Использование элементов управления, менеджеров компоновки и меню AWT

В этой главе продолжается исследование AWT. В ней рассматриваются стандартные элементы управления и менеджеры компоновки, а также элементы меню. В главу включено обсуждение двух высокоуровневых компонентов — диалогового окна и файлового диалогового окна. Завершает главу другой взгляд на обработку событий.

Элементы управления (controls) — это компоненты, которые предоставляют пользователю различные способы взаимодействия с приложением (например, командная *кнопка* (push button)). *Менеджер компоновки* (layout manager) автоматически позиционирует (размещает, располагает) компоненты в контейнере. Вид окна, таким образом, определяется комбинацией элементов управления, содержащихся в окне, и менеджера компоновки, используемого для их размещения.

В дополнение к элементам управления, фрейм-окно может также включать *строку меню* (menu bar) стандартного стиля. Каждый вход в строке меню активизирует раскрывающееся меню элементов, которые пользователь может выбирать. Строка меню всегда позиционируется наверху окна. Строки меню, хотя и различаются по виду, обрабатываются похожим способом, что и другие элементы управления.

Хотя компоненты окна можно позиционировать вручную, это весьма утомительно. Менеджер компоновки предназначен для автоматизации этой задачи. В первой части данной главы, которая представляет различные элементы управления, будет использован менеджер компоновки, заданный по умолчанию. Он отображает компоненты в контейнере, размещая их по принципу "слева-направо, сверху-вниз". Затем будут рассмотрены все менеджеры компоновки. Там будет видно, как лучше управлять позиционированием элементов управления.

Элементы управления. Основные понятия

AWT поддерживает следующие типы элементов управления:

- □ Текстовые метки (Labels)
- □ Кнопки (Push buttons)
- □ Флажки (Check boxes)
- □ Списки с выбором элементов (Choice lists)
- □ Списки (Lists)
- □ Полосы прокрутки (Scroll bars)

□ Элементы редактирования текста: текстовые поля (Text fields) и текстовые области (Text areas).

Элементы управления представлены специальными классами пакета AWT, которые являются подклассами класса Component.

Добавление и удаление элементов управления

Для включения элемента управления в окно нужно добавить его к окну. Для этого необходимо сначала создать экземпляр желательного элемента управления и затем добавить его к окну вызовом метода add(), который определен в классе Container. Метод add() имеет несколько форм. В первой части этой главы используется следующая форма:

```
Component add (Component compObj)
```

Здесь compObj — экземпляр элемента управления, который вы хотите добавить. Метод возвращает ссылку на объект, который передается параметром compObj. Сразу после добавления элемент управления будет автоматически выводиться на экран всякий раз, когда отображается его родительское окно.

Если вы захотите удалить элемент управления из окна, когда он больше не нужен, вызывайте метод remove(), который определен в классе Container. Его общая форма:

```
void remove(Component obj)
```

Здесь obj — ссылка на элемент управления, который нужно удалить. Вызывая метод removeAll(), можно удалить все элементы управления.

Реагирование на элементы управления

За исключением меток, которые являются пассивными, все элементы управления генерируют события, когда к ним обращается пользователь. Например, когда пользователь нажимает на кнопку, программе посылается сообщение о событии, которое идентифицирует кнопку. В общем случае, программа просто реализует соответствующий интерфейс и затем регистрирует блок прослушивания события для каждого элемента, которым нужно управлять. Как только блок прослушивания установлен, события посылаются к нему автоматически. В следующих разделах определен соответствующий интерфейс для каждого элемента управления.

Текстовые метки

Самый простой для использования элемент управления — (текстовая) метка (label). *Текстовая метка* — это объект класса Label, содержащий строку, которую она отображает. Метки — пассивные

элементы управления, которые не поддерживают никакого взаимодействия с пользователем. Класс Label определяет следующие конструкторы:

```
Label()
Label(String str)
Label (String str, int how)
```

Первая версия создает пустую метку, вторая — метку, которая содержит строку, специфицированную параметром str. Эта строка выровнена по левому краю. Третья версия создает метку, которая содержит строку, специфицированную параметром str, используя выравнивание, указанное в параметре how. Значением how должна быть одна из трех констант: Label.LEFT, Label.RIGHT или Label.CENTER.

Текст в метке можно устанавливать или изменять, используя метод setText(), а текущую метку можно получить, вызывая getText(). Формат этих методов:

```
void setText (String str)
String getText()
```

Параметр str в setText() определяет новую метку. Метод getText() возвращает текущую метку.

Вызывая метод setAlignment(), можно устанавливать выравнивание строки в пределах метки. Чтобы получить текущее выравнивание, вызовите метод getAlignment(). Формат этих методов:

```
void setAlignment (int how)
int getAlignment()
```

Параметр how должен быть одной из констант выравнивания, предствленных ранее.

Следующий пример создает три метки и добавляет их к апплету.

Программа 114. Метки

```
// Файл LabelDemo.java
// Демонстрирует метки.
import java.awt.*;
import java.applet.*;
/*
<applet code = "LabelDemo" width = 300 height = 200>
</applet>
*/
public class LabelDemo extends Applet {
   public void init() {
//  Создание меток
      Label one = new Label("One");
      Label two = new Label("Two");
      Label three = new Label("Three");
// Добавить метки в окно апплета
```

```
        add(one);
        add(two);
        add(three);
    }
}
```

На рис.1 показано окно, созданное апплетом LabelDemo. Обратите внимание, что метки организованы в окне менеджером компоновки, заданным по умолчанию. Позже вы увидите, как более точно управлять размещением меток.

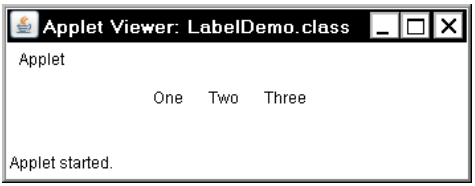

Рис. 1. Метки

Использование кнопок

Наиболее широко используемым элементом управления является кнопка (push button). *Кнопка* — это компонент, который содержит текстовую метку и генерирует событие, когда ее нажимают. Кнопки являются объектами класса Button. В классе Button определяются два конструктора:

```
Button()     // Кнопка без надписи
Button(String str)    // кнопка с надписью str
```

Первая версия создает пустую кнопку, вторая — кнопку с текстовой меткой, которая передается через параметр str.

После того как кнопка была создана, можно установить ее метку, вызывая метод setLabel(). Извлечь ее метку можно вызовом getLabel(). Форматы этих методов:

```
void setLabel (String str)
String getLabel()
```

Параметр str указывает новую метку для кнопки.

Обработка кнопок

Каждый раз, когда кнопка нажимается, генерируется action-событие. Оно посылается любым блокам прослушивания, которые предварительно зарегистрировали заинтересованность в приеме уведомления об action-событии от этого компонента. Каждый блок прослушивания реализует интерфейс ActionListener. Этот интерфейс

определяет метод actionPerformed(), который вызывается при возникновении события. В качестве аргумента в этот метод передается объект ActionEvent. Он содержит как ссылку на кнопку, которая сгенерировала событие, так и ссылку на строку, которая является меткой кнопки. Для идентификации кнопки можно использовать любую из этих ссылок.

Ниже показан пример, который создает три кнопки с метками "Yes", "No" и "Undecided". Каждый раз, когда одна из них нажимается, отображается сообщение, которое докладывает, что кнопка была нажата. В этой версии метка кнопки используется для того, чтобы определить, какая из них была нажата. Метка возвращается вызовом метода getActionCommand() объекта ActionEvent, передаваемого методу actionPerformed().

Программа 115. Кнопки

```
// Файл ButtonDemo.java
// Демонстрирует кнопки.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = "ButtonDemo" width = 250 height = 150>
</applet>
*/
public class ButtonDemo extends Applet implements ActionListener {
   String msg = "";          // Строка с сообщением
   Button yes, no, maybe;    // Ссылки на кнопки
   public void init() {
      yes = new Button("Yes");     // Создание
      no = new Button ("No");      // кнопок
      maybe = new Button("Undecided");
      add(yes);                    // Добавление
      add(no);                     // кнопок
      add(maybe);                  // в окно
      yes.addActionListener(this);    // Регистрация окна апплета (this)
      no.addActionListener(this);     // у кнопок
      maybe.addActionListener(this);
   }
// Обработчик события нажатия кнопок
   public void actionPerformed(ActionEvent ae) {
      String str = ae.getActionCommand();   // Получаем надпись на кнопке
      if(str.equals("Yes"))
         msg = "You pressed Yes.";
      else if(str.equals("No"))
         msg = "You pressed No.";
      else
         msg = "You pressed Undecided.";
      repaint ();                          // Вызов paint()
   }
   public void paint(Graphics g) {
```

```
      g.drawString(msg, 6, 100);
   }
}
```

Пример вывода программы ButtonDemo пока зан на рис. 2.

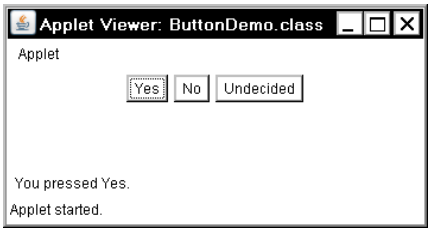

Рис. 2. Кнопки

В дополнение к сравнению текстовых меток можно также определять, какая кнопка была нажата, сравнивая объект, полученный от метода getSource(), с объектами кнопок, которые добавлены к окну. Чтобы это сделать, нужно сохранять список объектов во время их добавления. Данный подход показывает следующий апплет:

Программа 116. Распознавание нажатых кнопок

```
// Файл ButtonList.java
// Распознавание объектов типа Button.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = "ButtonList" width = 250 height = 150>
</applet>
*/
public class ButtonList extends Applet implements ActionListener {
   String msg = "";
   Button bList[] = new Button[3];    // Массив ссылок на кнопки
   public void init() {
      Button yes = new Button("Yes");
      Button no = new Button("No");
      Button maybe = new Button("Undecided") ;
   // Сохранить ссылки на кнопки при их добавлении
      bList[0] = (Button) add(yes);      // Преобразование
      bList[1] = (Button) add(no);       // результата add()
      bList[2] = (Button) add(maybe);    // к типу button
   // Регистрироваться для приема событий действия
      for(int i = 0; i < 3; i++)
         bList[i].addActionListener(this);
   }
   public void actionPerformed(ActionEvent ae) {
      for(int i = 0; i < 3; i++)
         if (ae.getSource() == bList [i])
            msg = "You pressed " + bList[i].getLabel();
```

```
      repaint();
   }
   public void paint(Graphics g) {
      g.drawString(msg, 6, 100);
   }
}
```

В этой версии программа сохраняет каждую ссылку кнопки в массиве bList[], когда кнопки добавляются к окну апплета. (Напомним, что метод add() возвращает ссылку на добавляемую кнопку.) Затем этот массив используется внутри метода actionPerformed(), чтобы определить, какая кнопка была нажата.

Для простых апплетов обычно проще распознать кнопки по их меткам. Однако в ситуациях, когда требуется изменять метку кнопки во время выполнения программы, или при использовании кнопок, которые имеют одинаковые метки, может оказаться проще определить нажатую кнопку, используя ее ссылку на объект.

Применение флажков

Флажок (check box) — это элемент управления, который используется для включения или выключения некоторой опции (режима, параметра и т. п.). Он имеет вид маленького квадратика, который может содержать (или не содержать) маркер проверки (check mark). С каждым флажком связывается текстовая метка, которая описывает, какую опцию представляет флажок. Вы можете изменять состояние флажка щелчком мыши по нему. Флажки можно использовать индивидуально или как часть группы. Флажки являются объектами типа Checkbox, который поддерживает следующие конструкторы:

```
Checkbox()
Checkbox(String str)
Checkbox(String str, boolean on)
Checkbox (String str, boolean on, CheckboxGroup cbGroup)
Checkbox (String str, CheckboxGroup cbGroup, boolean on)
```

Первая форма создает флажок, чья текстовая метка изначально — пробел. Состояние флажка в этом случае — "выключен" (off, unchecked). Вторая форма создает флажок, чья текстовая метка определяется параметром str. Состояние флажка здесь также "off". Третья форма позволяет устанавливать начальное состояние флажка. Если параметр on равен true, создается флажок с первоначальным состоянием "включен" ("on"); иначе (когда on равен false) создается "чистый" флажок — без маркера проверки (т. е. в состоянии "off"). Четвертые и пятые формы создают флажок, чья текстовая метка определена параметром str, а группа указана параметром cbGroup. Если данный флажок не является частью группы, то cbGroup должен иметь

значение null (пустая ссылка). (Группы флажков описаны в следующем разделе.) Значение параметра on определяет начальное состояние флажка.

Для получения текущего состояния флажка вызовите метод getState(). Чтобы установить его состояние, вызовите setState(). Можно получить текущую строковую метку, связанную с флажком, вызывая getLabel(). Чтобы установить эту метку, вызовите setLabel(). Форматы эти методов следующие:

```
boolean getState()
void setState (boolean on)
String getLabel()
void setLabel (String str)
```

Если on есть true, флажок устанавливается в состояние "on" (включен). Если on — false, флажок сбрасывается (квадратик очищается от маркера проверки, т. е. флажок устанавливается в состояние "off" (выключен)). Строка, переданная в параметре str, становится новой меткой, связанной с флажком.

Обработка флажков

Каждый раз, когда флажок выбирается (или выбор отменяется), генерируется item-событие. Оно посылается к любым блокам прослушивания, которые предварительно зарегистрировали свою заинтересованность в приеме уведомлений об item-событиях от этого компонента. Каждый блок прослушивания реализует интерфейс ItemListener. Этот интерфейс определяет метод itemStateChanged(), которому через параметр передается объект itemEvent. Он содержит информацию относительно данного события (например, был ли выбор произведен или отменен).

Следующая программа создает четыре флажка. Начальное состояние первого — "on". На экране отображается информация о состоянии каждого флажка и модифицируется всякий раз, когда изменяется состояние какого-либо флажка (рис. 22.3).

Программа 117. Флажки

```
// Файл CheckboxDemo.java
// Демонстрирует флажки.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = "CheckboxDemo" width = 250 height = 200>
</applet>
*/
public class CheckboxDemo extends Applet implements ItemListener {
```

```
    String msg = "";
    Checkbox Win98, winNT, Solaris, mac;
    public void init() {
       Win98 = new Checkbox("Windows 98", null, true);
       winNT = new Checkbox("Windows NT");
       Solaris = new Checkbox("Solaris");
       mac = new Checkbox("MacOS");
       add(Win98);
       add(winNT);
       add(Solaris);
       add(mac);
       Win98.addItemListener(this);
       winNT.addItemListener(this);
       Solaris.addItemListener(this);
       mac.addItemListener(this);
    }
// Обработчик события изменения состояния флажка
    public void itemStateChanged(ItemEvent le) {
       repaint();
    }
// Отобразить текущее состояние флажков
    public void paint(Graphics g) {
       msg = "Current state: ";
       g.drawString(msg, 6, 80);
       msg = " Windows 98: " + Win98.getState();
       g.drawString(msg, 6, 100);
       msg = " Windows NT: " + winNT.getState();
       g.drawString(msg, 6, 120);
       msg = " Solaris: " + Solaris.getState();
       g.drawString(msg, 6, 140);
       msg = " MacOS: " + mac.getState();
       g.drawString(msg, 6, 160);
    }
}
```

Пример вывода программы показан на рис. 3

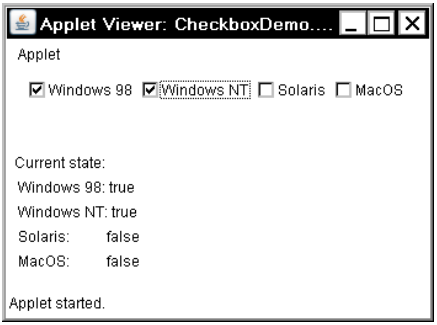

Рис. 3. Флажки

Класс CheckboxGroup

Возможно создание набора (группы) взаимоисключающих флажков, в котором может быть включен один и только один из них. Такие флажки часто называются "радиокнопками" (radio buttons), потому что они действуют подобно селектору станций на радиоприемнике — в любой момент может быть выбрана только одна станция. Чтобы создать набор взаимоисключающих флажков, нужно сначала определить группу, к которой они будут принадлежать, и затем указать эту группу, когда флажки будут создаваться. Группы флажков являются объектами типа CheckboxGroup. Пустую группу создает только конструктор, заданный по умолчанию.

Определить, какой флажок в группе выбран в текущий момент позволяет метод getSelectedCheckbox(). Установить флажок можно вызвав метода setSelectedCheckbox(). Форматы этих методов:

```
Checkbox getSelectedCheckbox()
void setSelectedCheckbox (Checkbox which)
```

где which — параметр, указывающий флажок, который нужно выбрать. При этом предварительно выбранный флажок будет выключен.

Ниже показана программа, которая использует флажки, являющиеся частью группы.

Программа 118. Радиокнопки

```
// Файл CBGroup.java
// Демонстрирует группу (взаимонезависимых) флажков.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = "CBGroupм width = 250 height = 200>
</applet>
*/
public class CBGroup extends Applet implements ItemListener {
   String msg = "";
   Checkbox Win98, winNT, Solaris, mac;
   CheckboxGroup cbg;
   public void init() {
      cbg = new CheckboxGroup();
      Win98 = new Checkbox("Windows 98", cbg, true);
      winNT = new Checkbox("Windows NT", cbg, false);
      Solaris = new Checkbox("Solaris", cbg, false);
      mac = new Checkbox ("MacOS", cbg, false);
      add(Win98);
      add(winNT);
      add(Solaris);
      add(mac);
      Win98.addItemListener(this);
```

```
      winNT.addItemListener(this);
      Solaris.addItemListener(this);
      mac.addItemListener(this);
   }
   public void itemStateChanged(ItemEvent ie) {
      repaint();
   }
// Отобразить текущее состояние группы
   public void paint(Graphics g) {
      msg = "Current selection:  ";
      msg += cbg.getSelectedCheckbox().getLabel();
      g.drawString(msg, 6, 100);
   }
}
```

Вывод, сгенерированный апплетом cBGroup, показан на рис.4. Обратите внимание, что флажки теперь имеют круглую форму.

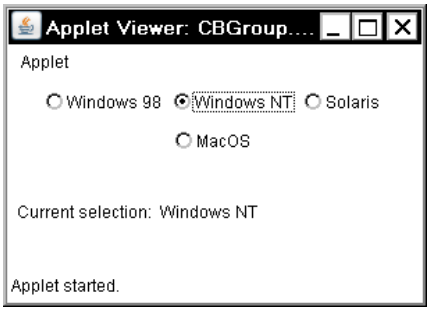

Рис. 4. Радиокнопки

Элемент управления Choice

Класс Choice используется для того, чтобы создавать раскрывающийся список элементов, из которых пользователь может делать выбор. Таким образом, элемент управления "выбор" (Choice) имеет форму меню: В неактивном состоянии компонент типа Choice занимает столько места, чтобы показывать только текущий выбранный элемент. Когда пользователь щелкает по нему мышью, раскрывается полный список элементов и можно сделать новый выбор. Каждый элемент в списке — это строка, которая выровнена по левому краю и появляется в списке в том порядке, в каком она добавлялась к объекту типа Choice. Класс Choice определяет только умолчиваемый конструктор, который создает пустой список. Чтобы добавить элемент выбора к списку, вызовите метод addItem() или add(). Сигнатуры этих методов:

```
void addItem (String name)
void add (String name)
```

Здесь name — имя добавляемого элемента. Элементы добавляются к списку в том порядке, в котором выполнялись вызовы add() или addItem().

Для определения выбранного в настоящее время элемента можно вызвать метод getSelectedItem() или getSelectedIndex() с форматами:

```
String getSelectedItem()
int getSelectedIndex()
```

Метод getSelectedItem() возвращает строку, содержащую имя элемента, а метод getSelectedIndex() — индекс (номер) элемента. Первый элемент имеет индекс 0. По умолчанию выбирается первый элемент, добавленный к списку.

Чтобы получить количество элементов в списке, вызовите метод getItemCount(). Выбранный элемент можно установить текущим, вызывая метод select() с аргументом в виде отсчитываемого от нуля целочисленного индекса или строки, которая совпадает с одним из имен в списке. Форматы соответствующих методов:

```
int getItemCount()
void select (int index)
void select(String name)
```

Зная индекс, можно получить имя элемента с этим индексом. Для этого нужно вызвать метод getItem(), который имеет следующую форму:

```
String getItem (int index)
```

где index специфицирует индекс желательного элемента.

Обработка списков типа Choice

Каждый раз, когда выбирается элемент Choice-списка, генерируется событие типа item. Оно посылается к любым блокам прослушивания, которые предварительно зарегистрировали заинтересованность в приеме уведомлений об item-событиях от данного компонента. Каждый блок прослушивания реализует интерфейс itemListener. Этот интерфейс определяет метод itemStateChanged(). Объект ItemEvent передается этому методу в качестве аргумента.

В следующем примере создаются два choice-меню. Одно выбирает операционную систему, другое — браузер.

Программа 119. Списки Choice

```
// Файл ChoiceDemo.java
// Демонстрирует Choice-списки.
import java.awt.*;
import java.awt.event.*;
```

```
import java.applet.*;
/*
<applet code = "ChoiceDemo" width = 300 height = 180>
</applet>
*/
public class ChoiceDemo extends Applet implements ItemListener {
   Choice os, browser;
   String msg = "";
   public void init () {
      os = new Choice ();      // Список ОС
      browser = new Choice();  // Список браузеров
// Добавить элементы в список OS
      os.add("Windows 98");
      os.add("Windows NT");
      os.add("Solaris");
      os.add("MacOS") ;
// Добавить элементы в список браузеров
      browser.add("Netscape 1.1");
      browser.add("Netscape 2.x");
      browser.add("Netscape 3.x");
      browser.add("Netscape 4.x");
      browser.add("Internet Explorer 2.0");
      browser.add("Internet Explorer 3.0");
      browser.add("Internet Explorer 4.0");
      browser.add("Lynx 2.4");
      browser.select("Netscape 4.x");
// Добавить choice-списки в окно
      add(os);
      add(browser);
// Регистрироваться для приема item-событий
      os.addItemListener(this);
      browser.addItemListener(this);
   }
// Обработчик события ItemEvent
   public void itemStateChanged(ItemEvent ie) {
      repaint();
   }
// Отобразить текущие выборы
   public void paint(Graphics g) {
      msg = "Current OS:  ";
      msg += os.getSelectedItem();// Добавляем выбранный элемент списка
      g.drawString(msg, 6, 120);
      msg = "Current Browser: ";
      msg += browser.getSelectedItem();
      g.drawString (msg, 6, 140);
   }
}
```

Пример вывода программы ChoiceDemo показан на рис.5.

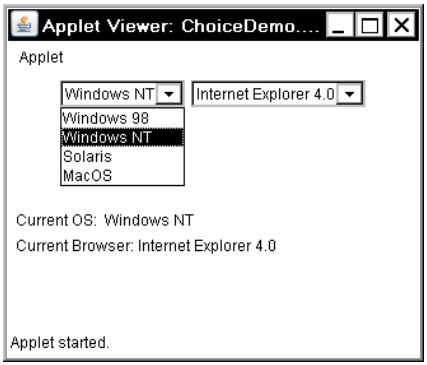

Рис. 5. Список выбора

Использование списков

Класс List обеспечивает компактный многоэлементный список со множественным выбором и прокруткой. В отличие от объекта типа Choice, который показывает в меню только один выбранный элемент, List-объект может быть сконструирован так, чтобы отображать любое число элементов выбора в видимом окне. Его можно создать так, чтобы разрешить множественный выбор. List содержит следующие конструкторы:

```
List ()
List(int numRows)
List(int numRows, boolean multipleSelect)
```

Первая форма создает элемент управления List, который позволяет выбирать только один элемент. Во второй форме значение параметра numRows определяет число строк в списке, которые будут всегда видимы в панели списка (другие могут прокручиваться в панели по мере необходимости). В третьей форме, если параметр multipleSelect равен true, то пользователь может выбирать два или несколько элементов одновременно. Если его значение — false, то можно выбрать только один элемент.

Чтобы добавить элемент выбора к списку, вызывайте метод add(), который имеет следующие формы:

```
void add (String name)
void add (String name, int index)
```

Здесь name — имя элемента, добавляемого к списку. Первая форма добавляет элементы к концу списка. Вторая — добавляет элементы по индексу (номеру), указываемому параметром index. Индексация

начинается с нуля. Чтобы добавить элемент в конец списка, нужно указать индекс, равный -1.

Для списков, которые допускают только одиночный выбор, можно определять, какой элемент выбран в текущий момент, если вызвать метод getSelectedItem() или getSelectedIndex(). Форматы этих методов:

```
String getSelectedItem()
int getSelectedIndex()
```

Метод getSelectedItem() возвращает строку, содержащую имя элемента. Если выбран больше чем один элемент или если никакого выбора еще не было сделано, возвращается null (пустой указатель). Метод getSelectedIndex() возвращает индекс элемента. Первый элемент имеет индекс 0. Если выбрано больше одного элемента, или если никакого выбора еще не было сделано, возвращается -1.

Чтобы определить текущие выбранные элементы для списков, которые допускают множественный выбор, нужно использовать метод getSelectedItems() или getSelectedIndexes() с форматами:

```
String[ ] getSelectedItems()
int[] getSelectedIndexes()
```

getSelectedItems() возвращает массив, содержащий имена текущих выбранных элементов. getSelectedIndexes() возвращает массив, содержащий индексы текущих выбранных элементов.

Для определения количества элементов в списке вызывайте метод getItemCount(). Можно устанавливать текущий выбранный элемент, используя метод select() с отсчитываемым от нуля целым индексом. Форматы этих методов:

```
int getItemCount()
void select(int index)
```

Зная индекс, можно получить имя, связанное с элементом с этим индексом, вызывая метод getItem(), который имеет следующую форму:

```
String getItem(int index)
```

Здесь index — указывает индекс (номер) желательного элемента.

Обработка списков

Чтобы обрабатывать list-события (list events), нужно реализовать интерфейс ActionListener. Каждый раз, когда выполняется двойной щелчок на элементе типа List, генерируется объект ActionEvent. Его метод getActionCommand() можно использовать для извлечения имени вновь выбранного элемента. Кроме того, всякий раз, когда выбирается или отменяется выбор элемента (одиночным щелчком мыши), генерируется объект itemEvent. Его метод getStateChange() можно

использовать, чтобы определить, что породило это событие — выбор или отмена выбора. Метод getItemSelectable() возвращает ссылку на объект, который породил это событие.

Ниже показан пример, который конвертирует Choice-элемент управления из предшествующего раздела в List-компоненты, один — с множественным выбором, другой — с одиночным.

Программа 120. Списки List

```
// Файл ListDemo.java
// Демонстрирует списки.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = "ListDemo" width = 300 height = 180>
</applet>
*/
public class ListDemo extends Applet implements ActionListener {
   List os, browser;
   String msg = "";
   public void init() {
      os = new List(4, true);
      browser = new List(4, false);
// Добавить- элементы в список OS
      os.add("Windows 98");
      os.add("Windows NT");
      os.add("Solaris");
      os.add("MacOS");
// Добавить элементы в список браузеров
      browser.add("Netscape 1.1");
      browser.add("Netscape 2.x");
      browser.add("Netscape 3.x");
      browser.add("Netscape 4.x");
      browser.add("Internet Explorer 2.0");
      browser.add("Internet Explorer 3.0");
      browser.add("Internet Explorer 4.0");
      browser.add("Lynx 2.4");
      browser.select(1);
// Добавить списки в окно
      add(os);
      add(browser);
// Регистрироваться для приема action-событий
      os.addActionListener(this);
      browser.addActionListener(this);
   }
// Обработчик события ActionEvent
   public void actionPerformed(ActionEvent ae) {
      repaint();
   }
// Отобразить текущие выборы
   public void paint(Graphics g) {
      int idx[];
```

```
        msg = "Current OS:  ";
        idx = os.getSelectedIndexes();
        for(int i = 0; i < idx.length; i++)
           msg += os.getItem(idx[i]) + "     ";
        g.drawString(msg, 6, 120);
        msg = "Current Browser:  ";
        msg += browser.getSelectedItem();
        g.drawString(msg, 6, 140);
    }
}
```

Пример вывода, сгенерированного апплетом ListDemo, показан на рис.6. Обратите внимание, что список браузеров имеет полосу прокрутки, так как все элементы не вставить в число строк, указанное при его создании. Из списка ОС сделан множественный выбор.

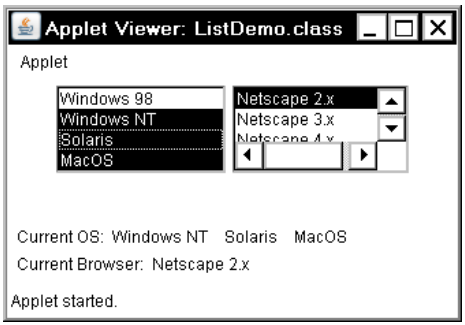

Рис. 6. Выбор из списков List

Управление полосами прокрутки

Полосы прокрутки (scroll bars) используются для выбора непрерывных значений из некоторого интервала с конечными границами. Полосы прокрутки могут быть ориентированы горизонтально или вертикально. Полоса прокрутки фактически является композицией нескольких индивидуальных частей. На каждом конце полосы имеется кнопка-стрелка, которую можно нажимать (щелчком мыши), чтобы переместить текущее значение полосы лрокрутки на одну позицию в направлении стрелки. Текущее значение полосы прокрутки относительно ее минимальных и максимальных значений обозначено *ползунком* (или *бегунком*) полосы прокрутки. Ползунок может перетаскиваться пользователем в новую позицию. Пользователь может шелкать мышью в фоновых частях полосы, находящихся с обеих сторон ползунка, чтобы заставить бегунок перескакивать в этом направлении с некоторым приращением, большим чем 1. Обычно это действие приводит к некоторой форме листания страницы вверх (page up) или

вниз (page down). Полосы прокрутки инкапсулированы в классе Scrollbar. В Scrollbar определены следующие конструкторы:

```
Scrollbar()
Scrollbar(int style)
Scrollbar(int style, int initialValue, int thumbSize, int min, int max)
```

Первая форма создает вертикальную полосу прокрутки. Вторая и третья — позволяют указать ориентацию полосы прокрутки. Если параметр style задается как Scrollbar.VERTICAL, то создается вертикальная полоса прокрутки, если — как Scrollbar.HORIZONTAL, то — горизонтальная. В третьей форме конструктора начальное значение полосы прокрутки передается в параметре initialValue, а высота ползунка — в thimbSize. Минимальное и максимальное значения для полосы прокрутки указываются в параметрах min и max.

Если вы создаете полосу прокрутки при помощи одного из первых двух конструкторов, то перед использованием нужно установить ее параметры, вызывая метод setvalues() следующего формата:

```
void setValues (int initialValue, int thumbSize, int min, int max)
```

Параметры имеют те же значения, как в только что описанном третьем конструкторе.

Чтобы получить текущее значение полосы прокрутки, вызовите метод getValue(). Он возвращает текущую установку. Чтобы установить текущее значение, вызовите setValue(). Форматы этих методов:

```
int getValue()
void setValue(int newValue)
```

Здесь newValue определяет новое значение для полосы прокрутки. Когда вы устанавливаете значение, ползунок внутри полосы прокрутки будет перемещен в позицию, отражающую новое значение.

Вы можете также отыскивать минимальное и максимальное значения через показанные ниже методы getMinimum() и getMaximum():

```
int getMinimum()
int getMaximum()
```

Для прокрутки вверх или вниз на одну строку по умолчанию используется (строчное) приращение, равное 1. Можно изменить это приращение, вызывая метод setUnitIncrement(). По умолчанию, страничные (page-up и page-down) приращения равны 10. Это значение можно изменять, вызывая setBlockIncrement(). Форматы этих методов:

```
void setUnitIncrement (int nevIncr)
void setBlockIncrement (int newIncr)
```

Обработка полос прокрутки

Для обработки событий полосы прокрутки следует реализовать интерфейс AdjustmentListener. Каждый раз, когда пользователь взаимодействует с полосой прокрутки, генерируется объект AdjustmentEvent. Чтобы определить тип настройки, можно использовать его метод getAdjustmentType(). Типы событий настройки следующие:

BLOCK_DECREMENT. Событие page-down было сгенерировано.

BLOCK_INCREMENT. Событие page-up был сгенерировано.

TRACK. Абсолютное tracking-событие был сгенерировано.

UNIT_DECREMENT. Кнопка "строка-вниз" (line-down) на полосе прокрутки была нажата.

UNIT_INCREMENT. Кнопка "строка-вверх" (line-up) на полосе прокрутки была нажата.

Следующий пример создает как вертикальную, так и горизонтальную полосы прокрутки. На экране отображаются их текущие установки. Если вы перетаскиваете мышью элементы, находящиеся внутри окна, то координаты каждого drag-события (события перетаскивания мыши) используются для обновления полосы прокрутки. В текущей drag-позиции указателя мыши внутри окна отображается звездочка (asterisk).

Программа 121. Полосы прокрутки

```
// Файл SBDemo.java
// Демонстрирует полосы прокрутки.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = "SBDemo" width = 300 height = 200>
</applet>
*/
public class SBDemo extends Applet
implements AdjustmentListener, MouseMotionListener { // При работе
// с полосами прокрутки возникают события типа Adjustment (выравнивания)
   String msg = "";
   Scrollbar vertSB, horzSB;
   public void init() {
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      vertSB = new Scrollbar(Scrollbar.VERTICAL, 0, 1, 5, height);
      horzSB = new Scrollbar(Scrollbar.HORIZONTAL, 0, 1, 5, width);
// 0 - минимальное значение ползунка
// 1 - текущее значение ползунка
// 5 - максим. значение
      add(vertSB);
      add(horzSB);
// Зарегистрироваться для приема adjustment-событий
```

```
        vertSB.addAdjustmentListener(this);
        horzSB.addAdjustmentListener(this);
        addMouseMotionListener(this);
    }
// Обработчики событий
    public void adjustmentValueChanged(AdjustmentEvent mes) {
        repaint();
    }
// Обновить полосы прокрутки, чтобы отразить перетаскивание мыши
    public void mouseDragged(MouseEvent me) {
        int x = me.getX();
        int y = me.getY();
        vertSB.setValue(y);
        horzSB.setValue(x);
        repaint();
    }
// необходим для MouseMotionListener
    public void mouseMoved(MouseEvent me)  { }
// Отобразить текущее значение полос прокрутки
    public void paint(Graphics g) {
        msg = "Vertical:  " + vertSB.getValue();
        msg += ", Horizontal: " + horzSB.getValue();
        g.drawString(msg, 6, 160);
// Показать текущую drag-позицию мыши
        g.drawString("*", horzSB.getValue(), vertSB.getValue());
    }
}
```

Окно апплета с полосами прокрутки приведено на рис.7.

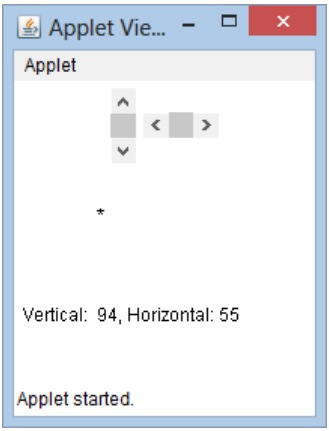

Рис. 7. Полосы прокрутки

Использование класса TextField

Класс TextField реализует однострочную область ввода текста, обычно называемую элементом редактирования (edit control). Текстовые

поля дают возможность пользователю вводить строки и редактировать текст, используя клавиши-стрелки, сочетание клавиш для операций "вырезать" и "вставить", а также выборки мышью. TextField — подкласс TextComponent. TextField определяет следующие конструкторы:

```
TextField ()
TextField(int numChars)
TextField(String str)
TextField(String str, int numChars)
```

Первая форма создает заданное текстовое поле по умолчанию. Вторая — создает текстовое поле шириной numChars символов. Третья форма инициализирует текстовое поле со строкой, содержащейся в str. Четвертая — инициализирует текстовое поле и устанавливает его ширину.

TextField (и его суперкласс TextComponent) обеспечивает несколько методов, которые позволяют использовать текстовое поле. Чтобы получить строку, содержащуюся в текущий момент в текстовом поле, вызовите метод getText(), а для установки текста вызовите setText(). Форматы этих методов следующие:

```
String getText()
void setText(String str)
```

Здесь str — новая строка.

Пользователь может выбирать часть текста в текстовом поле. Метод select() позволяет выбирать часть текста под программным управлением. Вызывая getseLectedText(), ваша программа может получить текущий выбранный текст. Формат этих методов:

```
String getSelectedText()
void select (int startIndex, int endIndex)
```

Метод getSelectedText() возвращает выбранный текст, а метод select() выбирает символы, начинающиеся в startIndex и заканчивающиеся в endIndex - 1.

Вызовом setEditable() можно управлять возможностью редактирования (изменения содержания) текстового поля пользователем. Вызовом isEditable() можно определить, редактируемо ли данное поле. Форматы этих методов:

```
boolean isEditable()
void setEditable (boolean canEdit)
```

isEditable() возвращает true, если текст может быть изменен, и false — в противном случае. В методе setEditable(), если canEdit true, то текст может быть изменен, а если false — не может.

Если нужно, чтобы пользователь мог вводить текст, который бы не отображался в секретном поле (типа пароля), то следует отключить

отображение на экране вводимых символов, вызывая setEchoChar(). Данный метод определяет одиночный символ (эхо-символ), который будет отображаться при вводе каждого символа (таким образом, фактически вводимые символы не будут показаны в поле). С помощью метода echoCharIsSet() можно проверить, находится ли текстовое поле в этом режиме. Вызывая метод getEchoChar() можно отыскать и извлечь эхо-символ. Форматы перечисленных методов следующие:

```
void setEchoChar(char ch)
boolean echoCharIsSet()
char getEchoChar()
```

Здесь ch определяет эхо-символ, который будет отображаться на экране.

Обработка TextField

Так как текстовые поля выполняют свои собственные функции редактирования, программа вообще не будет откликаться на индивидуальные key-события, которые происходят в текстовом поле. Однако можно обработать нажатие клавишы <Enter>. Когда это происходит, генерируется action-событие (типа "действие").

Пример, который создает классическое окно с именем и паролем пользователя:

Программа 122. Текстовые поля

```
// Файл TextFieldDemo.java
// Демонстрирует текстовое поле.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code="TextFieldDemo" width=380 height=150> </applet>
*/
public class TextFieldDemo extends Applet implements ActionListener
// При работе с текстовыми полями возникают Action - события
{
   TextField name, pass;
   public void init() {
      Label namep = new Label("Name: ", Label.RIGHT);
      Label passp = new Label("Password: ", Label.RIGHT);
      name = new TextField(12);
      pass = new TextField(8);
      pass.setEchoChar('?');   // Назначение эхо-символа
      add(namep);
      add(name);
      add(passp);
      add(pass);
// Регистрироваться для получения action-событий
```

```
        name.addActionListener(this);
        pass.addActionListener(this);
    }
// Клавиша <Enter>, нажатая пользователем
    public void actionPerformed(ActionEvent ae) {
        repaint();
    }
    public void paint(Graphics g) {
        g.drawString("Name: " + name.getText(), 6, 60);
        g.drawString("Selected text in name: "
            + name.getSelectedText(), 6, 80);
        g.drawString("Password: " + pass.getText(),  6,  100);
    }
}
```

Пример вывода апплета TextFieldDemo показан на рис.8.

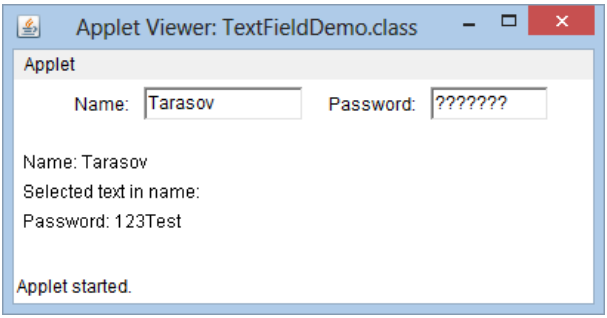

Рис. 8. Пример использования текстовых полей

Использование TextArea

Иногда однострочный текстовый ввод не достаточен для данной задачи. Чтобы обрабатывать эти ситуации, AWT включает простой многострочный редактор, по имени TextArea. Конструкторы TextArea:

```
TextArea ()
TextArea (int numLines, int numChars)
TextArea(String str)
TextArea (String str, int numLines, int numChars)
TextArea (String str, int numLines, int numChars, int sBars)
```

Здесь numLines определяет высоту текстовой области (в строках); numChars — ее ширину (в символах); str — начальный текст. В пятой форме можно определить полосы прокрутки. sBars должен принимать одно из следующих значений:

```
SCROLLBARS_BOTH
SCROLLBARS_HORIZONTAL_ONLY
CROLLBARS_NONE
SCROLLBARS_VERTICAL_ONLY
```

TextArea — подкласс TextComponent, поэтому он поддерживает методы getText(), setText(), getSelectedText(), select(), isEditable() и setEditable(), описанные в предыдущем разделе.

TextArea добавляет следующие методы:

```
void append (String str)
void insert (String str, int index)
void replaceRange (String str, int startIndex, int endIndex)
```

Метод append() добавляет строку, указанную в str, к концу текущего текста, insert() вставляет строку, передаваемую в str, в позицию, указанную в параметре index. Чтобы заменить текст, вызовите метод replaceRange(). Он заменяет символы от startIndex до endIndex - 1 текстом, передаваемым в str.

Текстовые области — почти автономный элемент управления. Ваша программа фактически не берет на себя никакого дополнительного администрирования. Текстовые области генерируют события получения и потери фокуса (got-focus и lost-focus events). Обычно такая программа просто выводит на экран текст, когда это необходимо.

Следующая программа создает элемент управления TextArea:

Программа 123. Текстовые области

```
// Файл TextAreaDemo.java
import java.awt.*;
import java.applet.*;
 /*
<applet code = "TextAreaDemo" width = 300 height = 250>
</applet>
*/
public class TextAreaDemo extends Applet {
   public void init(){
      String val = "There are two ways of constructing "
            + "a software design.\n"
            + "One way is to make it so simple\n"
            + "that there are obviously no deficiencies.\n"
            + "And the other way is to make it so complicated\n"
            + "that there are no obvious deficiencies.\n\n"
            + "Есть два способа сконструировать\n"
            + "проект программного обеспечения\n "
            + "Один путь состоит в том,\n"
            + " чтобы сделать его настолько простым,\n"
            + "чтобы не было очевидных недостатков.\n"
            + "И другой путь состоит в том, \n"
            + "чтобы сделать, его настолько сложным,\n"
            + "чтобы не было очевидных недостатков\n\n"
            + "                              - C.A.R. Hoare\n\n"
            + "There's an old story about the person who wished\n"
            + "his computer were as easy to use as his telephone \n"
            + "That wish has come true,\n"
            + "since I no longer know how to use my telephone.\n\n"
```

```
            + "          - Bjarne Stroustrup, AT&T, (inventor of C++)";
        TextArea text = new TextArea(val, 10, 30);
        add(text);
    }
}
```

Пример вывода апплета TextAreaDemo представлен на рис.9.

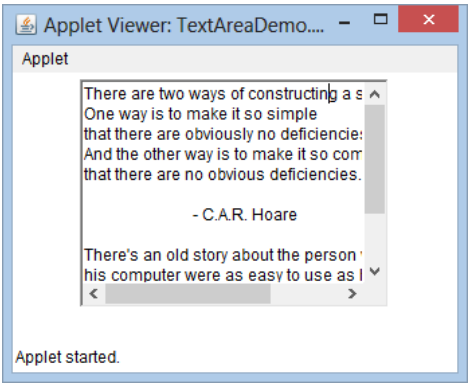

Рис. 9. Многострочный текст

Понятие менеджера компоновки

Все компоненты, которые в разобранных примерах, были размещены (позиционированы) в контейнере менеджером компоновки, заданным по умолчанию. Менеджер компоновки автоматически размещает элементы управления в пределах окна, используя некоторый тип алгоритма. В других GUI-средах, таких как Windows, элементов управления размещаются вручную. Хотя элементы управления Java тоже можно размещать вручную, в этом нет необходимости по двум причинам. Во-первых, очень утомительно вручную размещать большое количество компонентов. Во-вторых, иногда, в тот момент, когда нужно размещать некоторый элемент, информация о его размерах оказывается недоступной (чаще всего потому, что необходимые для получения этой информации компоненты комплекта инструментов еще не были реализованы). Правильное же размещение компонента без определенных размеров относительно других компонентов весьма проблематично. Поэтому разработаны специальные программные компоненты (менеджеры компоновки) для автоматического управления компоновкой.

Каждый объект типа Container имеет связанный с ним менеджер компоновки. Менеджер компоновки — это экземпляр некоторого класса, который реализует интерфейс LayoutManager. Менеджер

компоновки устанавливается методом setLayout(). Если вызов setLayout() не сделан, то используется менеджер компоновки по умолчанию (поточный). Менеджер компоновки применяется всякий раз, когда контейнер изменяется в размерах (или устанавливается впервые — с первоначальными размерами), чтобы позиционировать каждый из его внутренних компонентов.

Метод SetLayout() имеет следующую общую форму:

```
void setLayout (LayoutManager layoutObj)
```

Здесь layoutObj — ссылка на необходимый менеджер компоновки. Можно отключить менеджер компоновки и позиционировать компоненты вручную, передав в качестве параметра layoutObj значение null (пустой указатель). После этого нужно будет определять форму и позицию каждого компонента вручную, вызывая метод setBounds(), определенный в Component. Но, как правило, следует использовать менеджер компоновки.

Каждый менеджер компоновки хранит и отслеживает список имен компонентов. Всякий раз, когда добавляется компонент в контейнер, менеджер компоновки получает соответствующее уведомление. Если нужно изменять размеры контейнера, менеджер компоновки консультируется со своими методами minimumLayoutSize() и preferredLayoutSize(). Каждый компонент, который управляется менеджером компоновки, содержит методы getPreferredSize() и getMinimumSize(). Они возвращают предпочтительный и минимальный размер, требуемый для отображения каждого компонента. Если это вообще возможно, менеджер компоновки будет удовлетворять эти запросы (для поддержки целостности политики компоновки). Для элементов управления своего подкласса можно переопределить указанные методы. Иначе используются значения по умолчанию.

Java имеет несколько предопределенных классов LayoutManager, которые описываются ниже. Используйте тот менеджер компоновки, который наилучшим образом подходит вашему приложению.

Менеджер FlowLayout

FlowLayout — это менеджер *поточной* компоновки. Напомним, что если метод setLayout() не устанавливает никакой иной компоновщик, то данный компоновщик используется по умолчанию. Этот менеджер использовали все предшествующие примеры. FlowLayout реализует простой стиль размещения, похожий на поток слов в текстовом редакторе. Компоненты размещаются от левого верхнего угла окна, слева направо и сверху вниз. Когда нет больше компонентов, пригодных для размещения на строке, очередной компонент

размещается в начале следующей строки. Выше и ниже, справа и слева, а также между каждым компонентом оставляется маленькое пространство.

Конструкторы FiowLayout.

```
FlowLayout()
FlowLayout (int how)
FlowLayout (int how, int horz, int vert)
```

Первая форма создает размещение по умолчанию, которое выравнивает компоненты по центру и оставляет пять пикселов пробела между каждым компонентом. Вторая форма позволяет определить, как выравнивается каждая строка. Допустимы следующие значения параметра how:

- FlowLayout.LEFT
- FlowLayout.CENTER
- FlowLayout.RIGHT

Эти значения определяют выравнивание влево, по центру и вправо, соответственно. Третья форма позволяет задавать (в параметрах horz и vert) горизонтальный и вертикальный пробел, оставляемый между компонентами (в форме целого числа пикселов).

Ниже приводится версия апплета CheckboxDemo, показанного ранее в этой главе, модифицированная так, чтобы использовать поточное размещение, выровненное по левой границе.

Программа 124. Размещение компонент с левым выравниванием

```
// Файл FlowLayoutDemo.java
// Использует поточное размещение с левым выравниванием.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = "FlowLayoutDemo" width = 250 height = 200>
</applet>
*/
public class FlowLayoutDemo extends Applet implements ItemListener {
   String msg = "";
   Checkbox Win98, winNT, Solaris, mac;
   public void init() {
// Установить поточное размещение с левым выравниванием
   setLayout(new FlowLayout(FlowLayout.LEFT));
      Win98 = new Checkbox("Windows 98", null, true);
      winNT = new Checkbox("Windows NT");
      Solaris = new Checkbox("Solaris");
      mac = new Checkbox("MacOS");
      add(Win98);
      add(winNT);
```

```
        add(Solaris);
        add(mac);
// Зарегистрироваться для приема item-событий
        Win98.addItemListener(this);
        winNT.addItemListener(this);
        Solaris.addItemListener(this);
        mac.addItemListener(this);
    }
// Перерисовать, когда изменяется состояние флажка
    public void itemStateChanged(ItemEvent ie) {
        repaint ();
    }
// Показать текущее состояние флажков
    public void paint(Graphics g) {
        msg = "Current state: ";
        g.drawString(msg, 6, 80);
        msg = " Windows 98: " + Win98.getState();
        g.drawString(msg, 6, 100);
        msg = " Windows NT: " + winNT.getState();
        g.drawString (msg, 6, 120);
        msg = " Solaris: " + Solaris.getState();
        g.drawString(msg, 6, 140);
        msg = " Mac: " + mac.getState();
        g.drawString(msg, 6, 160);
    }
}
```

Пример вывода, сгенерированного апплетом FlowLayoutDemo, представлен на рис.10. Сравните это с выводом апплета CheckboxDemo, показанного ранее на рис.3.

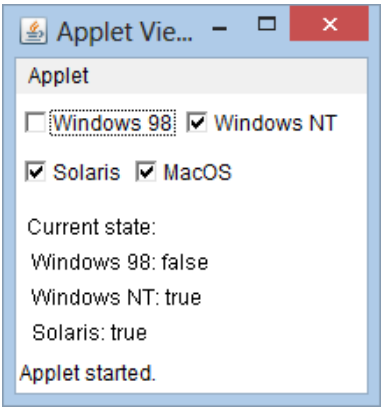

Рис. 10. Левое выравнивание компонентов

Класс BorderLayout

Класс BorderLayout реализует *граничный* стиль компоновки, используемый для окон верхнего уровня. Он имеет четыре узких

компонента фиксированной ширины по краям и один — в виде большой области — в центре. Четыре краевых компонента называют Север (North), Юг (South), Восток (East) и Запад (West). Средняя область называется Центр (Center). Конструкторы, определенные в BorderLayout:

```
BorderLayout()
BorderLayout(int horz, int vert)
```

Первая форма создает граничное размещение, используемое по умолчанию. Вторая — позволяет указывать количество (в параметрах horz и vert, соответственно) горизонтальных и вертикальных пробелов, оставляемых между компонентами. BorderLayout определяет следующие константы, которые специфицируют области размещения:

- BorderLayout.CENTER
- BorderLayout.SOUTH
- BorderLayout.EAST
- BorderLayout.WEST
- BorderLayout.NORTH

При добавлении компонентов вы будете использовать эти константы со следующей формой метода add(), который определен в классе Container:

```
void add(Component campObj, Object region);
```

Здесь campObj — компонент, который будет добавлен, а region специфицирует область размещения, куда компонент будет добавлен.

Пример граничного размещения (менеджером BorderLayout) с компонентом в каждой области компоновки:

Программа 125. Размещение компонент по границам и в центре

```
// Файл BorderLayoutDemo.java
// Демонстрирует BorderLayout.
import java.awt.*;
import java.applet.*;
import java.util.*;
/*
<applet code = "BorderLayoutDemo" width = 400 height = 200>
</applet>
*/

public class BorderLayoutDemo extends Applet {
   public void init() {
      setLayout(new BorderLayout());
      add(new Button("This is across the top."),
            BorderLayout.NORTH);
      add(new Label("The footer message might go here."),
            BorderLayout.SOUTH);
```

```
        add(new Button("Right"), BorderLayout.EAST);
        add(new Button("Left"), BorderLayout.WEST);
        String msg = "The reasonable man adapts " + "himself to the world;\n"
                   + "the unreasonable one persists in "
                   + "trying to adapt the world to himself An"
                   + "Therefore all progress depends "
                   + "on the unreasonable man.\n\n"
                    + "       - George Bernard Shaw\n\n";
          add(new TextArea(msg), BorderLayout.CENTER);
    }
}
```

Пример вывода апплета BorderLayoutDemo представлен на рис.11.

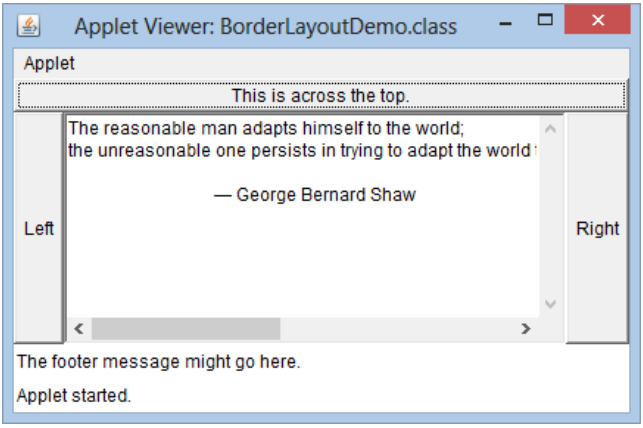

Рис. 11. Выравнивание по границам и центру

Использование вставок

Иногда нужно оставить немного пустого места между контейнером, который хранит компоненты, и окном, содержащим контейнер. Для этого необходимо переопределить метод getInsets(), который определен в классе Container. Эта функция возвращает объект типа Insets, содержащий верхнюю, нижнюю, левую и правую вставки, которые используются во время отображения контейнера. Менеджер компоновки использует эти значения при вставке компонентов, когда размещает окно вокруг контейнера. Конструктор Insets:

```
Insets (int top, int left, int bottom, int right)
```

Значения, пересылаемые параметрами top, left, bottom, righ определяют количество пробельного пространства (в пикселах) между контейнером и включающим его окном.

Метод getInsets() имеет общую форму:

```
Insets getInsets()
```

При переопределении одного из этих методов нужно возвратить новый Insets-объект, который содержит необходимую пробельную вставку.

Ниже показан предшествующий пример с компоновщиком BorderLayout, измененный так, что он вставляет компоненты в десяти пикселах от каждой границы. Чтобы лучше видеть вставки, установлен голубой цвет фона.

Программа 126. Использование вставок для выравнивания компонент

```
// Файл InsetsDemo.java
// Демонстрирует BorderLayout со вставками.
import java.awt.*;
import java.applet.*;
import java.util.*;
/*
<applet code = "InsetsDemo" width = 400 height = 200>
</applet>
*/
public class InsetsDemo extends Applet {
   public void init() {
// Установить цвет фона так, чтобы вставки были легко видимы
      setBackground(Color.cyan);
      setLayout(new BorderLayout());
      add(new Button("This is across the top."), BorderLayout.NORTH);
      add(new Label("The footer message might go here."),
      BorderLayout.SOUTH);
      add(new Button("Right" ), BorderLayout.EAST);
      add(new Button("Left"), BorderLayout.WEST);
      String msg = "The reasonable man adapts "
                 + "himself to the world;\n"
                 + "the unreasonable one persists in "
                 + "trying to adapt the world to himself.\n"
                 + "Therefore all progress depends "
                 + "on the unreasonable man.\n\n"
                 + " - George Bernard Shaw\n\n";
      add (new TextArea (msg), BorderLayout.CENTER);
   }
// добавить вставки
   public Insets getInsets() {
      return new Insets (10, 10, 10, 10);
   }
}
```

Вывод апплета InsetsDemo представлен на рис.12.

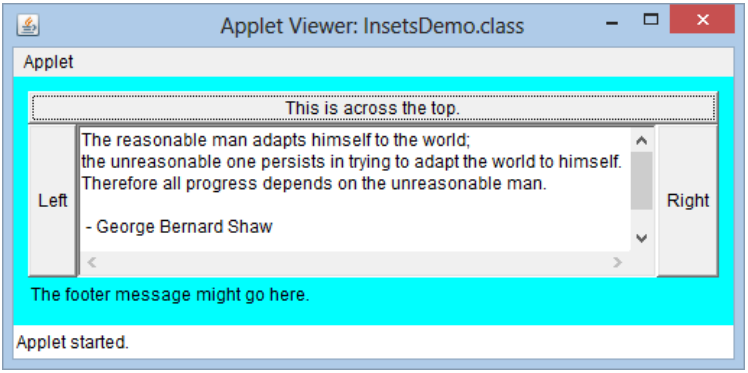

Рис. 12. Вставки промежутков для выравнивания

Менеджер GridLayout

Менеджер GridLayout располагает компоненты в двумерной сетке. Число строк и столбцов сетки следует определять при создании экземпляра (объекта) GridLayout. Конструкторы, определенные в GridLayout:

```
GridLayout()
GridLayout(int numRows, int numColumns)
GridLayout(int numRows, int numColumns, int horz, int vert)
```

Первая форма создает сеточное размещение с одиночным столбцом. Вторая — сеточное размещение с указанным числом строк и столбцов. Третья форма позволяет определять горизонтальный и вертикальный пробелы, оставляемые между компонентами (в параметрах horz и vert, соответственно). Параметр numRows или numColumns может быть нулевым. Нулевая спецификация numRows допускает столбцы с неограниченной длиной. Нулевая спецификация numColumns допускает строки с неограниченной длиной.

Пример программы, которая создает сетку 4x4 и вставляет в нее 15 кнопок, каждая из которых помечена своим индексом (порядковым номером):

Программа 127. Табличное размещение компонентов

```
// Файл GridLayoutDemo.java
// Демонстрирует GridLayout.
import java.awt.*;
import java.applet.*;
/*
<applet code = "GridLayoutDemo" width = 300 height = 200>
</applet>
*/
```

```
public class GridLayoutDemo extends Applet {
   static final int n = 4;      // Число строк и столбцов сетки
   public void init() {
      setLayout(new GridLayout(n, n));
      setFont(new Font("SansSerif", Font.BOLD, 24));
      for(int i = 0; i < n; i++) {
         for(int j = 0; j < n; j++) {
            int k = i * n + j;           // Номер на кнопке
            if(k > 0)
               add(new Button("" + k));
         }
      }
   }
}
```

Вывод, сгенерированный апплетом GridLayoutDemo, представлен на рис. 13.

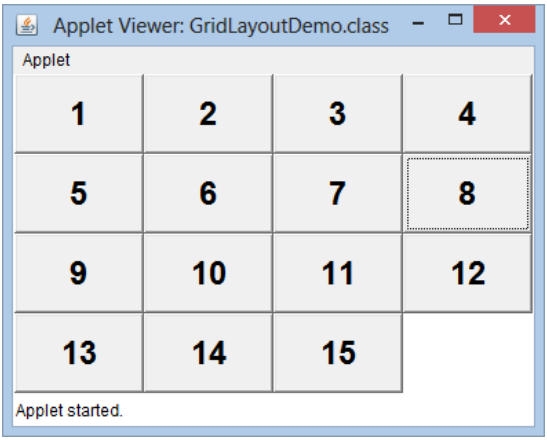

Рис. 13. Сеточное размещение кнопок

Этот пример можно использовать как отправную точку для головоломки "игра в 15".

Класс CardLayout

Класс CardLayout уникален среди других менеджеров компоновки тем, что хранит несколько различных размещений. Каждое размещение можно предствлять отдельной пронумерованной картой в колоде, которая может быть перетасована так, чтобы в данный момент наверху находилась любая карта. Это может быть полезно для интерфейсов пользователя с необязательными компонентами, которые можно динамически включать и выключать при вводе пользователя. Вы можете подготовить другие компоновки и сделать их скрытыми, но

готовыми к активизации, когда это необходимо. CardLayout обеспечивает два конструктора:

```
CardLayout ()
CardLayout (int horz, int vert)
```

Первая форма создает карточную компоновку по умолчанию. Вторая — позволяет указывать горизонтальный и вертикальный пробел, оставляемый между компонентами (в параметрах horz и vert, соответственно).

Использование карточной компоновки требует немного большей работы, чем в других компоновках. Карты обычно содержатся в объекте типа Panel. Эта панель должна выбрать CardLayout, как свой менеджер компоновки. Карты, которые формируют колоду, также обычно являются объектами типа Panel. Таким образом, нужно создать панель, которая содержит колоду и панель для каждой карты в колоде. Затем добавить к соответствующей панели компоненты, которые формируют каждую карту. Далее, вы добавляете карточные панели к панели менеджера CardLayout. Наконец, вы добавляете эту панель к главной панели апплета. Как только эти шаги закончены, вы должны обеспечить для пользователя некоторый способ выбора между панелями. Один из обычных подходов состоит в том, чтобы ввести по одной командной кнопке для каждой карты в колоде.

Когда карточные панели добавляются в панель колоды (т. е. компоновщика), им обычно присваиваются имена. Для добавления карт в панель колоды в большинстве случаев будет использоваться следующая форма метода add():

```
void add (Component panelobj, Object name) ;
```

Здесь name — строка, которая определяет имя карты, чья панель указана в параметре panelObj.

После того как вы создали колоду, программа активизирует карту, вызывая один из следующих методов, определенных в CardLayout:

```
void first(Container deck)
void last(Container deck)
void next(Container deck)
void previous (Container deck)
void show(Container deck, String cardtfame)
```

Здесь deck — ссылка на контейнер (обычно — панель), который содержит карты, а cardName — имя карты. Вызов first() заставляет показать первую карту в колоде. Чтобы отобразить последнюю карту, вызовите last(). Для показа очередной карты вызывайте next(), а предыдущей — previous(). Как nex(), так и previous() автоматически

повторяют цикл сверху или снизу колоды, соответственно. Метод show() отображает карту, чье имя передается в cardName.

Следующий пример создает двухуровневую колоду, которая позволяет пользователю выбирать операционную систему. Операционные системы на базе Windows отображаются на одной карте, Macintosh и Solaris — на другой.

Программа 128. Карточная компоновка

```
// Файл CardLayoutDemo.java
// Демонстрирует CardLayout.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = "CardLayoutDemo" width = 300 height = 100>
</applet>
*/
public class CardLayoutDemo
      extends Applet
      implements ActionListener, MouseListener {
   Checkbox Win98, winNT, Solaris, mac;
   Panel osCards;
   CardLayout cardLO;
   Button Win, Other;
   public void init() {
      Win = new Button("Windows");
      Other = new Button("Other");
      add(Win);
      add(Other);
      cardLO = new CardLayout();
      osCards = new Panel();
      osCards.setLayout(cardLO);
   // Установить panel-компоновку
   // для card-компоновки
      Win98 = new Checkbox("Windows 98", null, true);
      winNT = new Checkbox("Windows NT");
      Solaris = new Checkbox("Solaris");
      mac = new Checkbox("MacOS");
   // Добавить в панель Windows флажки
      Panel winPan = new Panel();
      winPan.add(Win98);
      winPan.add(winNT);
   // Добавить в панель другие OS-флажки
      Panel otherPan = new Panel();
      otherPan.add(Solaris);
      otherPan.add(mac);
   // Добавить панели к панели колоды карт
      osCards.add(winPan, "Windows");
      osCards.add(otherPan, "Other");
   // Добавить карты к главной панели апплета
      add(osCards);
   // Регистрироваться для приема action-событий
```

```
      Win.addActionListener(this);
      Other.addActionListener(this);
   // Регистрировать события мыши
      addMouseListener(this);
   }
// Цикл карт в панели
   public void mousePressed(MouseEvent me) {
      cardLO.next(osCards);
   }
// Обеспечить пустые реализации для других методов MouseListener
   public void mouseClicked(MouseEvent me) { }
   public void mouseEntered(MouseEvent me) { }
   public void mouseExited(MouseEvent me)  { }
   public void mouseReleased (MouseEvent me) { }
   public void actionPerformed(ActionEvent ae) {
      if(ae.getSource() == Win) {
         cardLO.show(osCards, "Windows");
      }
      else {
         cardLO.show(osCards, "Other");
      }
   }
}
```

На рис. 14 представлен вывод, сгенерированный апплетом cardLayoutDemo. Каждая карта активизируется нажатием своей кнопки. Допустим также циклический просмотр карт (щелчками мыши).

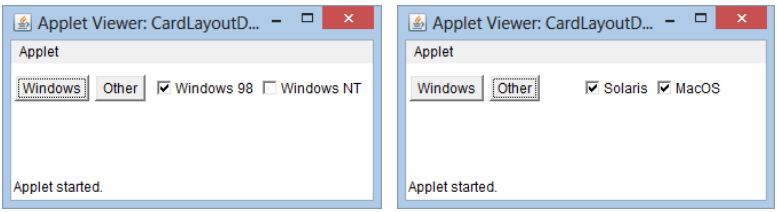

Рис. 14. Размещение компонентов на двух панелях

Панели меню и меню

Окно верхнего уровня может содержать связанную с ним панель меню (menu bar) верхнего уровня. В строке этого меню отображается список выбираемых элементов (выборов, choices). Каждый выбираемый элемент панели меню верхнего уровня связан с выпадающим (drop-down) меню, содержащим свои выбираемые элементы. Эти элементы меню реализованы в Java следующими классами:

MenuBar – меню верхнего уровня
Menu - впадающее меню
Menuitem – пункт меню.

В общем случае, панель меню содержит один или несколько объектов типа Menu. Каждый Menu-объект содержит список. Любой объект типа MenuItem представляет нечто, что может быть выбрано пользователем. Элементом списка в Menu-объекте может быть не только MenuItem-объект, но и другой Menu-объект со своим списком MenuItem-объектов. Таким образом может быть создана иерархия вложенных подменю. В меню можно также включать специальные элементы — с отметками. Они являются меню-элементами типа checkboxMenuItem и, когда они выбираются, перед ними проставляется специальная метка (check mark).

Чтобы создать панель меню, сначала создают экземпляр типа MenuBar. Этот класс определяет только конструктор по умолчанию. Затем создаются экземпляры типа Menu, которые определяют выбираемые элементы, расположенные на строке меню. Конструкторы класса Menu:

```
Menu()
Menu (String optionName)
Menu(String optionName, boolean removable)
```

Здесь optionName специфицирует имя выбираемого элемента меню-панели. Если параметр removable — true, выпадающее меню может быть удалено (из иерархии) и отправлено "в свободное плавание" (по окну). В противном случае это меню останется прикрепленным к панели меню. (Откреплляемые меню зависят от реализации.) Первая форма создает пустое меню.

Индивидуальные пункты меню имеют тип MenuItem. В этом классе определены следующие конструкторы:

```
MenuItem()
MenuItem(String ItemName)
MenuItem (String ItemName, MenuShortcut keyAccel)
```

где ItemName — имя, показанное в меню; keyAccel — клавиши быстрого доступа для этого пункта.

Вы можете деактивизировать или активизировать пункт меню, используя метод setEnabled(). Его формат:

```
void setEnabled (boolean enabledFlag)
```

Если параметр enabledFlag— true, пункт меню активизируется, если false — пункт деактивизируется (блокируется).

Можно определять состояние элемента, вызывая isEnabied(). Формат этого метода:

```
boolean isEnabled()
```

isEnabied() возвращает true, если пункт меню активизирован. Иначе, возвращается false.

Можно изменять имя пункта меню, вызывая метод setLabel(), и извлекать имя текущего пункта, вызывая метод getLabel(). Форматы этих методов:

```
void setLabel (String newName)
String getLabel()
```

где nevName устанавливает новое имя пункта меню. getLabel() возвращает текущее имя.

Можно создавать помечаемый пункт меню, используя подкласс MenuItem с именем CheckboxMenuItem. Он имеет следующие конструкторы:

```
CheckboxMenuItem ()
CheckboxMenuItem(String itemName)
CheckboxMenuItem(String itemName, boolean on)
```

где itemName — имя, показываемое в меню. Отмечаемые пункты работают как флажки. Каждый раз при выборке их состояние изменяется. В первых двух формах создается пункт без отметки. В третьей форме, если параметр on указан как true, создается изначально помеченный пункт, иначе — непомеченный.

Состояние помечаемого пункта можно получить, вызывая getState(). Для установки состояния такого пункта используется метод setsState(). Форматы этих методов:

```
boolean getState()
void setState (boolean checked)
```

Если элемент отмечен, getsState() возвращает true, иначе — false. Чтобы пометить пункт, передайте в setState() значение true. Чтобы удалить метку, передайте false.

Как только вы создали пункт меню, следует добавить элемент в объект типа Menu с помощью метода add(), который имеет следующую общую форму:

```
MenuItem add (MenuItem item)
```

где item— добавляемый пункт. Пункты добавляются к меню в порядке, в котором выполняются вызовы add(). Объект item используется и как возвращаемое значение.

Как только вы добавили все элементы к Menu-объекту, можно добавить этот объект в строку меню, используя следующую версию метода add(), определенную в MenuBar:

```
Menu add (Menu menu)
```

где menu — объект, представляющий как добавляемое меню, так и возвращаемое значение.

Меню генерируют события только тогда, когда выбираются элементы типа MenuItem или checkboxMenuItem. Такие события не генерируются, например, когда обращаются к строке меню, чтобы открыть выпадающее меню. При каждой выборке обычного (без отметки) пункта меню генерируется объект ActionEvent. При очередном сбросе или установке флажка помеченного пункта меню генерируется объект ItemEvent. Таким образом, чтобы обработать эти меню-события, нужно реализовать интерфейсы блоков прослушивания ActionListener и ItemListener.

Метод getItem() класса ItemEvent возвращает ссылку на элемент, который генерировал это событие. Общая форма этого метода:

```
Object getItem()
```

Ниже показан пример, который добавляет ряд вложенных меню в окно раскрывающегося меню (второго уровня). Выбранный элемент выделяется в окне инверсной строкой. Отображается также состояние двух помечаемых пунктов меню (один — включен, другой — нет).

Программа 129. Создание меню

```
// Файл MenuDemo.java
// Иллюстрирует меню.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = "MenuDemo" width = 250 height = 250>
</applet>
*/
// Создать подкласс для Frame
class MenuFrame extends Frame {
   String msg = "";
   CheckboxMenuItem debug, test;  // Пункты меню-флажки
   MenuFrame(String title) {
      super (title);
   // Создать строку главного меню и добавить ее во фрейм
      MenuBar mbar = new MenuBar();
      setMenuBar(mbar);
   // Создать элементы меню
      Menu file = new Menu("File");
      MenuItem item1, item2, item3, item4, item5;
      file.add(item1 = new MenuItem("New..."));
      file.add(item2 = new MenuItem("Open..."));
      file.add(item3 = new MenuItem("Close"));
      file.add(item4 = new MenuItem("-"));
      file.add(item5 = new MenuItem("Quit..."));
      mbar.add(file);
      Menu edit = new Menu("Edit");
```

```
        MenuItem item6, item7, item8, item9;
        edit.add(item6 = new MenuItem("Cut"));
        edit.add(item7 = new MenuItem("Copy"));
        edit.add(item8 = new MenuItem("Paste"));
        edit.add(item9 = new MenuItem("-"));
        Menu sub = new Menu("Special");
        MenuItem item10, item11, item12;  // Подпункты для меню Special
        sub.add(item10 = new MenuItem("First"));
        sub.add(item11 = new MenuItem("Second"));
        sub.add(item12 = new MenuItem("Third"));
        edit.add(sub);        // Добавление подменю к Special
    // Это элементы меню с метками.
        debug = new CheckboxMenuItem("Debug");
        edit.add(debug);
        test = new CheckboxMenuItem ("Testing");
        edit.add(test);
        mbar.add(edit) ;
    // Создать объект для обработки action- и item-событий
        MyMenuHandler handler = new MyMenuHandler(this);
    // `Регистрировать его для приема этих .событий
        item1.addActionListener(handler);
        item2.addActionListener(handler);
        item3.addActionListener(handler);
        item4.addActionListener(handler);
        item5.addActionListener(handler);
        item6.addActionListener(handler);
        item7.addActionListener(handler);
        item8.addActionListener(handler);
        item9.addActionListener(handler);
        item10.addActionListener(handler);
        item11.addActionListener(handler);
        item12.addActionListener(handler);
        debug.addItemListener(handler);
        test.addItemListener(handler);
    // Создать объект для обработки window-событий
        MyWindowAdapter adapter = new MyWindowAdapter(this);
    // Register it to receive those events
        addWindowListener(adapter);
    }
    public void paint(Graphics g) {
        g.drawString(msg, 10, 200);
        if(debug.getState())
            g.drawString("Debug is on.", 10, 220);
        else
            g.drawString("Debug is off.", 10, 220);
        if(test.getState())
            g.drawString("Testing is on.",  10, 240);
        else
            g.drawString("Testing is off.", 10, 240);
    }
}
class MyWindowAdapter extends WindowAdapter {
    MenuFrame menuFrame;
    public MyWindowAdapter(MenuFrame menuFrame) {
        this.menuFrame = menuFrame;
```

```
    }
    public void windowClosing(WindowEvent we) {
       menuFrame.setVisible(false);
    }
}
class MyMenuHandler implements ActionListener, ItemListener {
    MenuFrame menuFrame;
    public MyMenuHandler(MenuFrame menuFrame) {
       this.menuFrame = menuFrame;
    }
// Обработка action-событий.
    public void actionPerformed(ActionEvent ae) {
       String msg = "You selected ";
       String arg = (String)ae.getActionCommand();
       if(arg.equals("New..."))
          msg += "New.";
       else if(arg.equals("Open..."))
          msg += "Open.";
       else if(arg.equals("Close"))
          msg += "Close.";
       else if(arg.equals("Quit..."))
          msg += "Quit.";
       else if(arg.equals("Edit"))
          msg += "Edit.";
       else if(arg.equals("Cut"))
          msg += "Cut.";
       else if(arg.equals("Copy"))
          msg += "Copy.";
       else if(arg.equals("Paste"))
          msg += "Paste.";
       else if(arg.equals("First"))
          msg += "First.";
       else if(arg.equals("Second"))
          msg += "Second.";
       else if(arg.equals("Third"))
          msg += "Third.";
       else if(arg.equals("Debug"))
          msg += "Debug.";
       else if(arg.equals("Testing"))
          msg += "Testing.";
       menuFrame.msg = msg;
       menuFrame.repaint();
    }
// Обработка item-событий
    public void itemStateChanged(ItemEvent ie) {
       menuFrame.repaint();
    }
}
// Создать frame-окно
public class MenuDemo extends Applet {
    Frame f;
    public void init() {
       f = new MenuFrame("Menu Demo");
       int width = Integer.parseInt(getParameter("width"));
       int height = Integer.parseInt(getParameter("height"));
```

```
        setSize(new Dimension(width, height));
        f.setSize(width, height);
        f.setVisible(true);
    }
    public void start() {
        f.setVisible(true) ;
    }
    public void stop() {
        f.setVisible(false);
    }
}
```

Пример вывода апплета MenuDemo показан на рис.15.

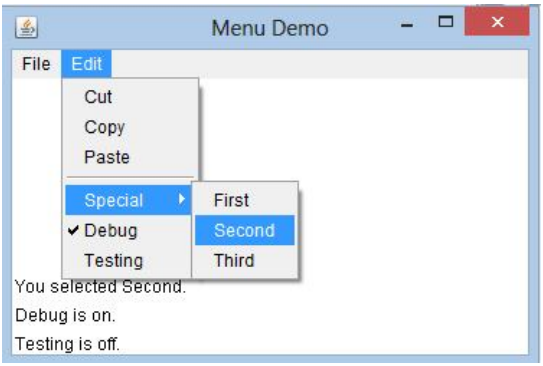

Рис. 15. Работа меню

Имеется еще один класс, связанный с меню, который может показаться интересным — PopupMenu. Он работает точно так же, как Menu, но создает меню, которое может быть отображено в определенном месте окна апплета. В некоторых ситуациях PopupMenu обеспечивает гибкую, полезную альтернативу классу Menu.

Диалоговые окна

Часто необходимо использовать диалоговое окно, содержащее набор связанных элементов управления. Диалоговые окна первоначально использовались для получения ввода от пользователя. Они подобны фрейм-окнам, за исключением того, что диалоговые окна — всегда дочерние окна для окна верхнего уровня. Кроме того, диалоговые окна не имеют строки меню. В других отношениях они функционируют подобно фреймовым окнам. (Можно, например, добавлять к ним элементы управления тем же способом, каким добавляются элементы управления к фреймовому окну.) Диалоговые окна могут быть модальными или немодальными. Когда модальное диалоговое окно активно, весь ввод направляется к нему, пока оно не

будет закрыто. Это означает, что вы не можете обращаться к другим частям программы до тех пор, пока не закрыли диалоговое окно. Когда немодальное диалоговое окно активно, фокус ввода может быть направлен другому окну вашей программы. Таким образом, другие части вашей программы остаются активными и доступными. Диалоговые окна обслуживает класс Dialog. Обычно используются следующие конструкторы этого класса:

```
Dialog (Frame parentWindow, boolean mode)
Dialog (Frame parentWindow, String title, boolean mode)
```

Здесь parentWindow — владелец диалогового окна. Если mode имеет значение true, диалоговое окно является модальным. Иначе, оно — немодальное. Заголовок диалогового окна можно передать через параметр title. В общем случае, ваша программа будет подклассом класса Dialog, добавляющим функциональные возможности, необходимые приложению.

Далее следует модифицированная версия предшествующей меню-программы, которая отображает немодальное диалоговое окно с выбранным пунктом New. Обратите внимание, что, когда диалоговое окно закрывается, вызывается метод dispose(). Данный метод определен в классе Window и освобождает все системные ресурсы, связанные с диалоговым окном.

Программа 130. Окна диалога

```
// Файл DialogDemo.java
// Демонстрирует диалоговое окно.
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
/*
<applet code = " DialogDemo" width = 250 height = 250>
</applet>
*/
// Создать подкласс класса Dialog
class SampleDialog extends Dialog implements ActionListener {
   SampleDialog(Frame parent, String title) {
      super(parent, title, false);
      setLayout(new FlowLayout());
      setSize(300, 200);
      add(new Label("Press this button:"));
      Button b;
      add(b = new Button("Cancel"));
      b.addActionListener(this);
   }
   public void actionPerformed(ActionEvent ae) {
      dispose();
   }
   public void paint(Graphics g) {
```

```
      g.drawString("This is in the dialog box", 10, 70);
   }
}
// Создать подкласс класса Frame
class MenuFrame extends Frame {
   String msg = "";
   CheckboxMenuItem debug, test;
   MenuFrame(String title) {
      super(title);
      // Создать строку меню и добавить ее к фрейму
      MenuBar mbar = new MenuBar();
      setMenuBar(mbar);
      // Создать пункты меню
      Menu file = new Menu("File");
      MenuItem item1, item2, item3, item4;
      file.add(item1 = new MenuItem("New..."));
      file.add(item2 = new MenuItem("Open..."));
      file.add(item3 = new MenuItem("Close"));
      file.add(new MenuItem("-"));
      file.add(item4 = new MenuItem("Quit..."));
      mbar.add(file);
      Menu edit = new Menu("Edit");
      MenuItem item5, item6, item7;
      edit.add(item5 = new MenuItem("Cut"));
      edit.add(item6 = new MenuItem("Copy"));
      edit.add(item7 = new MenuItem("Paste"));
      edit.add(new MenuItem("-"));
      Menu sub = new Menu("Special", true);
      MenuItem item8, item9, item10;
      sub.add(item8 = new MenuItem("First"));
      sub.add(item9 = new MenuItem("Second"));
      sub.add(item10 = new MenuItem("Third"));
      edit.add(sub);
      // Это помечаемые пункты
      debug = new CheckboxMenuItem("Debug") ;
      edit.add(debug);
      test = new CheckboxMenuItem("Testing");
      edit.add(test);
      mbar.add(edit);
      // создать объекты для обработки action- и item-событий
      MyMenuHandler handler = new MyMenuHandler(this);
      // Зарегистрировать их для приема этих событий
      item1.addActionListener(handler);
      item2.addActionListener(handler) ;
      item3.addActionListener(handler) ;
      item4.addActionListener(handler);
      item5.addActionListener(handler);
      item6.addActionListener(handler);
      item7.addActionListener(handler) ;
      item8.addActionListener(handler);
      item9.addActionListener(handler);
      item10.addActionListener(handler);
      debug.addItemListener(handler) ;
      test.addItemListener(handler);
      // Создать объект для обработки window-событий
```

```
      MyWindowAdapter adapter = new MyWindowAdapter(this);
      // Зарегистрировать его для приема этих событий
      addWindowListener(adapter);
   }
   public void paint(Graphics g) {
      g.drawString(msg, 10, 200);
      if(debug.getState())
         g.drawString("Debug is on.", 10, 220);
      else
         g.drawString("Debug is off.", 10, 220);
      if(test.getState())
         g.drawString("Testing is on.", 10, 240);
      else
         g.drawString("Testing is off.", 10, 240);
   }
}
class MyWindowAdapter extends WindowAdapter {
   MenuFrame menuFrame;
   public MyWindowAdapter (MenuFrame menuFrame) {
      this.menuFrame = menuFrame;
   }
   public void windowClosing(WindowEvent we) {
      menuFrame.dispose();
   }
}
class MyMenuHandler implements ActionListener, ItemListener {
   MenuFrame menuFrame;
   public MyMenuHandler(MenuFrame menuFrame) {
      this.menuFrame = menuFrame;
   }
// Обработать action-события
   public void actionPerformed(ActionEvent ae) {
      String msg = "You selected ";
      String arg = (String)ae.getActionCommand();
   // Активизировать диалоговое окно, когда выбран пункт New
      if(arg.equals("New...")) {
         msg += "New.";
         SampleDialog d = new SampleDialog(menuFrame, "New Dialog Box");
         d.setVisible(true);
      }
      // Попытка определить другие окна диалога для этих пунктов
      else if(arg.equals("Open..."))
         msg += "Open.";
      else if(arg.equals("Close"))
         msg += "Close.";
      else if(arg.equals("Quit..."))
         msg += "Quit.";
      else if(arg.equals("Edit"))
         msg += "Edit.";
      else if(arg.equals("Cut"))
         msg += "Cut.";
      else if(arg.equals("Copy"))
         msg += "Copy.";
      else if(arg.equals("Paste"))
         msg += "Paste.";
```

```
      else if(arg.equals("First"))
         msg += "First.";
      else if(arg.equals("Second"))
         msg += "Second.";
      else if(arg.equals("Third"))
         msg += "Third.";
      else if(arg.equals("Debug"))
         msg += "Debug.";
      else if(arg.equals("Testing"))
         msg += "Testing.";
      menuFrame.msg = msg;
      menuFrame.repaint();
   }
   public void itemStateChanged(ItemEvent ie) {
      menuFrame.repaint();
   }
}
// Создать frame-окно
public class DialogDemo extends Applet {
   Frame f;
   public void init() {
      f = new MenuFrame("Menu Demo");
      int width = Integer.parseInt(getParameter("width"));
      int height = Integer.parseInt(getParameter("height"));
      setSize(width, height);
      f.setSize(width, height);
      f.setVisible(true);
   }
   public void start() {
      f.setVisible(true);
   }
   public void stop() {
      f.setVisible(false);
   }
}
```

Пример вывода модифицированного апплета `MenuDemo` представлен на рис. 16. При выборе пункта меню File, New выводится диалоговое окно.

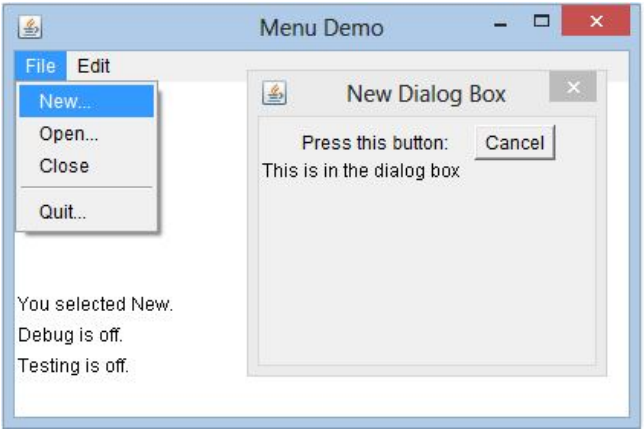

Рис. 16. Создание диалогового окна

Класс FileDialog

Java обеспечивает встроенное диалоговое окно, которое дает возможность пользователю специфицировать файл. Для открытия этого окна программе достаточно создать конкретный экземпляр объекта типа FileDialog. По форме это стандартное файловое диалоговое окно, используемое операционной системой для открытия файлов (см. вывод программы). Класс FileDialog обеспечивает следующие конструкторы:

```
FileDialog (Frame parent, String boxName)
FileDialog (Frame parent, String boxName, int hov)
FileDialog (Frame parent)
```

Здесь parent — владелец диалогового окна; boxName — имя, отображаемое в области заголовка окна. Если boxName опущен, заголовок диалогового окна остается пустым. Если hov имеет значение FileDialog.load, то окно выбирает файл для чтения, а если how имеет значение FileDialog.save, окно выбирает файл для записи (с целью сохранения). Третий конструктор создает диалоговое окно с выбором файла для чтения.

FileDialog содержит методы, которые позволяют определить имя и путь файла, выбранного пользователем, например:

```
String getDirectory()
String getFile()
```

Эти методы возвращают каталог и имя файла, соответственно. Следующая программа активизирует стандартное файловое диалоговое окно:

Программа 131. Диалог выбора файла

```
// Файл FileDialogDemo.java
/* Демонстрирует файловое диалоговое окно.
Это приложение,  не апплет. */
import java.awt.*;
import java.awt.event.*;
// Создать подкласс класса Frame
class SampleFrame extends Frame {
   SampleFrame(String title) {
      super(title);
   // Создать объект для обработки window-событий
      MyWindowAdapter adapter = new MyWindowAdapter(this);
   // Регистрировать его для приема этих событий
      addWindowListener(adapter);
   }
}
class MyWindowAdapter extends WindowAdapter {
   SampleFrame SampleFrame;
   public MyWindowAdapter(SampleFrame SampleFrame) {
      this.SampleFrame = SampleFrame;
   }
   public void windowClosing(WindowEvent we) {
      SampleFrame.setVisible(false);
   }
}
// Создать фрейм-окно
class FileDialogDemo {
   public static void main(String args[]) {
      Frame f = new SampleFrame ("File Dialog Demo");
      f.setVisible(true);
      f.setSize(100, 100);
      FileDialog fd = new FileDialog(f, "File Dialog");
      fd.setVisible(true);
   }
}
```

Вывод этой программы представлен на рис. 22.17.

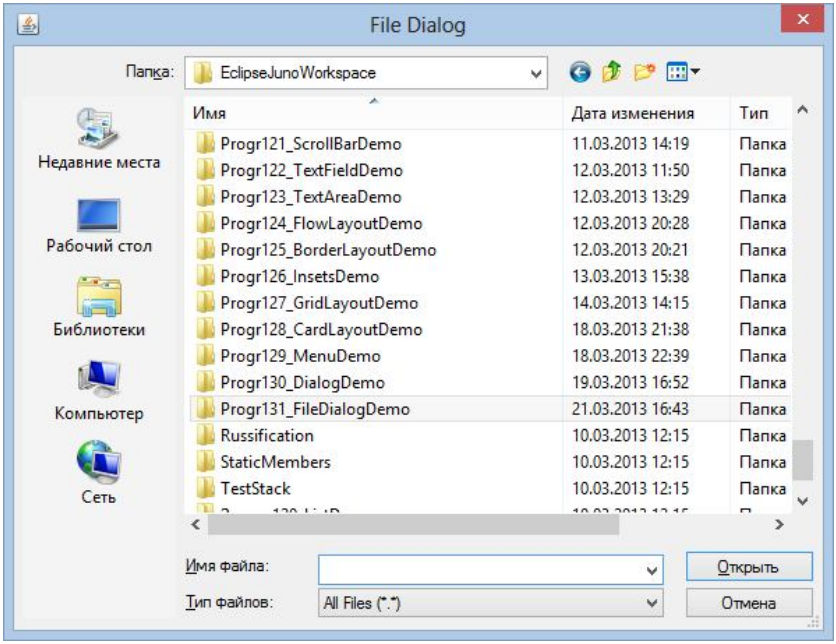

Рис. 17. Диалог выбора файла

Задачи 18-20. Элементы управления

18. В программе 130 создайте диалоговые окна для других пунктов, представленных в меню.

19. Используя текстовые поля, напишите программу-калькулятор, в которой должны быть два текстовых поля для ввода двух чисел и четыре кнопки для выполнения сложения, вычитания, умножения, деления.

20. Дополните программму 131 выводом в окно апплета названия файла, выбранного в диалоговом окне выбора файла.