Программирование на языке Java. Набор абстракций для работы с окнами

Картузов А.В.

Трудность при создании независимой от платформы библиотеки заключается в том, что ее разработчикам либо приходится требовать, чтобы все приложения на всех платформах вели себя и выглядели одинаково, либо для поддержки, скажем, трех различных разновидностей интерфейса приходится писать в три раза больше кода. Существуют два взгляда на эту проблему. Один подход заключается в том, что упор делается на графику низкого уровня—рисование пикселей, при этом разработчики библиотеки сами заботятся о внешнем виде каждого компонента. При другом подходе создаются абстракции, подходящие для библиотек каждой из операционных систем, и именно "родные" пакеты данной операционной системы служат подъемной силой для архитектурно-нейтральной библиотеки на каждой из платформ. В Java при создании библиотеки Abstraction Window Toolkit (AWT) выбран второй подход.

Классы Graphics и Fonts мы уже обсуждали в главе 15. В главе 18 будут обсуждаться различные возможности работы с изображениями. В данной главе мы пройдемся по базовой архитектуре AWT, касающейся интерфейсных объектов.

1. Компоненты

Component—это абстрактный класс, который инкапсулирует все атрибуты визуального интерфейса—обработка ввода с клавиатуры, управление фокусом, взаимодействие с мышью, уведомление о входе/выходе из окна, изменения размеров и положения окон, прорисовка своего собственного графического представления, сохранение текущего текстового шрифта, цветов фона и переднего плана (более 10 методов). Перейдем к

некоторым конкретным подклассам класса Component.

1.1. Container

Container—это абстрактный подкласс класса Component, определяющий дополнительные методы, которые дают возможность помещать в него другие компоненты, что дает возможность построения иерархической системы визуальных объектов. Container отвечает за расположение содержащихся в нем компонентов с помощью интерфейса LayoutManager, описание которого будет позднее в этой главе.

1.2. Panel

Класс Panel—это очень простая специализация класса Container. В отличие от последнего, он не является абстрактным классом. Поэтому о Panel можно думать, как о допускающем рекурсивную вложенность экранном компоненте. С помощью метода add в объекты Panel можно добавлять другие компоненты. После того, как в него добавлены какие-либо компоненты, можно вручную задавать их положение и изменять размер с помощью методов move, resize и reshape класса Component.

В предыдущей главе мы уже использовали один из подклассов Panel—Applet. Каждый раз, когда мы создавали Applet, методы paint и update рисовали его изображение на поверхности объекта Panel. Прежде, чем мы углубимся в методы Panel, давайте познакомимся с компонентом Canvas, который можно вставлять в пустую Panel при работе с объектом Applet.

2. Canvas

Основная идея использования объектов Canvas в том, что они являются семантически свободными компонентами. Вы можете придать объекту Canvas любое поведение и любой желаемый внешний вид. Его имя подразумевает, что этот класс является пустым холстом, на котором вы можете "нарисовать" любой компонент—такой, каким вы его себе представляете.

Произведем от Canvas подкласс GrayCanvas, который будет просто закрашивать себя серым цветом определенной насыщенности. Наш апплет будет создавать несколько

таких объектов, каждый со своей интенсивностью серого цвета.

```
/* <applet code = "PanelDemo"
width=300
height=300>
</applet>
*/
import java.awt.*;
import java.applet.*;
    class GrayCanvas extends Canvas {
        Color gray;
        public GrayCanvas(float g) {
            gray = new Color(g, g, g);
            }
        public void paint(Graphics g) {
            Dimension size = size();
            g.setColor(gray);
            g.fillRect(0, 0, size.width, size.height);
            g.setColor(Color.black);
            g.drawRect(0, 0, size.width-1, size.height-1);
            }
        }
    public class PanelDemo extends Applet {
        static final int n = 4;
        public void init() {
        setLayout(null);
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        for (int i=0;i<n;i++) {
            for (int j=0;j<n;j++) {
                float g=(i*n+j)/(float)(n*n);
                Canvas c=new GrayCanvas(g);
                add(c);
                c.resize(width/n,height/n);
                c.move(i*width/n,j*height/n);
                }
            }
        }
```

```
    }
```

3. Label

Функциональность класса Label сводится к тому, что он знает, как нарисовать объект String—текстовую строку, выровняв ее нужным образом. Шрифт и цвет, которыми отрисовывается строка метки, являются частью базового определения класса Component. Для работы с этими атрибутами предусмотрены пары методов getFont/setFont и getForeground/setForeground. Задать или изменить текст строки после создания объекта с помощью метода setText. Для задания режимов выравнивания в классе Label определены три константы—LEFT, RIGHT и CENTER. Ниже приведен пример, в котором создаются три метки, каждая—со своим режимом выравнивания.

```
/* <applet code = "LabelDemo" width=100 height=100>
   </applet>
*/
import java.awt.*;
import java.applet.*;
    public class LabelDemo extends Applet {
        public void init() {
        setLayout(null);
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        Label left = new Label("Left", Label.LEFT);
        Label right = new Label("Right", Label.RIGHT);
        Label center = new Label("Center", Label.CENTER);
        add(left);
        add(right);
        add(center);
        left.reshape(0, 0, width, height / 3);
        right.reshape(0, height / 3, width, height / 3);
        center.reshape(0, 2 * height / 3, width, height / 3);
        }
    }
```

На этот раз, чтобы одновременно переместить и изменить размер объектов Label, мы использовали метод reshape. Ширина каждой из меток равна полной ширине апплета, высота—1/3 высоты апплета. Вот как этот апплет должен выглядеть, если его

запустить—LabelDemo.html.

4. Button

Объекты-кнопки помечаются строками, причем эти строки нельзя выравнивать подобно строкам объектов Label (они всегда центрируются внутри кнопки). Позднее в данной главе речь пойдет о том, как нужно обрабатывать события, возникающие при нажатии и отпускании пользователем кнопки. Ниже приведен пример, в котором создаются три расположенные по вертикали кнопки.

```
/* <applet code = "ButtonDemo" width=100 height=100>
   </applet>
*/
import java.awt.*;
import java.applet.*;
    public class ButtonDemo extends Applet {
        public void init() {
        setLayout(null);
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        Button yes = new Button("Yes");
        Button no = new Button("No");
        Button maybe = new Button("Undecided");
        add(yes);
        add(no);
        add(maybe);
        yes.reshape(0, 0, width, height / 3);
        no.reshape(0, height / 3, width, height / 3);
        maybe.reshape(0, 2 * height / 3, width, height / 3);
        }
    }
```

5. Checkbox

Класс Checkbox часто используется для выбора одной из двух возможностей. При создании объекта Checkbox ему передается текст метки и логическое значение, чтобы задать исходное состояние окошка с отметкой. Программно можно получать и

устанавливать состояние окошка с отметкой с помощью методов getState и setState. Ниже приведен пример с тремя объектами Checkbox, задаваемое в этом примере исходное состояние соответствует отметке в первом объекте.

```
/* <applet code = "CheckBoxDemo" width=120 height=100>
   </applet>
*/
import java.awt.*;
import java.applet.*;
    public class CheckboxDemo extends Applet {
        public void init() {
        setLayout(null);
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        Checkbox win95 = new Checkbox("Windows 95/98", null, true);
        Checkbox Solaris = new Checkbox("Solaris 2.5");
        Checkbox mac = new Checkbox("MacOS 7.5");
        add(win95);
        add(solaris);
        add(mac);
        win95.reshape(0, 0, width, height / 3);
        Solaris.reshape(0, height / 3, width, height / 3);
        mac.reshape(0, 2 * height / 3, width, height / 3);
        }
    }
```

Ниже приведен внешний вид работающего апплета—CheckBoxDemo.html.

5.1. CheckboxGroup

Второй параметр конструктора Checkbox (в предыдущем примере мы ставили там null) используется для группирования нескольких объектов Checkbox. Для этого сначала создается объект CheckboxGroup, затем он передается в качестве параметра любому количеству конструкторов Checkbox, при этом предоставляемые этой группой варианты выбора становятся взаимоисключающими (только один может быть задействован). Предусмотрены и методы, которые позволяют получить и установить группу, к которой принадлежит конкретный объект Checkbox—getCheckboxGroup и setCheckboxGroup. Вы можете пользоваться методами getCurrent и setCurrent для

получения и установки состояния выбранного в данный момент объекта Checkbox. Ниже приведен пример, отличающийся от предыдущего тем, что теперь различные варианты выбора в нем взаимно исключают друг друга.

```
/* <applet code = "CheckboxGroupDemo" width=120 height=100>
   </applet>
*/
import java.awt.*;
import java.applet.*;
    public class CheckboxGroupDemo extends Applet {
        public void init() {
        setLayout(null);
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        CheckboxGroup g = new CheckboxGroup();
        Checkbox win95 = new Checkbox("Windows 95/98", g, true);
        Checkbox solaris = new Checkbox("Solaris 2.5", g, false);
        Checkbox mac = new Checkbox("MacOS 7.5", g, false);
        add(win95);
        add(solaris);
        add(mac);
        win95.reshape(0, 0, width, height / 3);
        solaris. reshape(0, height / 3, width, height / 3);
        mac.reshape(0, 2 * height / 3, width, height / 3);
        }
    }
```

Обратите внимание—окошки изменили свою форму, теперь они не квадратные, а круглые—CheckboxGroupDemo.html.

6. Choice

Класс Choice (выбор) используется при создании раскрывающихся списочных меню (выпадающих списков типа ComboBox в Windows). Компонент Choice занимает ровно столько места, сколько требуется для отображения выбранного в данный момент элемента, когда пользователь щелкает мышью на нем, раскрывается меню со всеми элементами, в котором можно сделать выбор. Каждый элемент меню—это строка, которая выводится, выровненная по левой границе. Элементы меню выводятся в том

порядке, в котором они были добавлены в объект Choice. Метод countItems возвращает количество пунктов в меню выбора. Вы можете задать пункт, который выбран в данный момент, с помощью метода select, передав ему либо целый индекс (пункты меню перечисляются с нуля), либо строку, которая совпадает с меткой нужного пункта меню. Аналогично, с помощью методов getSelectedItem и getSelectedIndex можно получить, соответственно, строку-метку и индекс выбранного в данный момент пункта меню. Вот очередной простой пример, в котором создается два объекта Choice.

```
/* <applet code = "ChoiceDemo" width=200 height=100>
   </applet>
*/
import java.awt.*;
import java.applet.*;
    public class ChoiceDemo extends Applet {
        public void init() {
        setLayout(null);
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        Choice os = new Choice();
        Choice browser = new Choice();
        os.addItem("Windows 95/98");
        os.addItem("Solaris 2.5");
        os.addItem("MacOS 7.5");
        browser.addItem("Netscape Navigator 3.0");
        browser.addItem("Netscape Communicator 4.5");
        browser.addItem("Internet Explorer 3.0");
        browser.addItem("Mosaic 3.0");
        browser.addItem("Lynx 2.4");
        browser.select("Netscape Communicator 4.5");
        add(os);
        add(browser);
        os.reshape(0, 0, width, height / 2);
        browser.reshape(0, height / 2, width, height / 2);
        }
    }
```

А вот как выглядят эти выпадающие списки—ChoiceDemo.html.

7. List

Класс List представляет собой компактный список с возможностью выбора нескольких вариантов и с прокруткой (аналог ListBox в Windows). Ниже приведен пример с двумя списками выбора, один из которых допускает выбор нескольких элементов, а второй—выбор единственного элемента.

```
/* <applet code = "ListDemo" width=200 height=100>
   </applet>
*/
import java.awt.*;
import java.applet.*;
public class ListDemo extends Applet {
    public void init() { setLayout(null);
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        List os = new List(0, true);
        List browser = new List(0, false);
        os.addItem("Windows 95/98");
        os.addItem("Solaris 2.5");
        os.addItem("MacOS 7.5");
        browser.addItem("Netscape Navigator 3.0");
        browser.addItem("Netscape Communicator 4.5");
        browser.addItem("lnternet Explorer 4.0");
        browser.addItem("Mosaic 3.0");
        browser.addItem("Lynx 2.4");
        browser.select(1);
        add(os);
        add(browser);
        os.reshape(0, 0, width, height / 2);
        browser.reshape(0, height / 2, width, height / 2);
        }
    }
```

Заметьте, что у нижнего списка имеется линейка прокрутки, поскольку все его элементы не уместились в заданный нами размер—ListDemo.html.

8. Scrollbar

Объекты Scrollbar (линейки прокрутки) используются для выбора подмножества значений между заданными минимумом и максимумом. Визуально у линейки прокрутки есть несколько органов управления, ориентированных либо вертикально, либо горизонтально. Стрелки на каждом из ее концов показывают, что, нажав на них, вы можете продвинуться на один шаг в соответствующем направлении. Текущее положение отображается с помощью движка линейки прокрутки, которым пользователь также может управлять, устанавливая требуемое положение линейки.

Конструктор класса Scrollbar позволяет задавать ориентацию линейки прокрутки—для этого предусмотрены константы VERTICAL и HORIZONTAL. Кроме того с помощью конструктора можно задать начальное положение и размер движка, а так же минимальное и максимальное значения, в пределах которых линейка прокрутки может изменять параметр. Для получения и установки текущего состояния линейки прокрутки используются методы getValue и setValue. Кроме того воспользовавшись методами getMinimum и getMaximum, вы можете получить рабочий диапазон объекта. Ниже приведен пример, в котором создается и вертикальная, и горизонтальная линейки прокрутки.

```
/* <applet code = "ScrollbarDemo" width=200 height=100>
   </applet>
*/
import java.awt.*;
import java.applet.*;
public class ScrollbarDemo extends Applet {
    public void init() {
        setLayout(null);
        int width=Integer.parseInt(getParameter("width"));
        int height=Integer. parseInt(getParameter("height"));
        Scrollbar hs=new Scrollbar(Scrollbar.HORIZONTAL,50,width/10,0,100);
        Scrollbar vs=new Scrollbar(Scrollbar.VERTICAL,50,height/2,0,100);
        add(hs);
        add(vs);
        int thickness = 16;
        hs.reshape(0, height-thickness,width-thickness,thickness);
        vs.reshape(width-thickness,0,thickness,height-thickness);
        }
    }
```

В этом примере скроллируется, конечно, пустая область—ScrollbarDemo.html.

9. TextField

Класс TextField представляет собой реализацию однострочной области для ввода текста. Такие области часто используются в формах для пользовательского ввода. Вы можете "заморозить" содержимое объекта TextField с помощью метода setEditable, а метод isEditable сообщит вам, можно ли редактировать текст в данном объекте. Текущее значение объекта можно получить методом getText и установить методом setText. С помощью метода select можно выбрать фрагмент строки, задавая его начало и конец, отсчитываемые с нуля. Для выбора всей строки используется метод selectAll.

Метод setEchoChar задает символ, который будет выводиться вместо любых вводимых символов. Вы можете проверить, находится ли объект TextField в этом режиме, с помощью метода echoCharIsSet, и узнать, какой именно символ задан для эхо-печати, с помощью метода getEchoChar. Вот пример, в котором создаются классические поля для имени пользователя и пароля.

```
/* <applet code = "TextFieldDemo" width=200 height=100>
   </applet>
*/
import java.awt.*;
import java.applet.*;
public class TextFieldDemo extends Applet {
    public void init() {
        setLayout(null);
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        Label namep = new Label("Name : ", Label.RIGHT);
        Label passp = new Label("Password : ", Label.RIGHT);
        TextField name = new Text Field(8);
        TextField pass = new TextField(8);
        pass.setEchoChar('*');
        add(namep);
        add(name);
        add(passp);
        add(pass);
```

```
        int space = 25;
        int w1 = width / 3;
        namep.setBounds(0, (height—space) / 2, w1, space);
        name.setBounds(w1, (height—space) / 2, w1, space);
        passp.setBounds(0, (height + space) / 2, w1, space);
        pass.setBounds(w1, (height + space) / 2, w1, space);
        }
    }
```

Обратите внимание, что в этом примере мы заменили устаревший в JDK 1.1 reshape на setBounds—TextFieldDemo.html. Вообще, в примерах могут встречаться вызовы Deprecated API, за что автор приносит извинения (после выхода Java 1.2, возможно, некоторые устаревшие функции будут действительно удалены, и тогда все примеры будут пересмотрены).

10. TextArea

Порой одной строки текста оказывается недостаточно для конкретной задачи. AWT включает в себя очень простой многострочный редактор обычного текста, называемый TextArea. Конструктор класса TextArea воспринимает значение типа String в качестве начального текста объекта. Кроме того, в конструкторе указывается число колонок и строк текста, которые нужно выводить. Есть три метода, которые позволяют программе модифицировать содержимое объекта TextArea: appendText добавляет параметр типа String в конец буфера; insertText вставляет строку в заданное отсчитываемым от нуля индексом место в буфере; replaceText копирует строку-параметр в буфер, замещая ею текст, хранящийся в буфере между первым и вторым параметрами-смещениями. Ниже приведена программа, создающая объект TextArea и вставляющая в него строку.

```
/* <applet code = "TextAreaDemo" width=200 height=100>
   </applet>
*/
import java.awt.*;
import java.applet.*;
public class TextAreaDemo extends Applet {
    public void init() {
        setLayout(null);
```

```
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        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" +
        "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, 80, 40);
        add(text);
        text.setBounds(0, 0, width, height);
        }
    }
```

11. Layout

Все компоненты, с которыми мы работали до сих пор в этой главе, размещались "вручную". И в каждом примере мы вызывали загадочный метод setLayout(null). Этот вызов запрещал использование предусмотренного по умолчанию механизма управления размещением компонентов. Для решения подобных задач в AWT предусмотрены диспетчеры размещения (layout managers).

11.1. LayoutManager

Каждый класс, реализующий интерфейс LayoutManager, следит за списком компонентов, которые хранятся с именами типа String. Всякий раз, когда вы добавляете компонент в Panel, диспетчер размещения уведомляется об этом. Если требуется изменить размер объекта Panel, то идет обращение к диспетчеру посредством методов minimumLayoutSize и hreferredLayoutSize. В каждом компоненте, который приходится обрабатывать диспетчеру, должны присутствовать реализации методов hreferredSize и minimumSize. Эти методы должны возвращать предпочтительный и минимальный размеры для прорисовки компонента,

соответственно. Диспетчер размещения по возможности будет пытаться удовлетворить эти запросы, в то же время заботясь о целостности всей картины взаимного расположения компонентов.

В Java есть несколько предопределенных классов—диспетчеров размещения, описываемых ниже.

11.2. FlowLayout

Класс FlowLayout реализует простой стиль размещения, при котором компоненты располагаются, начиная с левого верхнего угла, слева направо и сверху вниз. Если в данную строку не помещается очередной компонент, он располагается в левой позиции новой строки. Справа, слева, сверху и снизу компоненты отделяются друг от друга небольшими промежутками. Ширину этого промежутка можно задать в конструкторе FlowLayout. Каждая строка с компонентами выравнивается по левому или правому краю, либо центрируется в зависимости от того, какая из констант LEFT, RIGHT или CENTER была передана конструктору. Режим выравнивания по умолчанию—CENTER, используемая по умолчанию ширина промежутка—5 пикселей.

Ниже приведен пример, в котором в Panel включается несколько компонентов Label. Объект Panel использует FlowLayout с выравниванием RIGHT.

```
/* <applet code = "FlowLayoutDemo" width=200 height=100>
   </applet>
*/
import java.awt.*;
import java.applet.*;
import java.util.*;
public class FlowLayoutDemo extends Applet {
    public void init() {
        setLayout(new FlowLayout(FlowLayout.RIGHT, 10, 3));
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        String val = "Data is not information " +
        "is not knowledge is not wisdom.";
        StringTokenizer st = new StringTokenizer(val);
        while (st.hasMoreTokens()) {
```

```
            add(new Button(st.nextToken()));
            }
        }
    }
```

Необходимо вызвать пример для двух различных размеров—FlowLayoutDemo1.html, FlowLayoutDemo2.html для того, чтобы проиллюстрировать, как объекты Label перетекают из строки в строку, и при этом строки выравниваются по правому краю (или Вы можете изменять размеры окна appletViewer).

11.3. BorderLayout

Класс BorderLayout реализует обычный стиль размещения для окон верхнего уровня, в котором предусмотрено четыре узких компонента фиксированной ширины по краям, и одна большая область в центре, которая может расширяться и сужаться в двух направлениях, занимая все свободное пространство окна. У каждой из этих областей есть строки-имена: String.North, String.South, String.East и String.West соответствуют четырем краям, а Center—центральной области. Ниже приведен пример BorderLayout с компонентом в каждой из названных областей.

```
/* <applet code = "BorderLayoutDemo" width=300 height=200>
   </applet>
*/
import java.awt.*;
import java.applet.*;
import java.util.*;
public class BorderLayoutDemo extends Applet {
    public void init() {
        setLayout(new BorderLayout());
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        add("North", new Button("This is across the top"));
        add("South", new Label("The footer message might go here"));
        add("East", new Button("Left"));
        add("West", new Button("Right"));
        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 rnan.\n\n" +
        "George Bernard Shaw\n\n";
        add("Center", new TextArea(msg));
        }
    }
```

Опять читаем фразу со смыслом (спасибо Бернарду Шоу)—BorderLayoutDemo.html.

11.4. GridLayout

Класс GridLayout размещает компоненты в простой равномерной сетке. Конструктор этого класса позволяет задавать количество строк и столбцов. Ниже приведен пример, в котором GridLayout используется для создания сетки 4x4, 15 квадратов из 16 заполняются кнопками, помеченными соответствующими индексами. Как вы уже, наверное, поняли, это—панель для игры в "пятнашки".

```
/* <applet code = "GridLayoutDemo" width=200 height=200>
   </applet>
*/
import java.awt.*;
import java.applet.*;
public class GridLayoutDemo extends Applet {
    static final int n = 4;
    public void init() {
        setLayout(new GridLayout(n, n));
        setFont(new Font("Helvetica", Font.BOLD, 24));
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        for (int i=0;i<n;i++) {
            for (int j=0;j<n;j++) {
                int k=i*n+j;
                if (k>0)
                    add(new Button(""+k));
                    }
                }
            }
        }
```

11.5. Insets

Класс Insets используется для того, чтобы вставлять в объект Panel границы, напоминающие горизонтальные и вертикальные промежутки между объектами, которые делает диспетчер размещения. Для того, чтобы добиться вставки границ в объект Panel, нужно заместить метод Insets реализацией, возвращающей новый объект Insets с четырьмя целыми значениями, соответствующими ширине верхнего, нижнего, левого и правого краев.

```
public Insets insets() {
    return new Insets(10, 10, 10, 10);
}
```

11.6. CardLayout

Класс CardLayout по своему уникален. Он отличается от других программ управления размещением компонентов тем, что представляет несколько различных вариантов размещения, которые можно сравнить с колодой карт. Колоду можно тасовать так, чтобы в данный момент времени наверху была только одна из карт. Это может быть полезно при создании интерфейсов пользователя, в которых есть необязательные компоненты, включаемые и выключаемые динамически в зависимости от реакции пользователя.

12. Window

Класс Window во многом напоминает Panel за тем исключением, что он создает свое собственное окно верхнего уровня. Большая часть программистов скорее всего будет использовать не непосредственно класс Window, а его подкласс Frame.

13. Frame

Frame—это как раз то, что обычно и считают окном на рабочей поверхности экрана. У объекта Frame есть строка с заголовком, управляющие элементы для изменения размера и линейка меню. Для того, чтобы вывести/спрятать изображение объекта Frame, нужно использовать методы show и hide. Ниже приведен пример апплета,

который показывает объект Frame с содержащимся в нем компонентом TextArea.

```
/* <applet code = "FrameDemo" width=200 height=200>
   </applet>
*/
import java.awt.*;
import java.applet.*;
public class FrameDemo extends Applet {
    public void init() {
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        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" +
        "that there are no obvious deficiencies.\n\n" +
        "C.A.R. Hoare\n\n";
        TextArea text = new TextArea(val, 80, 40);
        Frame f = new Frame("Demo Frame");
        f.setSize(width, height);
        f.add("Center", text);
        f.show();
        }
    }
```

14. Меню

С каждым окном верхнего уровня может быть связана линейка меню. Объект MenuBar может включать в себя несколько объектов Menu. Последние, в свою очередь, содержат в себе список вариантов выбора—объектов MenuItem. Menu—подкласс MenuItem, так что объекты Menu также могут включаться в этот список, что позволяет создавать иерархически вложенные подменю. Вот пример, в котором к окну добавлены несколько вложенных меню.

```
/* <applet code = "MenuDemo" width=200 height=200>
   </applet>
*/
```

```
import java.awt.*;
import java.applet. *;
public class MenuDemo extends Applet {
    public void init() {
        int width = Integer.parseInt(getParameter("width"));
        int height = Integer.parseInt(getParameter("height"));
        Frame f = new Frame("Demo Frame");
        f.setSize(width, height);
        MenuBar mbar = new MenuBar();
        f.setMenuBar(mbar);
        Menu file = new Menu("File");
        file.add(new MenuItem("New... "));
        file.add(new MenuItem("0pen..."));
        file.add(new MenuItem("Close"));
        file.add(new MenuItem("-"));
        file.add(new MenuItem("Quit..."));
        mbar.add(file);
        Menu edit = new Menu("Edit");
        edit.add(new MenuItem("Cut"));
        edit.add(new MenuItem("Copy"));
        edit.add(new MenuItem("Paste"));
        edit.add(new MenuItem("-"));
        Menu sub = new Menu("Special");
        sub.add(new MenuItem("First"));
        sub.add(new MenuItem("Second"));
        sub.add(new MenuItem("Third"));
        edit.add(sub);
        edit.add(new CheckBoxMenuItem("Debug"));
        edit.add(new CheckBoxMenuItem("Testing"));
        mbar.add(edit);
        f.show();
        }
}
```

15. AWT при свете дня

AWT в своем нынешнем виде делает прекрасную работу, являясь общим знаменателем—библиотекой, единой для всех платформ. Некоторый недостаток AWT в том, что, поскольку каждый из AWT-компонентов реализован на основе

соответствующего компонента базовой операционной системы, их поведение и внешний вид может меняться при смене платформы. Хорошо известны расширения и аналоги AWT—Swing, Java Foundation Classes (Netscape), Application Foundation Classes (Microsoft).