom = new Label("Comments :");
   public TextComments(String title)
   {
      super(title);
      add(lblCom);
      add(txtComment);
}
public static void main(String args[]){
   TextComments ObjComment = new TextComments
   ObjComment.setSize(200,200);
    }
}
```



Кнопки

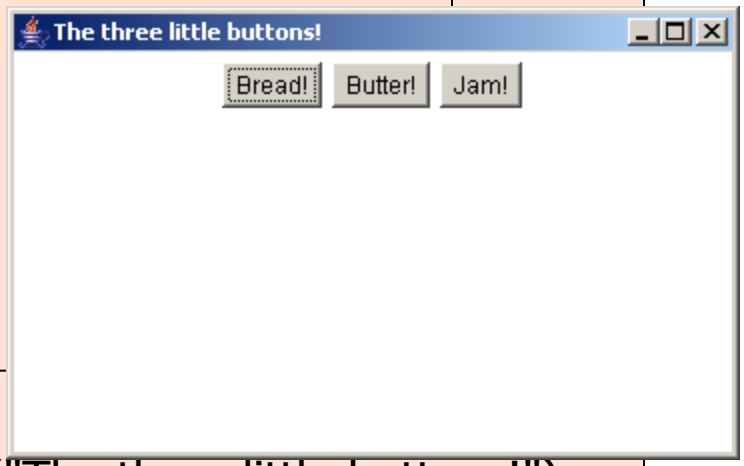
- Часть интерфейса GUI
- Самый простой способ перехватить действие пользователя.
- Кнопки в Java могут быть созданы с использованием одного из следующих конструкторов:
  - **`Button()`**

    Создаёт новую кнопку.
  - **`Button(String text)`**

    Создаёт новую кнопку с заданной строкой.

Пример

```
import java.awt.*;
import java.awt.event.*;
class ButtonTest extends Frame
{
   Button btnBread = new Button("Bread!");
   Button btnButter = new Button("Butter!");
   Button btnJam = new Button("Jam!");
   public ButtonTest(String title)
   {
      super(title);
      setLayout(new FlowLayout());
      add(btnBread);
      add(btnButter);
      add(btnJam);
   public static void main(String args[])
   {
      ButtonTest ObjTest = new ButtonTest("The three little buttons!");
      ObjTest.setSize(500,500);
      }
}
```

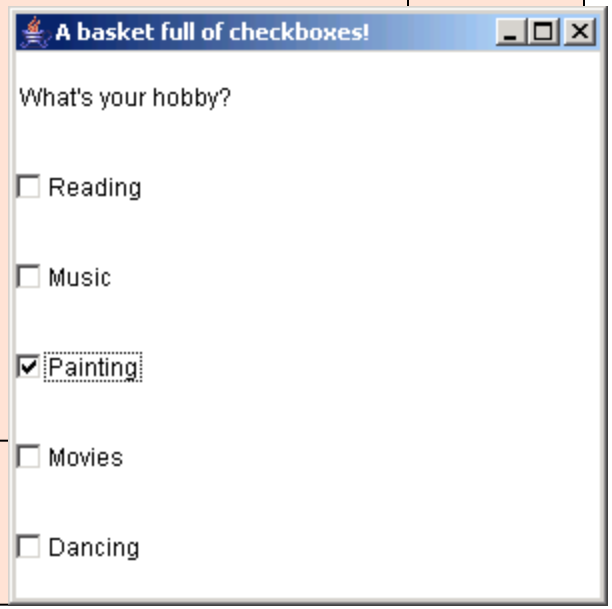


Checkbox

- Используется для многовариантного ввода пользователя, при котором пользователь может выбрать чек-бокс или отменить выбор простым щелчком мыши по нему.
- Чек-бокс в Java может быть создан с использованием одного из следующих конструкторов:
  - **`Checkbox()`**
    Создаёт пустой чек-бокс без надписи.
  - **`Checkbox(String text)`**
    Создаёт чек-бокс с заданной строкой в качестве надписи.
  - **`Checkbox(String text, boolean on)`**
    Создаёт чек-бокс с заданной строкой в качестве надписи, а также позволяет устанавливать состояние чек-бокса, задавая значение логической переменной, как true или false.

Пример

```
import java.awt.*;
import java.awt.event.*;
class Hobbies extends Frame
{
    Checkbox cboxRead = new Checkbox("Reading",false);
    Checkbox cboxMus = new Checkbox("Music",false);
    Checkbox cboxPaint = new Checkbox("Painting",false);
    Checkbox cboxMovie = new Checkbox("Movies",false);
    Checkbox cboxDance = new Checkbox("Dancing",false);
    Label lblQts = new Label("What's your hobby?" );
    public Hobbies(String str )
    {
        super(str);
        setLayout(new GridLayout(6,1));
        add(lblQts);
        add(cboxRead);
        add(cboxMus);
        add(cboxPaint);
        add(cboxMovie);
        add(cboxDance);
            ObjHobby.setSize(300,300);
              }
}
```



Радио кнопки

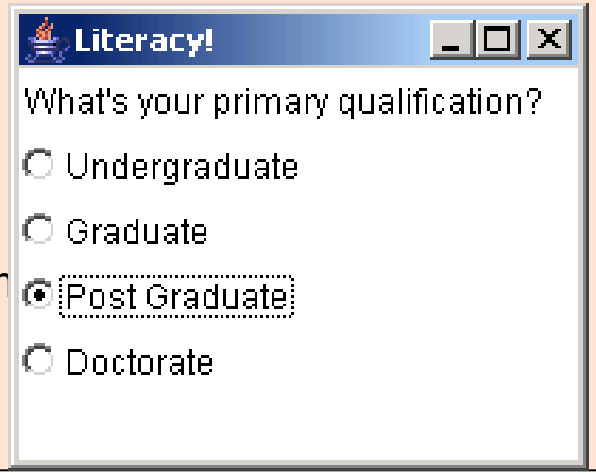
- Используется, как кнопка варианта при выборе только одного из нескольких вариантов.
- Из группы радиокнопок может быть выбрана только одна кнопка.
- Сначала создаётся объект группы радиокнопок CheckboxGroup:
  - `CheckboxGroup cg=new CheckboxGroup();`
- Затем создаётся каждая радиокнопка:
  - `Checkbox m=Checkbox(“male”,cg,true);`
  - `Checkbox f=Checkbox(“female”,cg,false);`

Пример

```
import java.awt.*;
import java.awt.event.*;
class Qualification extends Frame
{
    addWindowListener(new WindowAdapter()
        {
        public void windowClosing(WindowEvent we)
            {
                setVisible(false);
                System.exit(0);
            }
        });
    }

    public static void main(String args[])
    {
        Qualification ObjQualification = new Qualification

    }
}
```

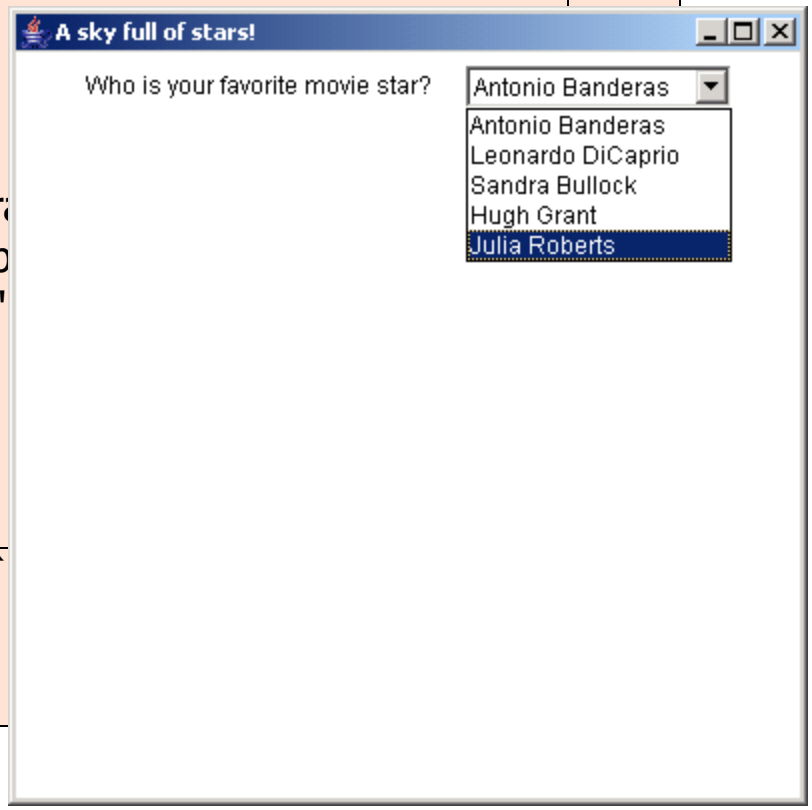


Списки

- Выводит список, предлагаемый пользователю для выбора.
- Пользователь может выбрать один или несколько пунктов.
- Создаётся с использованием набора строковых или текстовых значений.
- Класс Choice позволяет создавать списки с многими элементами. Синтаксис создания:
  - `Choice moviestars=new Choice();`
- Элементы добавляются с помощью метода `addItem()`, как показано ниже:
  - `moviestars.addItem("Antonio Banderas");`
  - `moviestars.addItem("Leonardo Dicaprio");`

Пример

```
import java.awt.*;
import java.awt.event.*;
class Stars extends Frame
{
   Choice moviestars = new Choice();
   Label lblQts = new Label("Who is your favorite movie star?");
   public Stars(String str)
   {
      super(str);
      setLayout(new FlowLayout());
      moviestars.addItem("Antonio Bandera
      moviestars.addItem("Leonardo DiCap
      moviestars.addItem("Sandra Bullock"
      moviestars.addItem("Hugh Grant");
      moviestars.addItem("Julia Roberts");
      add(lblQts);
      add(moviestars);
                ObjStar.setSize(400,400);
                   }
        }
```



Менеджер макетов – Layout Manager

- Экранные компоненты пользовательского интерфейса могут быть размещены различными способами.
- Каждый из этих способов может быть обозначен термином «**макет размещения компонентов**» (**layout of components**).
- Для управления этими макетами существуют менеджеры макетов (**layout managers**).
- Менеджеры макетов становятся необходимыми, когда размеры экрана (окна) должны быть изменены или когда любой элемент интерфейса должен быть перерисован.

Типы макетов

- Пакет AWT предоставляет группу классов, известных, как **менеджеры макетов** (**layout managers**), которые выполняют управление макетом
- Различные типы макетов включают:
  - `FlowLayout`
  - `BoxLayout`
  - `BorderLayout`
  - `CardLayout`
  - `GridLayout`
  - `GridBagLayout`
  - `SpringLayout`
  - *`null`*

Менеджер макетов

- Каждый менеджер макетов имеет собственное специализированное применение:
  - Для вывода нескольких компонентов одинакового размера в строках и столбцах лучше всего подходит **`GridLayout`**.
  - Для вывода компонента в максимально возможном пространстве следует сделать выбор из следующих двух менеджеров макетов **`BorderLayout`** и **`GridBagLayout`**.

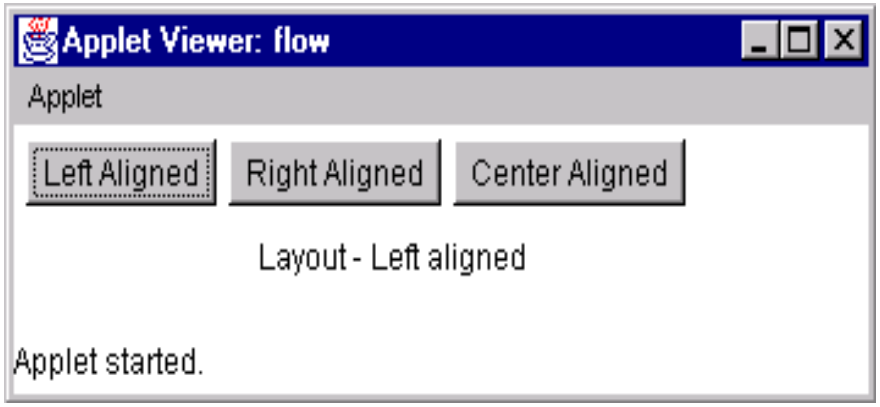
Как установить макеты?

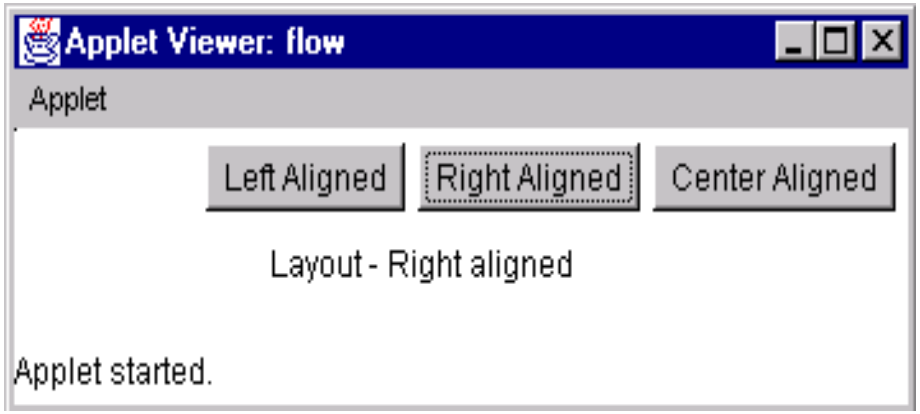
- При первоначальном создании компонента используется менеджер макетов, принятый по умолчанию.
  - Для апплетов по умолчанию принят макет `FlowLayout`
- Все компоненты размещаются в контейнере и располагаются в соответствии с принятым для него менеджером макетов.
- Новый менеджер макета может быть установлен с помощью метода `setLayout()`.

Менеджер FlowLayout

- Макет по умолчанию для апплетов и панелей.
- Компоненты располагаются последовательно, начиная с верхнего левого угла, и до нижнего правого угла.
- Конструкторы для **FlowLayout**:
  - `FlowLayout mylayout = new FlowLayout();`
  - `FlowLayout exLayout = new FlowLayout(FlowLayout.RIGHT);`
    // задан способ выравнивания

Менеджер FlowLayout

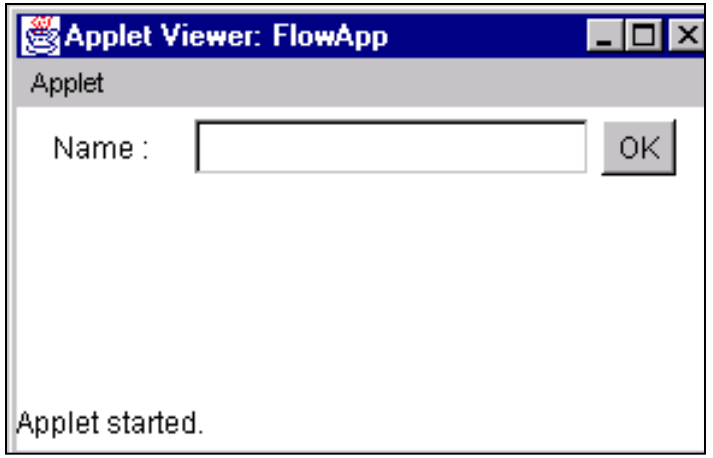




FlowLayout – выравнивание по левому и по правому краям

Пример

```
/*
<applet code=FlowApp width=500 height=500>
</applet>
*/
import java.awt.*;
import java.applet.*;
public class FlowApp extends Applet
{
    public void init()
    {
        TextField txtName = new TextField(20);
        Label lblName = new Label("Name:");
        Button ok = new Button("OK");
        add(lblName);
        add(txtName);
        add(ok);
    }
}
```



Менеджер BorderLayout

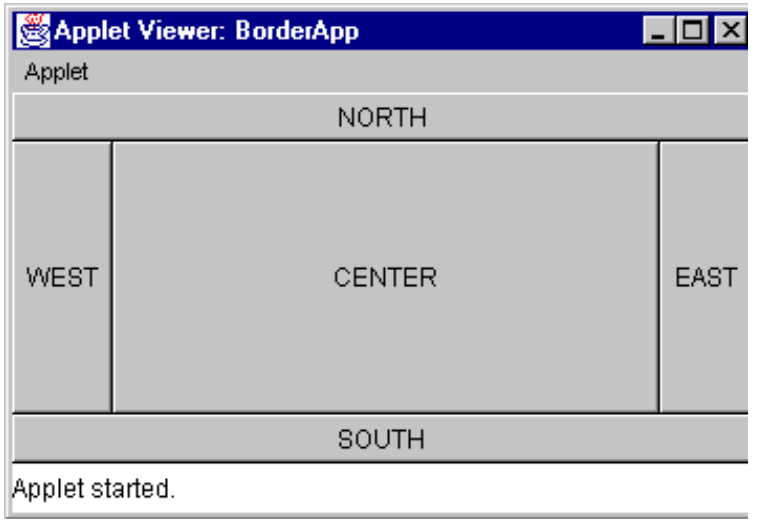
- Менеджер макета по умолчанию для объектов типов `Window`, `Frame` и `Dialog`
- При использовании этого макета компонентам назначаются позиции:
  - North («север»),
  - South («юг»),
  - East («восток»),
  - West («запад»)
  - Center («центр») внутри контейнера.

Менеджер BorderLayout

- Значения констант, которые позволяют позиционировать компоненты в макете `BorderLayout`:
  - PAGE_START: соответствует верхней части контейнера
  - LINE_END: соответствует правой части контейнера
  - PAGE_END: соответствует нижней части контейнера
  - LINE_START: соответствует левой части контейнера
  - LINE_CENTER: соответствует центральной части контейнера

Пример

```
/*<applet code = BorderApp width =500 height = 500>
</applet>
*/
import java.awt.*;
import java.applet.*;
public class BorderApp extends Applet
{
    public void init()
    {
        setLayout(new BorderLayout());
        Button best = new Button("EAST");
        Button bwest = new Button("WEST");
        Button bnorth = new Button("NORTH");
        Button bsouth = new Button("SOUTH");
        Button bcentre = new Button("CENTER");
        add(best, BorderLayout.LINE_END);
        add(bwest, BorderLayout.LINE_START);
        add(bnorth, BorderLayout.PAGE_START);
        add(bsouth, BorderLayout.PAGE_END);
        add(bcentre, BorderLayout.CENTER);
    }
}
```

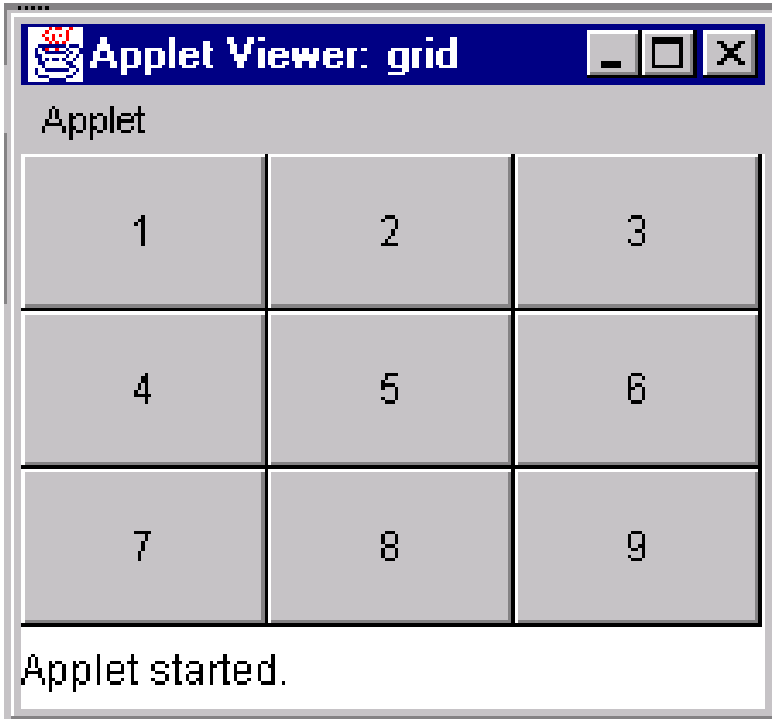


Менеджер GridLayout

- Помогает разметить область контейнера по прямоугольной сетке.
- Компоненты размещаются в строках и столбцах.
- Используется, когда все компоненты имеют один и тот же размер.
- Один из конструкторов `GridLayout` приведён ниже:
  - `GridLayout` g1= `new GridLayout(4,3);`

    (4 представляет количество строк, а 3 – количество столбцов)

Внешний вид GridLayout



Менеджер GridBagLayout

- Одинаковый размер компонентов не обязателен.
- Компоненты размещаются в строках и столбцах.
- Порядок размещения компонентов не определяется направлением сверху-вниз или слева-направо.
- Для контейнера менеджер `GridBagLayout` может быть установлен с использованием следующего синтаксиса:

```
GridBagLayout gb = new GridBagLayout();
ContainerName.setLayout(gb);
```

Менеджер GridBagLayout

- Для использования этого макета должна быть предоставлена информация о размере и расположении каждого компонента.
- Класс `GridBagConstraints` содержит всю информацию о положении и размере каждого компонента, требуемую классом `GridBagLayout`.
- Класс `GridBagConstraints` можно считать вспомогательным классом для `GridBagLayout`.

Менеджер GridBagLayout

- Список переменных-членов класса `GridBagConstraints`:
  - `weightx, weighty`: определяют распределение пространства
  - `gridwidth, gridheight`: определяют количество ячеек по ширине или по высоте в выводимой области компонента.
  - `ipadx, ipady`: определяют пространство, необходимое для изменения минимальной высоты и ширины компонента.

Менеджер GridBagLayout

- Список переменных-членов класса `GridBagConstraints`:
  - `Anchor`: определяет положение компонентов.
  - `gridx, gridy`: определяют ячейку для хранения компонента.
  - `Fill`: определяет, как компонент заполняет ячейку, если ячейка больше компонента
  - `Insets`: определяет интервал между компонентами сверху, снизу, слева и справа.

Пример

```
/*<applet code= MyGridBag width = 500 height = 500>
</applet>
*/
import java.awt.*;
import java.applet.Applet;
public class MyGridBag ex
{
    TextArea ObjTa;
    TextField ObjTf;
    Button butta, buttf;
    CheckboxGroup cbg;
    Checkbox cbbold,cbitali
    GridBagLayout gb;
    GridBagConstraints gbc
    public void init()
    {
        gb = new GridBagLa
        setLayout(gb);
        gbc = new GridBagC
        ObjTa = new TextAr
        ObjTf = new TextFie
        butta = new Button(
        buttf = new Button(
```

```
        cbg = new CheckboxGroup();
        cbbold = new Checkbox("Bold",cbg,false);
        cbitalic = new Checkbox("Italic",cbg,false);
        cbplain = new Checkbox("Plain",cbg,false);
        cbboth = new Checkbox("Bold/Italic",cbg,true);
        gbc.fill = GridBagConstraints.BOTH;
        addComponent(ObjTa,0,0,4,1);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(butta,0,1,1,1);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(buttf,0,2,1,1);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(cbbold,2,1,1,1);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(cbitalic,2,2,1,1);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(cbplain,3,1,1,1);
```

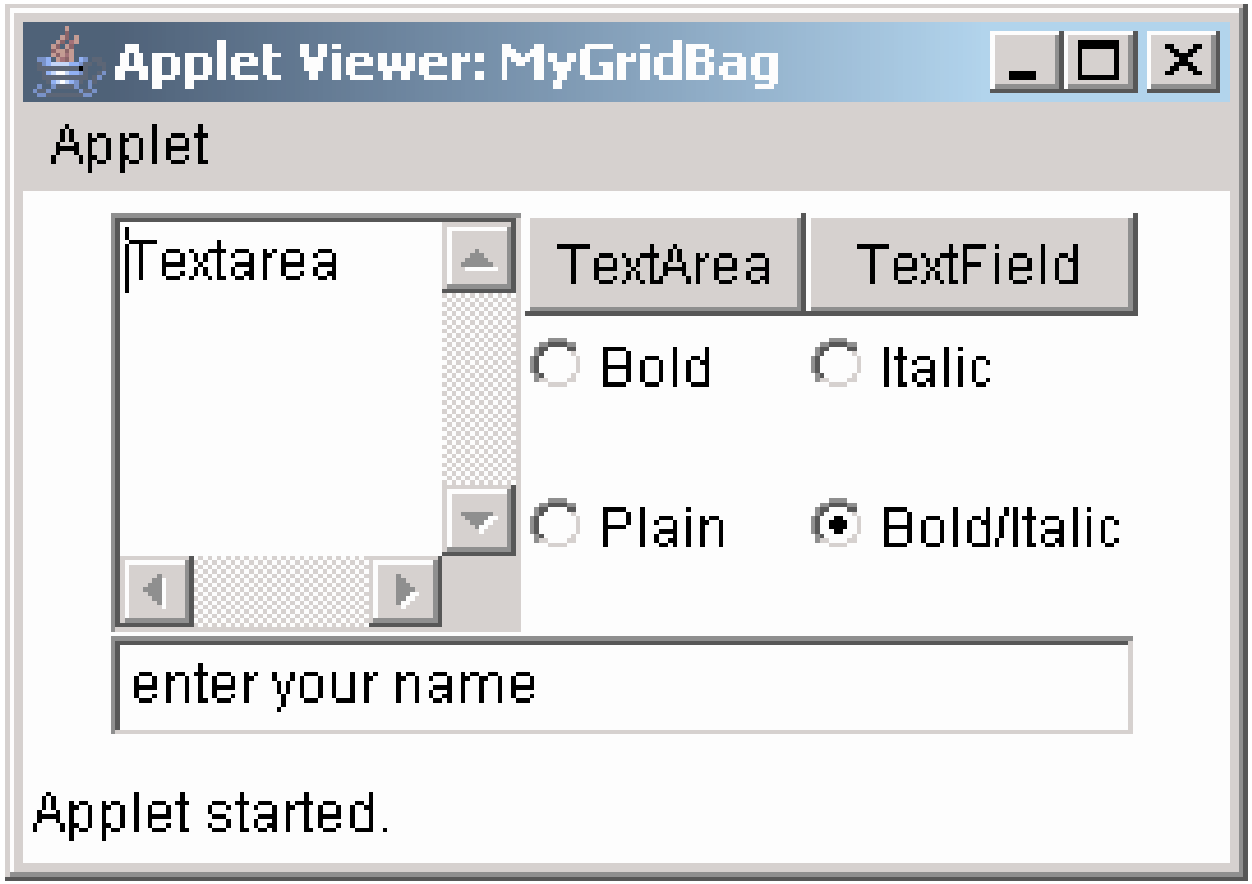
Пример (продолжение)

```
        gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(cbboth,3,2,1,1);
        gbc.fill = GridBagConstraints.HORIZONTAL;
        addComponent(ObjTf,4,0,1,3);
    }
    public void addComponent(Component comp, int row, int col, int    nrow, int ncol)
    {
        gbc.gridx = col;
        gbc.gridy = row;

        gbc.gridwidth = ncol;
        gbc.gridheight = nrow;

        gb.setConstraints(comp,gbc);
        add(comp);
    }
}
```

Пример (продолжение)



Менеджер CardLayout

- Может хранить набор («стек» - stack) из нескольких макетов.
- Каждый макет подобен карте в колоде карт.
- Эти карты обычно хранятся в объекте `Panel`.
- Используется, когда необходимо некоторое количество панелей со своими отдельными макетами, которые выводятся друг за другом.
- Все эти панели содержит главная панель.

Пример (Вывод)