Программирование на языке Java. Апплеты

Картузов А.В.

Апплеты—это маленькие приложения, которые размещаются на серверах Internet, транспортируются клиенту по сети, автоматически устанавливаются и запускаются на месте, как часть документа HTML. Когда апплет прибывает к клиенту, его доступ к ресурсам ограничен.

Ниже приведен исходный код канонической программы HelloWorld, оформленной в виде апплета:

```
import java.awt.*;
import java.applet.*;
        public class HelloWorldApplet extends Applet {
                public void paint(Graphics g) {
                g.drawString("Hello World!", 20, 20);
                }
        }
```

Этот апплет начинается двумя строками, которые импортируют все пакеты иерархий java.applet и java.awt. Дальше в нашем примере присутствует метод paint, замещающий одноименный метод класса Applet. При вызове этого метода ему передается аргумент, содержащий ссылку на объект класса Graphics. Последний используется для прорисовки нашего апплета. С помощью метода drawString, вызываемого с этим объектом типа Graphics, в позиции экрана (20,20) выводится строка "Hello World".

Для того, чтобы с помощью браузера запустить этот апплет, нам придется написать несколько строк html-текста.

```
        <applet code="HelloWorldApplet" width=200 height=40>
        </applet>
```

Вы можете поместить эти строки в отдельный html-файл (HelloWorldApplet.html), либо вставить их в текст этой программы в виде комментария и запустить программу appletviewer с его исходным текстом в качестве аргумента.

1. Тег HTML <Applet>

Тег &ltapplet> используется для запуска апплета как из HTML-документа, так и из программы appletviewer. Программа appletviewer выполняет каждый найденный ей тег <applet> в отдельном окне, в то время как браузеры позволяют разместить на одной странице несколько апплетов. Синтаксис тэга <APPLET> в настоящее время таков :

```
<APPLET
        CODE = appletFile
        OBJECT = appletSerialFile
        WIDTH = pixels
        HEIGHT = pixels
        [ARCHIVE = jarFiles]
        [CODEBASE = codebaseURL]
        [ALT = alternateText]
        [NAME = appletInstanceName]
        [ALIGN = alignment]
        [VSPACE = pixels]
        [HSPACE = pixels]
        >
                [< PARAM NAME = AttributeNamel VALUE = AttributeValuel >]
                [< PARAM NAME = AttributeName2 VALUE = AttributeValue2 >]
                [HTML-текст, отображаемый при отсутствии поддержки Java]
        </APPLET>
```

1.1. CODE = appletClassFile

CODE—обязательный атрибут, задающий имя файла, в котором содержится оттранслированный код апплета. Имя файла задается относительно codebase, то есть либо от текущего каталога, либо от каталога, указанного в атрибуте CODEBASE. В Java 1.1 вместо этого атрибута может использоваться атрибут OBJECT.

1.2. OBJECT = appletClassSerialFile

Указывает имя файла, содержащего сериализованный апплет, из которого последний будет восстановлен. При запуске определяемого таким образом апплета должен вызываться не метод init(), а метод start(). Для апплета необходимо задать либо атрибут CODE, либо атрибут OBJECT, но задавать эти атрибуты одновременно нельзя.

1.3. WIDTH = pixels

1.4. HEIGHT = pixels

WIDTH и HEIGHT—обязательные атрибуты, задающие начальный размер видимой области апплета.

1.5. ARCHIVE = jarFiles

Задает список jar-файлов (разделяется запятыми), которые предварительно загружаются в Web-броузер. В этих архивных файлах могут содержаться файлы классов, изображения, звуки и любые другие ресурсы, необходимые апплету. Для создания архивов используется утилита JAR, синтаксис вызова которой напоминает вызов команды TAR Unix:

```
c:\> jar cf soundmap.jar *.class image.gif sound.wav
```

Очевидно, что передача сжатых jar-файлов повышает эффективность работы. Поэтому многие средства разработки (Lotus JavaBeans, Borland JBuilder) уже имеют средства для публикации апплетов в виде jar-файлов.

1.6. CODEBASE = codebaseURL

CODEBASE—необязательный атрибут, задающий базовый URL кода апплета, являющийся каталогом, в котором будет выполняться поиск исполняемого файла апплета (задаваемого в признаке CODE). Если этот атрибут не задан, по умолчанию используется каталог данного HTML-документа. CODEBASE не обязательно должен указывать на тот же узел, с которого был загружен HTML-документ.

1.7. ALT = alternateAppletText

Признак ALT—необязательный атрибут, задающий короткое текстовое сообщение, которое должно быть выведено в том случае, если используемый браузер распознает синтаксис тега <applet>, но выполнять апплеты не умеет. Это не то же самое, что HTML-текст, который можно вставлять между <applet> и </applet> для браузеров, вообще не поддерживающих апплетов.

1.8. NAME = appletInstanceName

NAME—необязательный атрибут, используемый для задания имени для данного экземпляра апплета. Присвоение апплетам имен необходимо для того, чтобы другие апплеты на этой же странице могли находить их и общаться с ними. Для того, чтобы получить доступ к подклассу MyApplet класса Applet с именем "Duke", нужно написать:

```
MyApplet a = getAppletContext().getApplet("Duke");
```

После того, как вы получили таким образом дескриптор именованного экземпляра апплета, вы можете вызывать его методы точно так же, как это делается с любым другим объектом.

1.9. ALIGN = alignment

ALIGN—необязательный атрибут, задающий стиль выравнивания апплета. Этот атрибут трактуется так же, как в теге IMG, возможные его значения—LEFT, RIGHT, TOP, TEXTTOP, MIDDLE, ABSMIDDLE, BASELINE, BOTTOM, ABSBOTTOM.

1.10. VSPACE = pixels

HSPACE = PIXELS

Эти необязательные атрибуты задают ширину свободного пространства в пикселях сверху и снизу апплета (VSPACE), и слева и справа от него (HSPACE). Они трактуются точно так же, как одноименные атрибуты тега IMG.

1.11. PARAM NAME = appletAttribute1 VALUE = value1

Этот тег дает возможность передавать из HTML-страницы апплету необходимые ему аргументы. Апплеты получают эти атрибуты, вызывая метод getParameter(), описываемый ниже.

2. Передача параметров

```
getParameter(String)
```

Метод getParameter возвращает значение типа String, соответствующее указанному имени параметра. Если вам в качестве параметра требуется значение какого-либо другого типа, вы должны преобразовать строку-параметр самостоятельно. Вы сейчас увидите некоторые примеры использования метода getParameter для извлечения параметров из приведенного ниже примера:

```
<applet code=Testing width=40 height=40>
<param name=fontName value=Univers>
<param name=fontSize value=14>
<param name=leading value=2>
<param name=accountEnabled value=true>
```

Ниже показано, как извлекается каждый из этих параметров:

```
String FontName = getParameter("fontName");
String FontSize = Integer.parseInt(getParameter("fontSize"));
String Leading = Float.valueOf(getParameter("leading"));
String PaidUp = Boolean.valueOf(getParameter("accountEnabled"));
```

3. Контекст апплета

3.1. getDocumentBase и getCodeBase

Возможно, Вы будете писать апплеты, которым понадобится явно загружать данные и текст. Java позволяет апплету загружать данные из каталога, в котором располагается HTML-документ, запустивший апплет (база документа—getDocumentBase), и из каталога, из которого был загружен class-файл с кодом апплета (база кода—getCodeBase).

3.2. AppletContext и showDocument

AppletContext представляет собой средства, позволяющие получать информацию об окружении работающего апплета. Метод showDocument приводит к тому, что заданный его параметром документ отображается в главном окне браузера или фрейме.

4. Отладочная печать

Отладочную печать можно выводить в два места: на консоль и в статусную строку программы просмотра апплетов. Для того, чтобы вывести сообщение на консоль, надо написать:

```
        System.out.println("Hello there, welcome to Java");
```

Сообщения на консоли очень удобны, поскольку консоль обычно не видна пользователям апплета, и в ней достаточно места для нескольких сообщений. В браузере Netscape консоль Java доступна из меню Options, пункт "Show Java Console".

Метод showStatus выводит текст в статусной области программы appletviewer или браузера с поддержкой Java. В статусной области можно вывести только одну строку сообщения.

5. Порядок инициализации апплета

Ниже приведен порядок, в котором вызываются методы класса Applet, с пояснениями, нужно или нет переопределять данный метод.

5.1. init

Метод init вызывается первым. В нем вы должны инициализировать свои переменные.

5.2. start

Метод start вызывается сразу же после метода init. Он также используется в качестве стартовой точки для возобновления работы после того, как апплет был остановлен. В

то время, как метод init вызывается только однажды—при загрузке апплета, start вызывается каждый раз при выводе HTML-документа, содержащего апплет, на экран. Так, например, если пользователь перейдет к новой WWW-странице, а затем вернется назад к странице с апплетом, апплет продолжит работу с метода start.

5.3. paint

Метод paint вызывается каждый раз при повреждении апплета. AWT следит за состоянием окон в системе и замечает такие случаи, как, например, перекрытие окна апплета другим окном. В таких случаях, после того, как апплет снова оказывается видимым, для восстановления его изображения вызывается метод paint.

5.4. update

Используемый по умолчанию метод update класса Applet сначала закрашивает апплет цветом фона по умолчанию, после чего вызывает метод paint. Если вы в методе paint заполняете фон другим цветом, пользователь будет видеть вспышку цвета по умолчанию при каждом вызове метода update—то есть, всякий раз, когда вы перерисовываете апплет. Чтобы избежать этого, нужно заместить метод update. В общем случае нужно выполнять операции рисования в методе update, а в методе paint, к которому будет обращаться AWT, просто вызвать update.

5.5. stop

Метод stop вызывается в тот момент, когда браузер покидает HTML-документ, содержащий апплет. При вызове метода stop апплет еще работает. Вы должны использовать этот метод для приостановки тех подпроцессов, работа которых необязательна при невидимом апплете. После того, как пользователь снова обратится к этой странице, вы должны будете возобновить их работу в методе start.

5.6. destroy

Метод destroy вызывается тогда, когда среда (например, браузер Netscape) решает, что апплет нужно полностью удалить из памяти. В этом методе нужно освободить все ресурсы, которые использовал апплет.

6. Перерисовка

Возвратимся к апплету HelloWorldApplet. В нем мы заместили метод paint, что позволило апплету выполнить отрисовку. В классе Applet предусмотрены дополнительные методы рисования, позволяющие эффективно закрашивать части экрана. При разработке первых апплетов порой непросто понять, почему метод update никогда не вызывается. Для инициации update предусмотрены три варианта метода repaint.

6.1. repaint

Метод repaint используется для принудительного перерисовывания апплета. Этот метод, в свою очередь, вызывает метод update. Однако, если ваша система медленная или сильно загружена, метод update может и не вызваться. Близкие по времени запросы на перерисовку могут объединяться AWT, так что метод update может вызываться спорадически. Если вы хотите добиться ритмичной смены кадров изображения, воспользуйтесь методом repaint(time)—это позволит уменьшить количество кадров, нарисованных не вовремя.

```
        repaint(time)
```

Вы можете вызывать метод repaint, устанавливая крайний срок для перерисовки (этот период задается в миллисекундах относительно времени вызова repaint).

```
        repaint(x, y, w, h)
```

Эта версия ограничивает обновление экрана заданным прямоугольником, изменены будут только те части экрана, которые в нем находятся.

```
        repaint(time, x, y, w, h)
```

Этот метод—комбинация двух предыдущих.

6.2. Задание размеров графических изображений.

Графические изображения вычерчиваются в стандартной для компьютерной графики

системе координат, в которой координаты могут принимать только целые значения, а оси направлены слева направо и сверху вниз. У апплетов и изображений есть метод size, который возвращает объект Dimension. Получив объект Dimension, вы можете получить и значения его переменных width и height:

```
        Dimension d = size();
        System.out.println(d. width + "," + d.height);
```

7. Простые методы класса Graphics

У объектов класса Graphics есть несколько простых функций рисования. Каждую из фигур можно нарисовать заполненной, либо прорисовать только ее границы. Каждый из методов drawRect, drawOval, fillRect и fillOval вызывается с четырьмя параметрами: int x, int y, int width и int height. Координаты x и y задают положение верхнего левого угла фигуры, параметры width и height определяют ее границы.

7.1. drawLine

```
        drawline(int x1, int y1, int x2, int y2)
```

Этот метод вычерчивает отрезок прямой между точками с координатами (x1,y1) и (x2,y2). Эти линии представляют собой простые прямые толщиной в 1 пиксель. Поддержка разных перьев и разных толщин линий не предусмотрена.

7.2. drawArc и fillArc

Форма методов drawArc и fillArc следующая:

```
        drawArc(int x, int y, int width, int height, int startAngle, int sweepAngle)
```

Эти методы вычерчивают (fillArc заполняет) дугу, ограниченную прямоугольником (x,y,width, height), начинающуюся с угла startAngle и имеющую угловой размер sweepAngle. Ноль градусов соответствует положению часовой стрелки на 3 часа, угол отсчитывается против часовой стрелки (например, 90 градусов соответствуют 12 часам, 180—9 часам, и так далее).

7.3. drawPolyson и fillPolyson

Прототипы для этих методов:

```
drawPolygon(int[], int[], int)
fillPolygon(int[], int[], int)
```

Метод drawPolygon рисует контур многоугольника (ломаную линию), задаваемого двумя массивами, содержащими x и y координаты вершин, третий параметр метода—число пар координат. Метод drawPolygon не замыкает автоматически вычерчиваемый контур. Для того, чтобы прямоугольник получился замкнутым, координаты первой и последней точек должны совпадать.

8. Цвет

Цветовая система AWT разрабатывалась так, чтобы была возможность работы со всеми цветами. После того, как цвет задан, Java отыскивает в диапазоне цветов дисплея тот, который ему больше всего соответствует. Вы можете запрашивать цвета в той семантике, к которой привыкли—как смесь красного, зеленого и голубого, либо как комбинацию оттенка, насыщенности и яркости. Вы можете использовать статические переменные класса Color.black для задания какого-либо из общеупотребительных цветов—black, white, red, green, blue, cyan, yellow, magenta, orange, pink, gray, darkGray и lightGray.

Для создания нового цвета используется один из трех описанных ниже конструкторов.

```
Color(int, int, int)
```

Параметрами для этого конструктора являются три целых числа в диапазоне от 0 до 255 для красного, зеленого и голубого компонентов цвета.

```
Color(int)
```

У этого конструктора—один целочисленный аргумент, в котором в упакованном виде заданы красный, зеленый и голубой компоненты цвета. Красный занимает биты 16-23, зеленый—8-15, голубой—0-7.

```
Color(float, float, float)
```

Последний из конструкторов цвета, Color(float, float, float), принимает в качестве параметров три значения типа float (в диапазоне от 0.0 до 1.0) для красного, зеленого и голубого базовых цветов.

8.1. Методы класса Color

```
HSBtoRGB(float, float, float)
RGBtoHSB(int, int, int, float[1)
```

HSBtoRGB преобразует цвет, заданный оттенком, насыщенностью и яркостью (HSB), в целое число в формате RGB, готовое для использования в качестве параметра конструктора Color(int). RGBtoHSB преобразует цвет, заданный тремя базовыми компонентами, в массив типа float со значениями HSB, соответствующими данному цвету.

Цветовая модель HSB (Hue-Saturation-Brightness, оттенок-насыщенность-яркость) является альтернативой модели Red-Green-Blue для задания цветов. В этой модели оттенки можно представить как круг с различными цветами (оттенок может принимать значения от 0.0 до 1.0, цвета на этом круге идут в том же порядке, что и в радуге—красный, оранжевый, желтый, зеленый, голубой, синий, фиолетовый). Насыщенность (значение в диапазоне от 0.0 до 1.0)—это шкала глубины цвета, от легкой пастели до сочных цветов. Яркость—это также число в диапазоне от 0.0 до 1.0, причем меньшие значения соответствуют более темным цветам, а большие—более ярким.

```
getRedQ, getGreenO, setBlue()
```

Каждый из этих методов возвращает в младших восьми битах результата значение соответствующего базового компонента цвета.

```
getRGB()
```

Этот метод возвращает целое число, в котором упакованы значения базовых компонентов цвета, причем

```
red = Oxff & (getRGB() >> 16);
```

```
        green = Oxff & (getRGB() >> 8);
        blue = Oxff & getRGB();
```

8.2. setPaintMode() и setXORMode(Color)

Режим отрисовки paint—используемый по умолчанию метод заполнения графических изображений, при котором цвет пикселей изменяется на заданный. XOR устанавливает режим рисования, когда результирующий цвет получается выполнением операции XOR (исключающее или) для текущего и указанного цветов (особенно полезно для анимации).

9. Шрифты

Библиотека AWT обеспечивает большую гибкость при работе со шрифтами благодаря предоставлению соответствующих абстракций и возможности динамического выбора шрифтов. Вот очень короткая программа, которая печатает на консоли Java имена всех имеющихся в системе шрифтов.

```
/*
* &ltapplet code="WhatFontsAreHere" width=100 height=40>
* </applet>
*
*/
import java.applet.*;
import java.awt.*;
        public class WhatFontsAreHere extends Applet {
        public void init() {
                String FontList[];
                FontList = getToolkit().getFontList();
                for (int i=0; i < FontList.length; i++) {
                        System.out.println(i + ": " + FontList[i]);
                        }
                }
        }
```

9.1. drawString

В предыдущих примерах использовался метод drawString(String, x, y). Этот метод

выводит строку с использованием текущих шрифта и цвета. Точка с координатами (x,y) соответствует левой границе базовой линии символов, а не левому верхнему углу, как это принято в других методах рисования. Для того, чтобы понять, как при этом располагается описывающий строку прямоугольник, прочтите раздел о метрике шрифта в конце этой главы.

9.2. Использование шрифтов

Конструктор класса Font создает новый шрифт с указанным именем, стилем и размером в пунктах:

```
        Font StrongFont = new Font("Helvetica", Font.BOLD|Font.ITALIC, 24);
```

В настоящее время доступны следующие имена шрифтов: Dialog, Helvetica, TimesRoman, Courier и Symbol. Для указания стиля шрифта внутри данного семейства предусмотрены три статические переменные.—Font.PLAIN, Font.BOLD и Font.ITALIC, что соответствует обычному стилю, курсиву и полужирному.

Теперь давайте посмотрим на несколько дополнительных методов.

9.2.1. getFamily и getName

Метод getFamily возвращает строку с именем семейства шрифтов. С помощью метода getName можно получить логическое имя шрифта.

9.2.2. getSize

Этот метод возвращает целое число, представляющее собой размер шрифта в пунктах.

9.2.3. getStyle

Этот метод возвращает целое число, соответствующее стилю шрифта. Полученный результат можно побитово сравнить со статическими переменными класса Font:—PLAIN, BOLD и ITALIC.

9.2.4. isBold, isItalic, isPlain

Эти методы возвращают true в том случае, если стиль шрифта—полужирный (bold), курсив (italic) или обычный (plain), соответственно.

9.3. Позиционирование и шрифты: FontMetrics

В Java используются различные шрифты, а класс FontMetrics позволяет программисту точно задавать положение выводимого в апплете текста. Прежде всего нам нужно понять кое-что из обычной терминологии, употребляемой при работе со шрифтами:

- Высота (height)—размер от верхней до нижней точки самого высокого символа в шрифте.
- Базовая линия (baseline)—линия, по которой выравниваются нижние границы символов (не считая снижения (descent)).
- Подъем (ascent)—расстояние от базовой линии до верхней точки символа.
- Снижение (descent)—расстояние от базовой линии до нижней точки символа.
- Использование FontMetrics

Ниже приведены некоторые методы класса FontMetrics:

9.3.1. stringWidth

Этот метод возвращает длину заданной строки для данного шрифта.

9.3.2. bytesWidth, charsWidth

Эти методы возвращают ширину указанного массива байтов для текущего шрифта.

9.3.3. getAscent, getDescent, getHeight

Эти методы возвращают подъем, снижение и ширину шрифта. Сумма подъема и снижения дают полную высоту шрифта. Высота шрифта—это не просто расстояние от самой нижней точки букв g и y до самой верхней точки заглавной буквы T и символов вроде скобок. Высота включает подчеркивания и т.п.

9.3.4. getMaxAscent и getMaxDescent

Эти методы служат для получения максимальных подъема и снижения всех символов

в шрифте.

9.3.5. Центрирование текста

Давайте теперь воспользуемся методами объекта FontMetrics для получения подъема, снижения и длины строки, которую требуется нарисовать, и с помощью полученных значений отцентрируем ее в нашем апплете.

```
/*
* <applet code="HelloWorld" width=200 height=100>
* </applet>
*
*/
import java.applet.*;
import java.awt.*;
        public class HelloWorld extends Applet {
                final Font f = new Font("Helvetica", Font.BOLD, 18);
                public void paint(Graphics g) {
                Dimension d = this.size();
                g.setColor(Color.white);
                g.fillRect(0,0,d.width,d.height);
                g.setColor(Color.black);
                g.setFont(f);
                drawCenteredString("Hello World!", d.width, d.height, g);
                g.drawRect(0,0,d.width-1,d.height-1);
                }
        public void drawCenteredString(String s, int w, int h, Graphics g) {
                FontMetrics fm = g.getFontMetrics();
                int x=(w-fm.stringWidth(s))/2;
                int y=(fm.getAscent()+(h-(fm.getAscent()+fm.getDescent()))/2);
                g.drawString(s, x, y);
                }
}
```