

Аналогично классы из пакета Swing используют для вывода графических изображений окна **JPanel** и **JFrame**. Еще один используемый для вывода класс **Canvas**, представляющий пустое окно, в котором можно рисовать, является наследником класса **Component**.

Апплеты

Графические интерфейсы, рисунки и изображения могут быть реализованы в апплетах. Апплеты представляют собой разновидность графических приложений, реализованных в виде классов языка Java, которые размещаются на серверах Internet, транспортируются клиенту по сети, автоматически устанавливаются и запускаются браузером как часть документа HTML. Апплеты позволяют вставлять в документы поля, содержание которых меняется во времени, организовывать "бегущие строки", меню, мультипликацию, производить вычисления на клиентской странице. Апплеты выполняются браузером при отображении HTML-документа или просматриваются программой `appletviewer`. Апплеты не могут записывать файлы и читать информацию из файлов. Эти ограничения связаны с проблемой безопасности клиентского компьютера, поскольку клиенты не желают запускать «троянского коня» в свой компьютер. Существует несколько уровней безопасности, устанавливаемых клиентом для апплетов, загружаемых с сервера (надежные апплеты). Класс **Applet** обеспечивает интерфейс между апплетами и их окружением. Апплеты являются наследниками класса **Applet** из пакета **java.applet** из пакета AWT или его подкласса **JApplet** из пакета Swing.

Есть несколько методов класса **Applet**, которые управляют созданием и выполнением апплета на Web-странице. Апплету не нужен метод `main()`, код запуска помещается в методе `init()`. Перегружаемый метод `init()` автоматически вызывается при загрузке апплета для выполнения начальной инициализации. Метод `start()` вызывается каждый раз, когда апплет переносится в поле зрения браузера, чтобы начать операции. Метод `stop()` вызывается каждый раз, когда апплет выходит из поля зрения Web-браузера, чтобы позволить апплету завершить операции. Метод `destroy()` вызывается, когда апплет начинает выгружаться со страницы для выполнения финального освобождения ресурсов. Кроме этих методов, при выполнении апплета автоматически запускается метод `paint()` класса **Component**. Метод `paint()` не вызывается явно, а только из других методов, например из метода `repaint()`, если необходима перерисовка.

Ниже приведен пример апплета, в котором используются методы `init()`, `paint()`, метод `setColor()` установки цвета символов и метод `drawString()` рисования строк.

```
/* пример # 1 : вывод даты : DateApplet.java */
package chapt11;
import java.awt.Color;
import java.awt.Graphics;
import java.util.Calendar;
import java.util.Formatter;
import javax.swing.JApplet;

public class DateApplet extends JApplet {
```

```

private Formatter dateFmt = new Formatter();
private Formatter timeFmt = new Formatter();

public void init() {
    setSize(180, 100);
    Calendar c = Calendar.getInstance();
    String era = "";
    if (c.get(Calendar.ERA) == 1)
        era = "н.э.";
    dateFmt.format("%tA %td.%tm.%tY года "
                  + era, c, c, c, c);
    timeFmt.format("%tT", c);
}
public void paint(Graphics g) {
    g.setColor(Color.RED);
    g.drawString("Апплет стартовал в " + timeFmt,
                10, getHeight()/2);
    g.setColor(new Color(0,87,127));
    g.drawString(dateFmt.toString(), 13,
                getHeight() - 10);
}
}

```

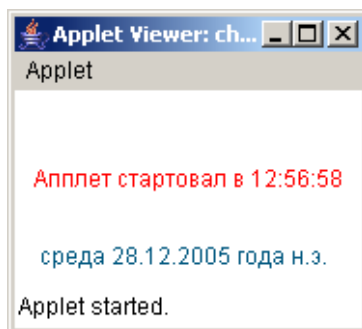


Рис. 11.2. Вывод строки и даты

Доступ к элементам даты осуществлен с помощью класса `java.util.Calendar`. Цвет выводимых символов устанавливается с помощью полей класса `Color`.

После выполнения компиляции имя класса, содержащего байт-код апплета, помещается в тег `<applet параметры> </applet>` документа HTML. Например:

```

<html>
<applet code = chapt11.DateApplet.class
        width = 250 height = 250> </applet></html>

```

Исполнителем HTML-документа является браузер, который и запускает соответствующий ссылке апплет.

Большинство используемых в апплетах графических методов, как и использованные в примере методы `setColor()`, `drawString()`, – методы абстракт-

ного базового класса `java.awt.Graphics`. Класс `Graphics` представляет графический контекст для рисования, который затем отображается на физическое устройство. Методы апплета получают объект класса `Graphics` (графический контекст) в качестве параметра и вместе с ним – текущий цвет, шрифт, положение курсора. Установку контекста обычно осуществляют методы `update()` или `paint()`.

Ниже перечислены некоторые методы класса `Graphics`:

- `drawLine(int x1, int y1, int x2, int y2)` – рисует линию;
- `drawRect(int x, int y, int width, int height)` и `fillRect(int x, int y, int width, int height)` – рисуют прямоугольник и заполненный прямоугольник;
- `draw3DRect(int x, int y, int width, int height, boolean raised)` – рисует трехмерный прямоугольник;
- `drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)` – рисует округленный прямоугольник;
- `drawOval(int x, int y, int width, int height)` – рисует овал;
- `drawPolygon(int[] xPoints, int[] yPoints, int nPoints)` – рисует полигон (многоугольник), заданный массивами координат `x` и `y`;
- `drawPolygon(Polygon p)` – рисует полигон, заданный объектом `Polygon`;
- `drawPolyline(int[] xPoints, int[] yPoints, int nPoints)` – рисует последовательность связанных линий, заданных массивами `x` и `y`;
- `drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)` – рисует дугу окружности;
- `drawImage(Image img, int x, int y, Color bgcolor, ImageObserver observer)` – вставляет изображение;
- `drawString(String str, int x, int y)` – рисует строку;
- `setColor(Color c), getColor()` – устанавливает и возвращает текущий цвет;
- `getFont()` – возвращает текущий шрифт;
- `setFont(Font font)` – устанавливает новый шрифт.

Методы класса `Graphics` используются для отображения графики как для классов `Applet`, так и для классов `JApplet`.

В примерах 2–4, приведенных ниже, демонстрируется использование методов класса `Graphics` для вывода графических изображений в окно апплета.

/ пример # 2 : трехмерные прямоугольники : ThrRect.java */*

```
package chapt11;
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JApplet;

public class ThrRect extends JApplet {
    public void draw3D(Graphics g, int x, int y, int
width, int height, boolean isRaised, boolean isFilled) {
        g.draw3DRect(x, y, width - 1, height - 1,
            isRaised);
        g.draw3DRect(x + 1, y + 1, width - 3,
            height - 3, isRaised);
    }
}
```

```

        g.draw3DRect(x + 2, y + 2, width - 5,
                     height - 5, isRaised);
        if (isFilled)
            g.fillRect(x + 3, y + 3, width - 6,
                      height - 6);
    }
    public void paint(Graphics g) {
        g.setColor(Color.GRAY);
        draw3D(g, 10, 5, 80, 40, true, false);
        draw3D(g, 130, 5, 80, 40, false, false);
        draw3D(g, 10, 55, 80, 40, true, true);
        draw3D(g, 130, 55, 80, 40, false, true);
    }
}

```

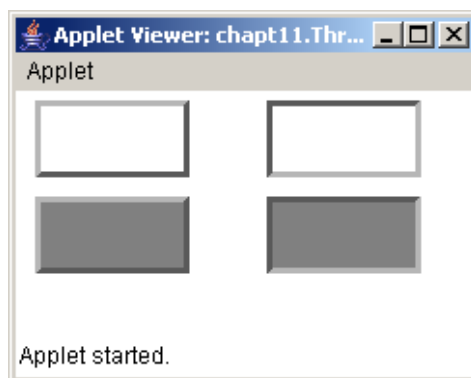


Рис. 11.3. Трехмерные прямоугольники

Пакет **java.awt** содержит большое число классов, используемых для вывода изображения: **Color**, **Font**, **Image**, **Shape**, **Canvas** и т.д. Кроме того, возможности этого пакета расширяют пакеты **java.awt.geom**, **java.awt.color**, **java.awt.image** и другие.

/ пример # 3 : построение фигур : BuildShape.java */*

```

package chapt11;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Shape;
import java.awt.geom.*;
import javax.swing.JApplet;

public class BuildShape extends JApplet {
    public void init() {
        setSize(200, 205);
    }
    public void paint(Graphics g) {
        Graphics2D g2 = (Graphics2D) g;
        setBackground(Color.LIGHT_GRAY);
    }
}

```

```

        g2.rotate(Math.PI / 6);
        drawChessBoard(g);
//поворот
        g2.rotate(-Math.PI / 6);
        g.setXORMode(new Color(200, 255, 250));
        Shape e = new Ellipse2D.Float(70, 75, 70, 50);
//рисование эллипса
        g2.fill(e);
    }
//рисование шахматной доски
    public void drawChessBoard(Graphics g) {
        int size = 16;
        for (int y = 0; y < 8; y++) {
            for (int x = 0; x < 8; x++) {
                if ((x + y) % 2 == 0)
                    g.setColor(Color.BLUE);
                else
                    g.setColor(Color.WHITE);
                g.fillRect(75 + x * size, y * size - 25, size, size);
            }
            g.setColor(Color.BLACK);

            g.drawString(new Character(
(char) ('8' - y)).toString(), 66, y * size - 13);
            g.drawString(new Character(
(char) (y + 'a')).toString(),
                79 + y * size, 8 * size - 14);
        }
    }
}

```

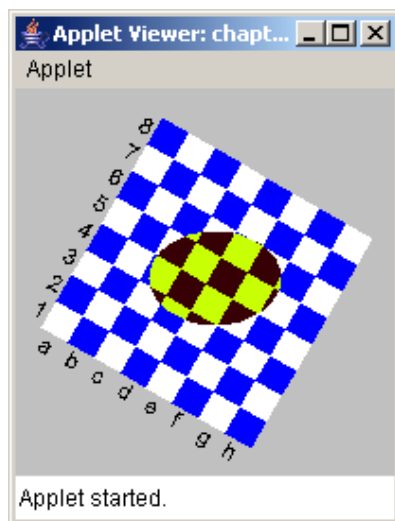


Рис. 11.4. Построение различных фигур

// пример # 4 : вывод GIF-изображения : DrawImage.java

```
package chapt11;
import java.awt.Graphics;
import java.awt.Image;
import javax.swing.JApplet;

public class DrawImage extends JApplet {
    private Image img;

    public void init() {
        //загрузка изображения из корня проекта
        img = getImage(getCodeBase(), "joli.gif");
    }
    public void paint(Graphics g){
        g.drawImage(img, 0, 0, this);
    }
}
```

При использовании свойств тега **<applet>** существует возможность передать параметры из HTML-документа в код апплета. Пусть HTML-документ имеет вид:

```
<html><head><title>Параметры апплета</title></head>
<body>
<applet code=test.com.ReadParam.class
        width=250 height=300>
    <param name = bNumber value = 4>
    <param name = state value = true>
</applet></body> </html>
```

Тогда для чтения и обработки параметров **bNumber** и **state** апплет должен выглядеть следующим образом:

/ пример # 5 : передача параметров апплету : ReadParam.java */*

```
package chapt11;
import java.awt.Color;
import java.awt.Graphics;
import java.applet.Applet;

public class ReadParam extends Applet {
    private int bNum;
    private boolean state;

    public void start() {//чтение параметров
        String param;
        param = getParameter("state");
        if(param != null)
            state = new Boolean(param);
        try {
            param = getParameter("bNumber");
            if(param != null)
                bNum = Integer.parseInt(param);
        }
```

```

    } catch (NumberFormatException e) {
        bNum = 0;
        state = false;
    }
}
public void paint(Graphics g) {
    double d = 0;
    if (state) d = Math.pow(bNum, 2);
    else g.drawString("Error Parameter", 0, 11);
    g.drawString("Statement: " + state, 0, 28);
    g.drawString("Value b: " + bNum, 0, 45);
    g.drawString("b power 2: " + d, 0, 62);
}
}

```

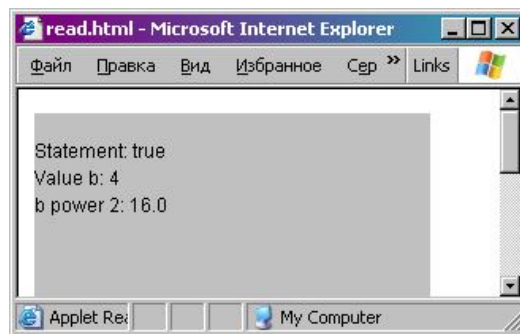


Рис. 11.5. Передача параметров в апплет

Если параметр недоступен, метод `getParameter()` возвращает `null`.

При применении управляющих компонентов в апплетах принято использовать классы из пакета `javax.swing`, в которых компонент на экране создается средствами Java и в минимальной степени зависит от платформы и оборудования. Такими классами-компонентами являются `JButton`, `JCheckBox`, `JDialog`, `JMenu`, `JComboBox`, `JMenuItem`, `JTextField`, `JTextArea` и другие. Компоненты, как правило, размещают в контейнер (класс `Container`), являющийся неявным объектом любого графического приложения. Явную ссылку на контейнер можно получить с помощью метода `getContentPane()`.

// пример # 6 : апплет с компонентом : *MyJApplet.java*

```

package chapt11;
import java.awt.Container;
import javax.swing.JApplet;
import javax.swing.JLabel;

public class MyJApplet extends JApplet {
    private JLabel lbl = new JLabel("Swing-applet!");

    public void init() {
        Container c = getContentPane();
        c.add(lbl);
    }
}

```

```

}

```

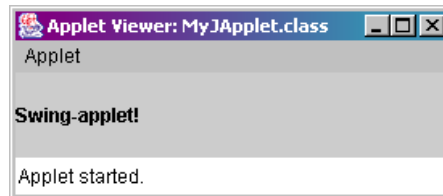


Рис. 11.6. Апплет с меткой

В этой программе производится помещение текстовой метки **JLabel** на форму в апплете. Конструктор класса **JLabel** принимает объект **String** и использует его значение для создания метки. Автоматически вызываемый при загрузке апплета метод **init()** обычно отвечает за инициализацию полей и размещение компонента на форму. Для этого вызывается метод **add()** класса **Container**, который помещает компонент в контейнер. Метод **add()** сразу не вызывается, как можно сделать в библиотеке AWT. Пакет **Swing** требует, чтобы все компоненты добавлялись в "панель содержания" формы **ContentPane**, так что требуется сначала вызывать метод **getContentPane()** класса **JApplet** для создания ссылки на объект, как часть процесса **add()**.

// пример # 7 : жизненный цикл апплета : DemoLC.java

```

package chapt11;
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JApplet;

public class DemoLC extends JApplet {
    private int starX[] =
    { 112, 87, 6, 71, 47, 112, 176, 151, 215, 136 };

    private int starY[] =
    { 0, 76, 76, 124, 200, 152, 200, 124, 76, 76 };

    private int i;
    private Color c;

    public void init() {
        c = new Color(0, 0, 255);
        setBackground(Color.LIGHT_GRAY);
        i = 1;
    }

    public void start() {
        int j = i * 25;
        if (j < 255)
            c = new Color(j, j, 255 - j);
        else i = 1;
    }

    public void paint(Graphics g) {
        g.setColor(c);
    }
}

```



```

        g.fillPolygon(starX, starY, starX.length);
        g.setColor(Color.BLACK);
        g.drawPolygon(starX, starY, starX.length);
    }
    public void stop() {
        i++;
    }
}

```

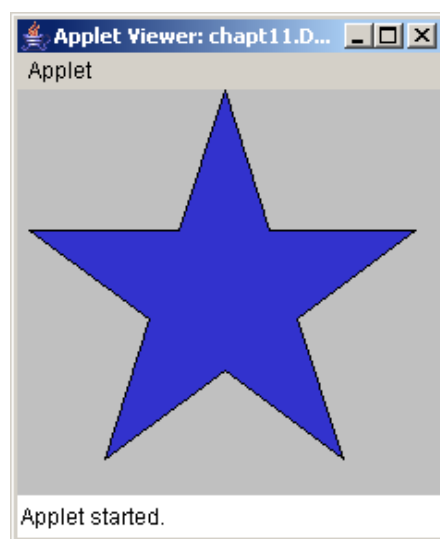


Рис. 11.7. Цвет полигона и жизненный цикл апплета

При работе со шрифтами можно узнать, какие из них доступны на компьютере, и использовать их. Для получения этой информации применяется метод **getAvailableFontFamilyNames()** класса **GraphicsEnvironment**. Метод возвращает массив строк, содержащий имена всех доступных семейств шрифтов, зарегистрированных на данном компьютере.

// пример #8 : доступ к шрифтам ОС : Fonts.java

```

package chapt11;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.GraphicsEnvironment;
import javax.swing.JApplet;

public class Fonts extends JApplet {
    private String[] fonts;

    public void init() {
        GraphicsEnvironment ge =
GraphicsEnvironment.getLocalGraphicsEnvironment();

        fonts = ge.getAvailableFontFamilyNames();
        setSize(700, 400);
    }
}

```

```

    }
    public void paint(Graphics g) {
        int xSize = getWidth() / 170;
        for (int i = 0; i < fonts.length; i++) {
            g.setFont(new Font(
fonts[i], Font.PLAIN, 12)); //название, стиль, размер
            g.drawString(fonts[i],
                170 * (i % xSize), 13 * (i / xSize + 1));
        }
    }
}

```



Рис. 11.8. Вывод списка шрифтов

Фреймы

В Java окно верхнего уровня (не содержащееся в другом окне) называется *фреймом*. В отличие от апплетов графические приложения, расширяющие класс `java.awt.Frame` или его подкласс `javax.swing.JFrame`, не нуждаются в браузере. Для создания графического интерфейса приложения необходимо предоставить ему в качестве окна для вывода объект `Frame` или `JFrame`, в который будут помещаться используемые приложением компоненты GUI (Graphics User Interface). Большинство своих методов класс `Frame` наследует по иерархии от классов `Component`, `Container` и `Window`. Класс `JFrame` из библиотеки Swing является подклассом класса `Frame`.

Такое приложение запускается с помощью метода `main()` и само отвечает за свое отображение в окне `Frame` или `JFrame`.

```

/* пример # 9 :простейший фрейм – овалы и дуги : FrameDemo.java */
package chapt11;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;

```