1. Класс Applet

Класс Applet содержится в пакете java.applet. Applet включает несколько методов, которые дают детальный контроль над выполнением апплета. Дополнительно java.applet определяет три интерфейса: AppletContext, AudioClip и AppletStub.

1.1. Основы апплетов

Все апплеты являются подклассами Applet. Таким образом, они должны импортировать java.applet, а также java.awt. AWT — сокращение Abstract Window Toolkit (абстрактный оконный интерфейс). Так как все апплеты выполняются в окне, необходимо включить поддержку для этого окна. Алплеты не исполняются Java-интерпретатором времени выполнения, работающим в консольном режиме. Они выполняются или Web-браузером или программой просмотра апплета, называемой appletviewer и поставляемой с пакетом разработки JDK (Java Developer Kit, инструментарий разработчика Java).

Выполнение апплета не начинается с метода main(). Некоторые из них даже содержат метод main(), однако выполнение апплета начинается и управляется совершенно иным механизмом. Вывод в окно апплета не выполняется методом system.out.printin(). Для вывода используются различные AWT-методы, такие как drawstring(), который выводит строку в указанную точку экрана. Ввод также обрабатывается иначе, чем в приложении.

Как только апплет откомпилирован, он включается в HTML-файл, используя тег <appiet>. Апплет будет выполняться Java-совместимым браузером, когда тот встретит в HTML-файле указанный тег. Для более удобного просмотра и проверки апплета просто включите в начало файла исходного кода Java-комментарий, который содержит тег <applet>. Этим способом код документируется вместе с инструкциями HTML, необходимыми апплету, и вы можете проверить откомпилированный апплет, запустив программу просмотра с вашим файлом исходного кода в качестве параметра. Пример такого комментария:

```
/*
<applet code="MyApplet" width=200 height=60>
</applet>
*/
```

Этот комментарий содержит тег <applet>, который выполнит апплет с именем MyApplet в окне с размерами 200 x 60 пикселов. Так как

включение команды <applet> делает тестирование апплетов проще, все показанные далее апплеты будут содержать соответствующий тег, внедренный в комментарий.

1.2. Класс Applet

Класс Applet определяет методы, представленные в табл.12.1. Applet обеспечивает всю необходимую поддержку для выполнения апплетов, такую как запуск и остановка. Он также реализует методы, которые загружают и показывают изображения, и методы, которые загружают и проигрывают аудио-клипы. Applet расширяет AWT-класс panel. Кроме того, Panel расширяет Container, который, в свою очередь, расширяет Component. Эти классы обеспечивают поддержку графического интерфейса Java при работе с окнами. Таким образом, Applet обеспечивает всю необходимую поддержку для работы с окнами. (AWT описан подробно далее.)

Таблица 12.1. Методы, определенные в классе Applet

Метод	Описание
void destroy()	Освобождает все ресурсы, занятые апплетом. Вызывается браузером непосредственно перед тем, как апплет завершается. Следует переопределить данный метод, если потребуется выполнить какую-нибудь дополнительную чистку перед его выполнением
AppletContext getAppletContext()	Возвращает контекст, связанный с апплетом
String getAppletInfo()	Возвращает строку, которая описывает апплет
AudioClip getAudioClip(URL url)	Возвращает объект AudioClip, который инкапсулирует аудиоклип, найденный по адресу, указанному в url
AudioClip getAudio-Clip(URL url, String clipName)	Возвращает объект AudioClip, который инкапсулирует аудиоклип, найденный по адресу, указанному в **url**, и имеющий имя, указанное в параметре clipName
URL getCodeBase()	Возвращает URL, связанный с вызывающим апплетом
URL getDocumentBase()	Возвращает URL HTML-документа, который вызывает апплет
Image getImage(URL url)	Возвращает объект Image, который инкапсулирует изображение, найденное по адресу url

Image getImage(URL url, String imageName)	Возвращает объект Image, который инкапсулирует изображение, найденное по адресу url, и имеющий имя, указанное в параметре imageName
Locale getLocale()	Возвращает объект Locale, который используется различными чувствительными к локализации классами и методами
String getParameter(String paramName)	Возвращает параметр, указанный в paramName. Если указанный параметр не найден, возвращается null (пустой указатель)
String[ ] [ ] getParameterInfo()	Возвращает таблицу строк, описывающую параметры, распознанные апплетом. Каждый вход в таблицу должен состоять из трех строк, которые содержат имя параметра, описание его типа и/или диапазона, и объяснения его цели
void init()	Вызывается, когда апплет начинает выполнение. Это первый метод, который вызывается для любого апплета
boolean isActive()	Возвращает true, если апплет был запущен. Возвращает false, если апплет был остановлен
static final AudioClip newAudioClip (URL url)	Возвращает объект AudioClip, который инкапсулирует аудиоклип, найденный по адресу url. Этот метод подобен getAudioClip() за исключением того, что он статический и может быть выполнен без потребности в Applet-объекте. (Добавлен в Java 2.)
void play (URL url)	Если аудиоклип найден по адресу url, то он проигрывается
void play (URL url, String clipName)	Если аудиоклип найден по адресу url с именем clipName, то клип проигрывается
void resize (Dimension dim)	Изменяет размеры апплета согласно измерениям, указанным в dim. Dimension — это класс пакета java.awt. Он содержит два целочисленных поля: width и height
void resize (int width, int height)	Изменяет размеры апплета согласно размерам, указанным в width и height
final void setStub (AppletStub stvubObj)	Делает stubQbj-заглушку для апплета. Этот метод используется исполнительной системой Java и обычно не вызывается апплетом. *Заглушка* — маленькая часть кода, которая обеспечивает связь между апплетом и браузером
void showStatus (String str)	Отображает значение параметра *str* в окне состояния браузера или программы просмотра апплета. Если браузер не поддерживает окно

	состояния, то никакое действие не выполняется
void start()	Вызывается браузером, когда апплет должен запустить (или возобновить) выполнение. После init() (когда апплет впервые начинает выполнение) вызывается автоматически
void stop()	Вызывается браузером, чтобы приостановить выполнение апплета. После остановки апплет перезапускается, когда браузер вызывает start()

1.3. Архитектура апплета

Апплет — программа, работающая с окном. Поэтому, его архитектура отличается от программ, основанных на консольном вводе/выводе. Имеется несколько ключевых концепций, которые нужно понимать.

Во-первых, апплеты управляются событиями. Апплет походит на набор программ обработки прерываний. Процесс выполняется так: апплет ожидает возникновение некоторого события. AWT уведомляет апплет о событии вызовом обработчика события, который был обеспечен апплетом. Как только это случается, апплет должен выполнить соответствующее действие и затем быстро возвратить управление AWT. Это критический момент. По большей части апплет не должен входить в режим работы, в котором он поддерживает управление длительный период. В тех ситуациях, когда апплет вынужден исполнять повторяющуюся задачу сам по себе (например, отображая прокручивающееся в его окне сообщение), вы должны запустить дополнительный поток выполнения

Во-вторых, пользователь инициализирует взаимодействие с апплетом. "Неоконная" программа, которая нуждается во вводе, выдает подсказку пользователю и затем вызывает некоторый метод ввода, такой как readLine(). В апплете пользователь взаимодействует с апплетом, как и когда он этого захочет. Эти взаимодействия посылаются апплету как события, на которые апплет должен ответить. Например, когда пользователь щелкает мышью внутри окна апплета, генерируется событие "щелчок мышью". Если пользователь нажимает клавишу в то время, когда окно апплета имеет фокус ввода, генерируется событие "нажатие клавиши". Апплеты могут содержать различные элементы управления, такие как кнопки и переключатели. Когда пользователь взаимодействует с одним из этих элементов управления, также генерируется событие.

1.4. Скелетная схема апплета

Все апплеты, кроме наиболее тривиальных, переопределяют набор методов, обеспечивающих основной механизм, с помощью которого браузер или программа просмотра взаимодействует с апплетом и управляет его выполнением. Четыре таких метода — init(), start(), stop() и destroy() — определены в Applet. Пятый, paint(), определен AWT-классом Component. Для всех этих методов обеспечены также и реализации по умолчанию. Апплетам не нужно переопределять те методы, которые они не используют. Однако только в очень простых апплетах не нужно определять все эти методы сразу. Все пять методов можно собрать в следующую скелетную схему:

Программа 82. Схема апплета

```
// Файл AppletSkel.java
// Скелетная схема  (скелет)  апплета.
import java.awt.*;
import java.applet.*;
/*
<applet code = "AppletSkel" width = 300 height = 100>
</applet>
*/
public class AppletSkel extends Applet { // Вызывается первым.
   public void init()   {                // инициализация
   }
   /* Вызывается вторым, после init().
      Вызывается также для перезапуска апплета.
   */
   public void start () {
   // Начало или продолжение выполнения
   }
   // Вызывается,  когда апплет остановлен.
   public void stop() {
   // Приостанавливает выполнение
 }
   /* Вызывается, когда апплет завершается.
   Это – последний выполняемый метод.
   */
   public void destroy() {
   // Выполняет завершающие действия
   }
   // Вызывается, когда окно апплета должно быть перерисовано.
   public void paint(Graphics g) {
   // Повторный показ содержимого окна
   }
}
```

Программу можно запустить непосредственно из среды Eclipse командой **Run**, **Run** или комбинацией клавиш **Ctrl+F11**. Окно апплета показано на рис. 1.

Рис. 1.Окно простейшего апплета

Среда Eclipse сама создает html-файл на время запуска апплета, помещая его в папку **bin** вместе с class-файлом откомпилированного класса (рис.2).

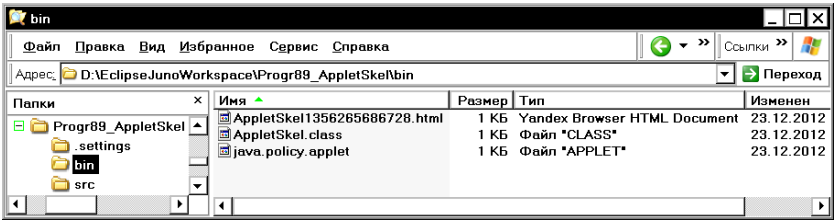

Рис. 2. Результаты компиляции апплета

Содержимое созданного автоматически в рассматриваемом примере html-файла AppletSkel1356265686728.html следующее:

```
<html>
<meta http-equiv="Content-Type" content="text/html; charset=Cp1251"/>
<body>
<applet code=AppletSkel.class width="200" height="200" >
</applet>
</body>
</html>
```

1.5. Утилита для запуска апплета

Для независимого запуска апплета нужно создать html-файл самостоятельно с помощью текстового редактора, например **Блокнота**. Создадим в папке bin файл RunAppletSkil.html:

```
<html>
<body>
<applet code = AppletSkel.class width="200" height="200" >
</applet>
</body>
</html>
```

Для запуска апплета можно воспользоваться утилитой appletviewer, запустив ее командной строкой:

```
…\>appletviewer RunAppletSkel.html
```

Удобно использовать файловый менеджер типа **Windows Commander**, который позволяет легко выбирать нужный каталог (рис.3).

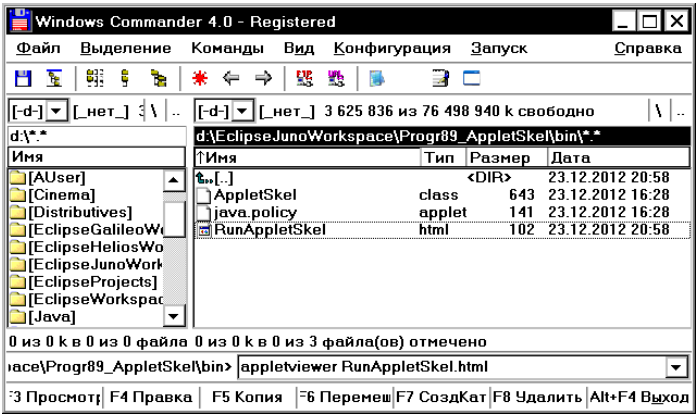

Рис. 3.Запуск апплета из **Windows Commander**

1.6. Инициализация и завершение апплета

Важно понять порядок, в котором вызываются различные методы, показанные в скелетной схеме. Когда апплет начинает выполняться, AWT вызывает методы в такой последовательности:

1. init()
2. start()
3. paint()

При завершении апплета имеет место следующая последовательность вызовов:

1. stop()
2. destroy()

Рассмотрим подробнее эти методы.

Метод init()

Метод init() — первый метод, который требует вызова. В нем необходимо инициализировать переменные. Вызывается он только однажды в течение времени выполнения апплета.

Метод start()

Метод start() следует после init(). Он также вызывается, чтобы перезапустить апплет после его остановки. В то время как init() вызывается однажды (в первый момент, когда апплет загружается),

start() запускается каждый раз, когда HTML-документ апплета отображается на экране. Так, если пользователь покидает Web-страницу и возвращается обратно, апплет возобновляет выполнение в start().

Метод paint()

Метод paint() вызывается всякий раз, когда вывод апплета должен быть перерисован. Это ситуация может возникнуть по нескольким причинам. Например, окно, в котором апплет выполняется, может быть перекрыто сверху другим окном, которое затем закрывается, или окно апплета может быть свернуто и затем восстановлено. Метод paint() вызывается также, когда апплет начинает выполнение. Таким образом, вне зависимости от причины paint() вызывается всякий раз, когда апплет должен перерисовывать свой вывод. Метод paint() имеет один параметр типа Graphics. Он должен содержать графический контекст, описывающий графическую среду, в которой выполняется апплет. Данный контекст используется всякий раз, когда требуется вывод в апплет.

Метод stop()

Метод stop() вызывается, если Web-браузер покидает HTML-документ, содержащий апплет, при переходе к другой странице. Когда вызывается stop(), апплет, вероятно, продолжает выполняться. Следует использовать stop() для приостановки потоков, не требующих выполнения, если апплет невидим. Их можно перезапустить вызывом start(), когда пользователь возвращается к странице.

Метод destroy()

Метод destroy() вызывается, когда среда решает, что апплет должен быть полностью удален из памяти. В этот момент следует освободить любые ресурсы, которые апплет может использовать. Метод stop() всегда вызывается перед destroy().

1.7. Переопределение метода update()

В некоторых ситуациях апплет может переопределить метод update(), определенный в AWT. Данный метод вызывается, когда требуется перерисовка части окна апплета. Заданная по умолчанию версия update() сначала заполняет апплет заданным по умолчанию цветом фона и затем вызывает paint(). Если вы заполняете фон, используя другой цвет в paint(), то пользователь будет видеть вспышку заданного по умолчанию фона каждый раз, когда вызывается update(), т. е. всякий раз, когда окно перерисовывается. Один из способов обойти

указанную проблему заключается в переопределении метода update() так, чтобы он исполнял все необходимые действия дисплея. Тогда, вызывая paint(), просто запускают update(). Итак, для некоторых приложений скелет апплета переопределяет paint() и update() так:

```
public void update(Graphics g) {
// Здесь повторный показ вашего окна.
}
public void paint(Graphics g) {
   update(g) ;
}
```

В примерах мы будем переопределять update(), только когда необходимо.

1.8. Простые методы отображения апплетов

Как мы уже упоминали, апплеты отображаются в окне, и они используют AWT для организации ввода и вывода. Здесь мы рассмотрим некоторые методы, процедуры и технику, необходимые для записи простых апплетов.

Чтобы вывести строку в окно апплета, используют метод drawString(), который является членом класса Graphics. Как правило, он вызывается внутри или update(), или paint(). Он имеет следующую общую форму:

```
void drawString (String message, int x, int y)
```

Здесь message — строка, которую нужно вывести, начиная с позиции x, y. (В окне Java левый верхний угол имеет позицию с координатами 0, 0.) Метод drawString() не распознает символы newline. Если нужно начать порцию текста с новой строки, требуется сделать это вручную, определяя точные (x, y) координаты, с которых нужно начать размещение строки.

Метод setBackground() устанавливает цвета фона в окне апплета, а метод setForeground() - цвет переднего плана (цвет, который применяется для отображения например, текста). Оба метода определены в Component и имеют следующие общие формы:

```
void setBackground (Color newColor)
void setForeground (Color newColor)
```

Здесь newColor — назначает новый цвет. Класс Color определяет константы, которые можно использовать для указания цвета:

- Color.black
- Color.magenta
- Color.blue
- Color.orange
- Color.cyan
- Color.pink
- Color.darkGray
- Color.red
- Color.gray
- Color.white
- Color.green
- Color.yellow

□ Color.lightGray

Например, следующие вызовы устанавливают зеленым цвет фона и красным цвет текста:

```
setBackground[Color.green);
setForeground(Color.red);
```

Удачным местом указания цветов переднего плана и фона является метод init(). Конечно, можно изменять эти цвета так часто, как необходимо, во время выполнения апплета. При этом заданный по умолчанию цвет переднего плана — черный, а заданный по умолчанию цвет фона — светло-серый.

Текущие установки для фона и символов можно получить методами getBackground() и getForeground(). Они определены в классе Component со следующими сигнатурами:

```
Color getBackground()
Color getForeground()
```

Ниже показан очень простой апплет, который устанавливает голубым цвет фона и красным цвет переднего плана (символов), а затем отображает сообщение, иллюстрирующее порядок вызова методов init(), start() и paint() после запуска апплета:

Программа 83. Установка цветов

```
// Файл Sample.java
/* Простой апплет, который устанавливает цвета
символов и фона и выводит строку. */
import java.awt.*;
import java.applet.*;
/*
<applet code="Sample" width=300 height=50>
</applet>
*/
public class Sample extends Applet {
   String msg;
   // Устанавливает цвета символов и фона
   public void init() {
      setBackground(Color.cyan);
      setForeground(Color.red);
      msg = "Inside init()  -";
   }
   // Инициализирует строку для показа
   public void start() {
      msg += " Inside start()  -";
   }
   // Показывает msg в окне апплета
   public void paint(Graphics g) {
      msg += " Inside paint().";
      g.drawString(msg, 10, 30);
   }
```

```
}
```

Этот апплет генерирует окно, представленное на рис. 19.2.

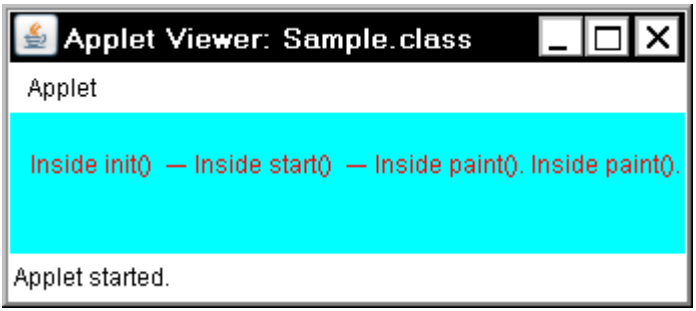

Рис. 4. Настройка цветов фона и переднего плана апплета

Методы stop() и destroy() не переопределяются, потому что они не нужны этому простому апплету.

Создадим файл RunSample.html:

```
<html>
<body>
<applet code = Sample.class width="200" height="200" >
</applet>
</body>
</html>
```

1.9. Требование перерисовки

Существует общее правило: апплет организует вывод в свое окно только тогда, когда AWT вызывает его методы update() или paint() . Это поднимает интересный вопрос: как сам апплет может вызвать модификацию своего окна, когда его информация изменяется? Например, если апплет отображает движущийся заголовок, какой механизм используется для обновления окна каждый раз, когда этот заголовок прокручивается? Одно из фундаментальных архитектурных ограничений, наложенных на апплет состоит в том, что он должен быстро возвратить управление в исполняющую систему AWT. Он не может создавать цикл внутри paint(), который, например, повторно прокручивает заголовок. Это помешало бы передаче управления обратно в AWT. При данном ограничении может показаться, что вывод в окно апплета будет, в лучшем случае, затруднен. К счастью, дело обстоит не так. Всякий раз, когда апплет должен обновить информацию, отображенную в его окне, он просто вызывает repaint().

Метод repaint() определен в пакете AWT. Он заставляет исполняющую систему AWT вызывать метод update() апплета, который в его реализации по умолчанию вызывает paint(). Таким образом, в той

части апплета, где требуется отобразить нечто в окне, просто сохраните вывод и затем вызовите repaint(). Тогда AWT выполнит вызов paint(), который может отображать сохраненную информацию. Например, если часть апплета должна вывести строку, она может сохранить эту строку в string-переменной и затем вызвать repaint(). Внутри paint() строка выводится методом drawString ().

Метод repaint() имеет четыре формы. Рассмотрим каждую по очереди. Самая простая версия repaint():

```
void repaint ()
```

Она вызывает перерисовку всего окна. Следующая версия определяет область, которая будет перерисована:

```
void repaint (int left, int top, int width, int height)
```

Здесь координаты верхнего левого угла области определены параметрами left и top, а ширина и высота области пересылается в width и height. Эти измерения определены в пикселах. При указании области для перерисовки экономится время, т. к. обновление окна — длительная операция. Если нужно обновить только маленькую часть окна, более эффективно перерисовать именно эту область.

Вызов repaint() — это, по существу, требование, чтобы ваш applet был перерисован несколько позднее. Однако, если ваша система "нетороплива" или занята, update(), возможно, не будет вызываться немедленно. Множественные запросы перерисовки, которые происходят в пределах короткого времени, могут быть отвергнуты AWT, так что update() вызывается только время от времени. Это может стать проблемой во многих ситуациях, включая мультипликацию, где постоянно требуется время для обновления. Одно из решений этой проблемы состоит в том, чтобы использовать следующие формы repaint():

```
void repaint (long maxDelay)
void repaint (long maxDelay, int x, int y, int width, int height)
```

Здесь maxDelay определяет максимальное число миллисекунд, на которое задерживается вызов update(). Однако, если это время закончится прежде, чем update() может быть вызван, он вовсе не вызывается. У метода нет никакого возвращаемого значения или выброшенного исключения, так что следует быть внимательным.

Возможен вывод в окно апплета другим способом — не методами paint() или update(). Для этого следует получить графический контекст, вызывая getGraphics() (определенный в Component), и затем использовать этот контекст для вывода в окно. Однако для большинства

приложений лучше и проще направлять вывод окна через paint() и вызывать repaint(), когда содержимое окна изменяется.

Для демонстрации repaint() разработан апплет с бегущим заголовком, который прокручивает сообщение через окно апплета справа налево. Так как прокрутка сообщения — повторяющаяся задача, она выполняется отдельным потоком, создаваемым апплетом во время инициализации. Исходный код этого апплета:

Программа 84. Бегущий заголовок

```
// Файл SimpleBanner.java
/* Апплет с бегущим заголовком.
Этот апплет создает поток, который прокручивает сообщение,
содержащееся в msg, через окно апплета.
*/
import java.awt.*;
import java.applet.*;
/*
<applet code="SimpleBanner" width = 300 height = 150>
</applet>
*/
public class SimpleBanner extends Applet implements Runnable {
   String msg = " A Simple Moving Banner.";
   Thread t = null;
   int state;
   boolean stopFlag;
   // Установка цветов и инициализация потока.
   public void init () {
      setBackground(Color.cyan);
      setForeground(Color.red);
   }
   // Запустить поток.
   public void start() {    // Это для апплета
      t = new Thread(this);
      stopFlag = false;
      t.start();            // Запуск апплета в отдельном потоке
   }
   // Точка входа для потока, который запускает заголовок.
   public void run() {
      char ch;
      // Показать заголовок.
      for(;;) {
         try {
            repaint();
            Thread.sleep(250);
            ch = msg.charAt(0);
            msg = msg.substring(1, msg.length());
            msg += ch;
            if(stopFlag)
               break;
         }
         catch(InterruptedException e) {}
      }
```

```
    }
    // Остановить заголовок.
    public void stop() {
       stopFlag = true;
       t = null;
    }
    // Показать заголовок.
    public void paint(Graphics g) {
       g.drawString(msg, 50, 30);
    }
}
```

Пример вывода изображен на рис. 5.

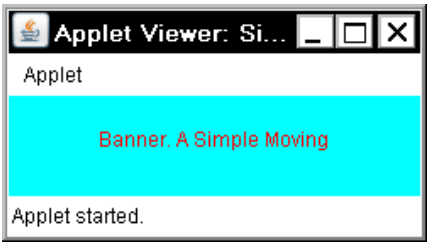

Рис. 5. Бегущая строка в окне апплета

Рассмотрим подробнее работу этого апплета. Во-первых, заметим, что SimpleBanner расширяет класс Applet и реализует интерфейс Runnabie. Это необходимо, т.к. апплет будет создавать второй поток выполнения, который используется для прокручивания заголовка. Внутри init() устанавливаются цвета переднего плана и фона апплета.

После инициализации исполнительная система AWT вызывает метод start(), чтобы начать выполнение апплета. Внутри start() создается новый поток выполнения и назначается Thread-переменной t. Затем, булевой переменной stopFlag, которая управляет выполнением апплета, присваивается значение false. Далее, вызов t.start() запускает поток. Напомним, что t.start() вызывает метод, определенный классом Thread, который начинает выполнение с помощью метода run(). Он не обращается к версии start(), определенной в классе Applet. Это два отдельных метода.

Внутри run() символы в строке, содержащейся в msg, циклически смещаются влево. Между каждым смещением выполняется обращение к repaint(). В итоге вызывается метод paint() и отображается текущее содержимое msg. Между каждой итерацией run() замирает на четверть секунды. Сетевой эффект run() состоит в том, что содержимое msg прокручивается справа налево в постоянно перемещающемся представлении. Переменная stopFlag проверяется на каждой итерации. Когда она становится true, метод run() завершает свое выполнение.

Если при выполнении апплета браузер переключается на просмотр новой страницы, вызывается метод stop(), который устанавливает в stopFlag значение true и завершает работу run(). Данный механизм применяется для остановки потока, когда страница больше не находится в поле зрения браузера. Когда апплет возвращается обратно в режим просмотра, start() вызывается еще раз, запуская новый поток для прокрутки заголовка.

1.10. Запуск апплета в браузере

Апплеты отображаются в окне браузера при открытии соответствующего html-файла. Браузеры считают апплеты потенциально опасными программами, поэтому могут отказываться выполнять апплеты. Кроме того, может потребоваться настроить безопасности Java до уровеня **Medium** (рис.6).

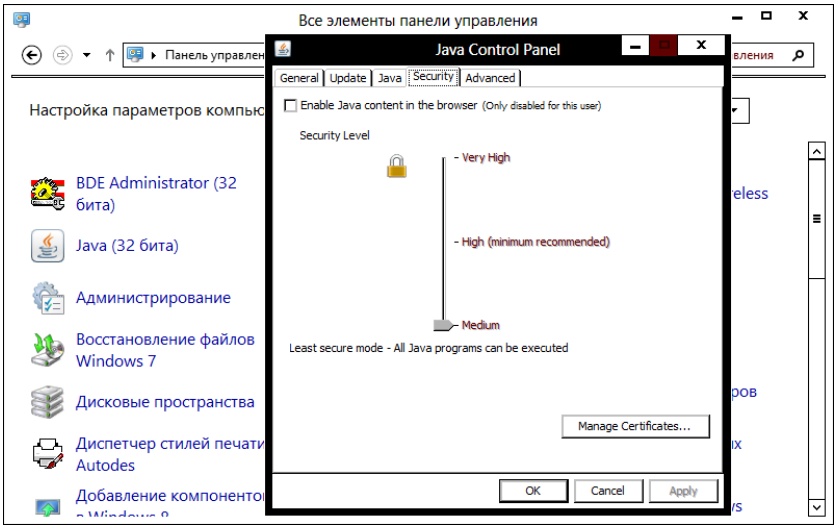

Рис. 6. Настройка уровня безопасности Java

Для запуска апплета из программы создадим файл RunSimpleBanner.html:

```
<html>
<body>
<applet code = SimpleBanner.class width="200" height="200" >
</applet>
</body>
</html>
```

Откроем этот файл RunAppletSkil.html в браузере Firefox (рис.7).

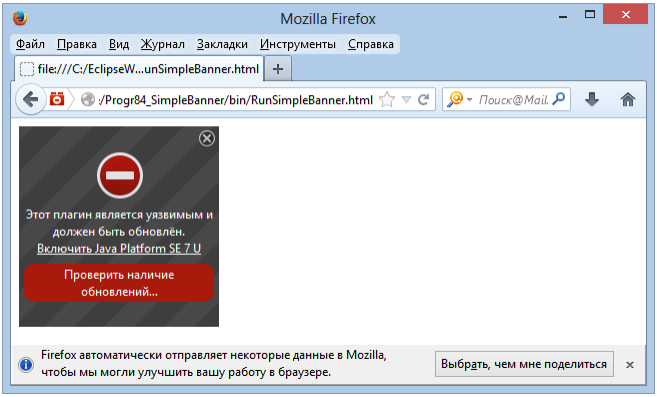

Рис. 7. Окно браузера с предложением включить Java

Нажмем в этом окне ссылку **Включить Java Platform**. Далее нажмем кнопку **Временно разрешить** (рис.8).

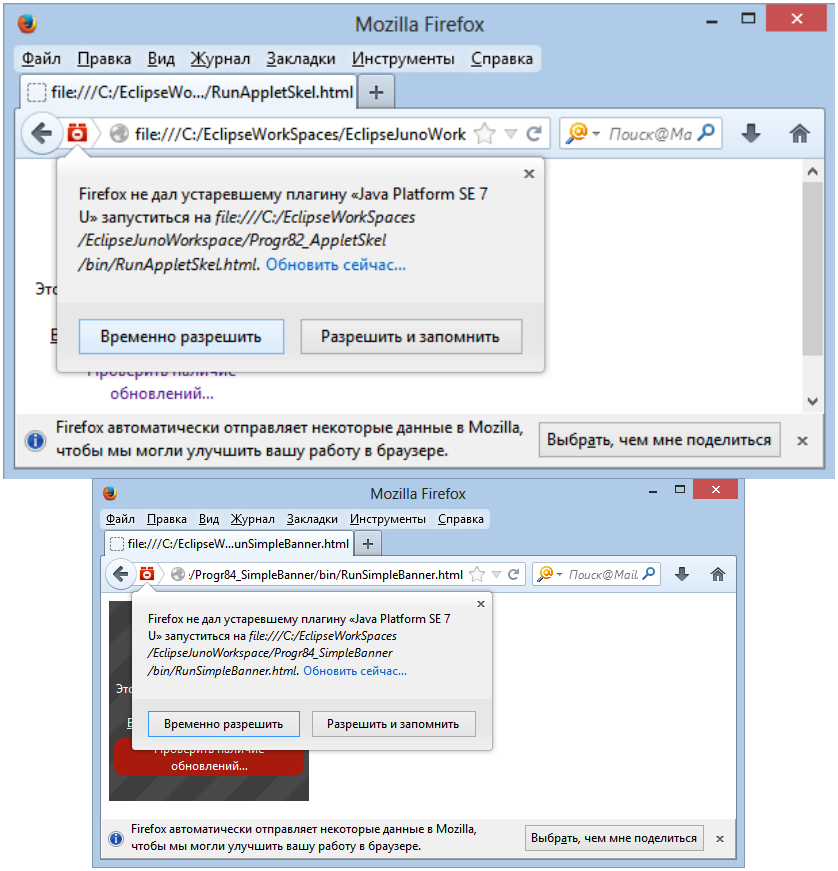

Рис. 8. Разрешение на запуск апплета

Далее нажимаем **Continue** (рис.9).

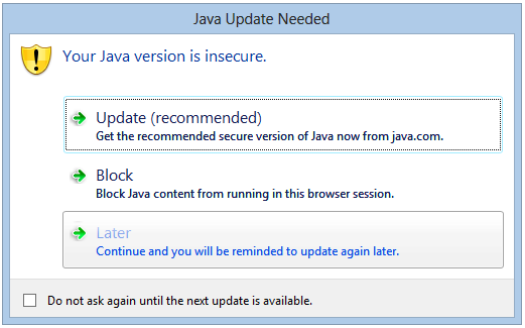

Рис. 9. Окно с предложением обновить Java

Окно браузере с работающем апплете показано на рис.10.

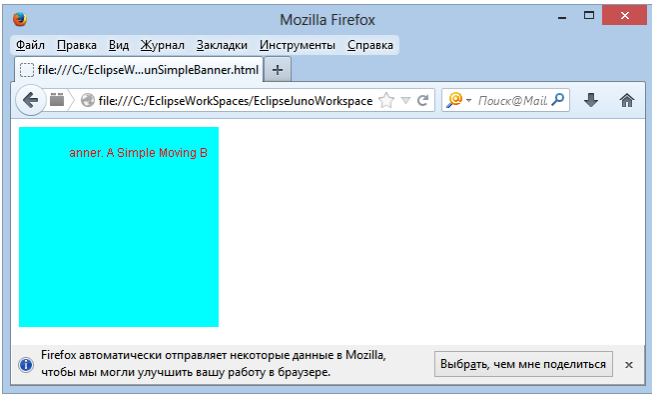

Рис. 10. Апплет, работающий в окне браузера

1.11. Использование окна состояния

В дополнение к отображению информации в своем окне, апплет может также выводить сообщение в окно состояния браузера или программы просмотра апплетов (appletviewer), которая выполняет его. Для этого нужно вызвать метод showStatus(), указывая в его аргументе строку для отображения. Окно состояния обеспечивает обратную связь пользователя с работающей программой. В нем программа может показать, что происходит в апплете (например, предоставлять сведения о режимах работы или, возможно, сообщать о некоторых типах ошибок). Кроме того, окно состояния служит средством отладки, обеспечивающим простой способ вывода информации о работе апплета.

Следующий апплет демонстрирует использование showStatus():

Программа 85. Вывод в окно состояния

```
// Файл StatusWindow.java
// Использование окна состояния.
import java.awt.*;
import java.applet.*;
/*
<applet code = "StatusWindow" width = 300 height = 50>
</applet>
*/
public class StatusWindow extends Applet {
   public void init () {
      setBackground(Color.cyan);
   }
```

```
    // Отображает msg в окне апплета.
    public void paint(Graphics g) {
        g.drawString("This is in the applet window.", 10, 20);
        showStatus("This is shown in the status window.");
    }
}
```

Пример вывода этой программы представлен на рис.11.

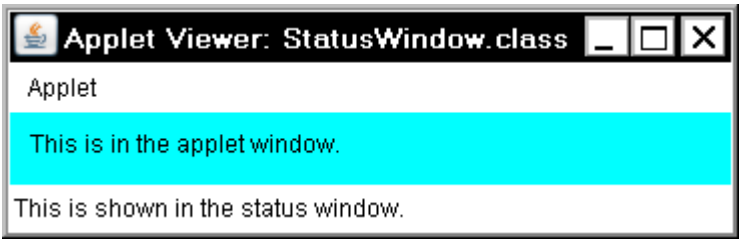

Рис. 11. Вывод в окно состояния

Тег <applet>

Апплет можно запускать как из HTML-документа, так и из программы просмотра апплета. Для этого используется тег <applet> языка HTML. Программа просмотра апплета выполняет каждый <applet>-тег, который она находит, в отдельном окне, в то время как Web-браузеры много апплетов на одной странице. Пока мы использовали только упрощенную форму тега <applet>. Теперь пришло время взглянуть на него поближе.

Далее показан синтаксис стандартного тега <applet>. Параметры в квадратных скобках — необязательны.

```
<applet
[CODEBASE = codebaseVRL]
CODE = appletFile
[ALT = altemateText]
[NAME = appletInstanceName]
WIDTH = pixels HEIGHT = pixels
[ALIGN = alignment]
[VSPACE = pixels]   [HSPACE = pixels]
>
[< param NAME = AttributeName VALUE = AttributeValue>]
[< param NAME = AttributeName2 VALUE = AttributeValue>]
[HTML Displayed in the absence of Java]
</applet>
```

Рассмотрим каждую часть подробнее.

CODEBASE — необязательный параметр, который определяет базовый URL-адрес кода апплета. Базовый URL — это каталог, в котором будет разыскиваться исполняемый файл апплета (имя этого файла указывается параметром CODE). Если атрибут CODEBASE не

определен, то по умолчанию используется базовый URL (т. е. каталог) текущего HTML-документа. Указанный в CODEBASE URL не должен находиться на хост-компьютере, из которого был считан HTML-документ.

CODE — обязательный параметр, который задает имя откомпилированного файла (с расширением .class) апплета. Этот файл относится к базовому URL кода апплета, являющемуся каталогом, в котором находится HTML-файл, или каталогом, указанным в параметре CODEBASE (если он используется).

ALT — необязательный параметр, используемый для указания короткого текстового сообщения, которое должно быть отображено, если браузер понимает тег <applet>, но не может в текущий момент выполнять Java апплеты. (Эта ситуация отличается от того случая, когда для браузеров, не поддерживающих апплеты, вы предусматриваете альтернативный HTML-документ.)

NAME — необязательный параметр, используемый для определения имени экземпляра апплета. Апплеты должны быть каким-то образом названы для обеспечения поиска и связи с ними других апплетов по имени. Для того чтобы получить апплет по имени, используйте метод getApplet(), который определен в интерфейсе AppletContext.

WIDTH и HEIGHT — это обязательные параметры, которые задают размер области показа апплета (в пикселах).

ALIGN — необязательный параметр, который определяет выравнивание апплета. Данный параметр трактуется так же, как HTML-тег <img> со следующими возможными значениями: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMTODLE и ABSBOTTOM.

VSPACE и HSPACE. Эти параметры необязательные. VSPACE определяет пустой интервал (в пикселах) выше и ниже области показа апплета. HSPACE задает пустой интервал (в пикселах) на каждой стороне области показа апплета. Они трактуются так же, как атрибуты VSPACE и HSPACE тега <img>.

Тег <param> (с параметрами NAME= и VALUE=). Вложенный тег <param> позволяет указывать на HTML-странице параметры, специфические для данного апплета. Апплет получает доступ к этим параметрам с помощью метода getParameter().

1.12. Пересылка параметров в апплеты

Как мы только что видели, HTML тег <applet> позволяет передавать параметры включающему апплету. Для получения этих параметров следует использовать метод getParameter(). Он возвращает значение указанного параметра в форме string-объекта. Таким образом, для числовых и булевых значений нужно будет преобразовать их

строчные представления во внутренние форматы. Ниже приводится пример, который демонстрирует передачу параметров:

Программа 86. Использование параметров апплета

```
// Файл ParamDemo.java
// Использование параметров.
import java.awt.*;
import java.applet.*;
/*
<applet code="ParamDemo" width = 300 height = 80>
<param name = fontName value = Courier>
<param name = fontSize vaiue = 14>
<param name = leading value = 2>
<param name = accountEnabled value = true>
</applet>
*/
public class ParamDemo extends Applet {
   String fontName;
   int fontSize;
   float leading;
   boolean active;
   // Инициализация строки для показа.
   public void start () {
      String param;
      fontName = getParameter("fontName");
      if(fontName == null)
         fontName = "Not Found";
      param = getParameter("fontSize");
      try {
         if(param != null)     // Если найден
            fontSize = Integer.parseInt(param);// Преобр. строки param
                                               // в число
         else
            fontSize = 0;
      }
      catch(NumberFormatException e) {
         fontSize = -1;
      }
      param = getParameter("leading");
      try {
         if(param != null)     // Если не найден
            leading = Float.valueOf(param).floatValue();
         else
            leading = 0;
      }
      catch(NumberFormatException e) {
         leading = -1;
      }
      param = getParameter("accountEnabled");
      if(param != null)
         active = Boolean.valueOf(param).booleanValue();
   }
   // Показ параметров на экране.
```

```
   public void paint(Graphics g) {
      g.drawString("Font name: "      + fontName, 0, 10);
      g.drawString("Font size: "      + fontSize, 0, 26);
      g.drawString("Leading: "        + leading,  0, 42);
      g.drawString("Account Active: "  + active,   0, 58);
   }
}
```

Пример вывода этой программы при запуске из среды Eclipse представлен на рис. 12.

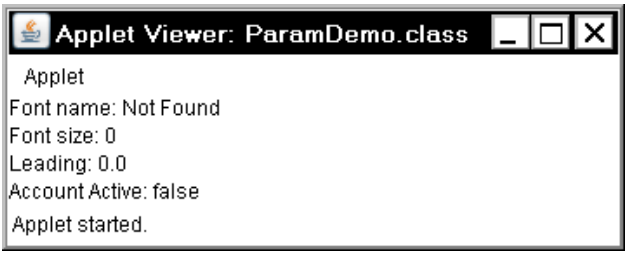

Рис. 12. Параметры апплета

Как показывает программа, следует проверить значения, возвращаемые от getParameter(). Если параметр недоступен, getParameter() возвращает null-указатель. Кроме того, в операторе try, который отлавливает исключение NumberFormatException, должны быть выполнены преобразования к числовым типам. Непойманные исключения никогда не должны появляться в пределах апплета.

Для независимого запуска создадим файл RunParamDemo.html:

```
<html>
  <body>
    <applet code = "ParamDemo" width = 300 height = 80>
      <param name = fontName  value = Courier>
      <param name = fontSize  value = 14>
      <param name = leading   value = 2>
      <param name = accountEnabled value = true>
   </applet>
  </body>
</html>
```

При запуске с использванием этого файла параметры передаются в апплет, что видно из рис.9.

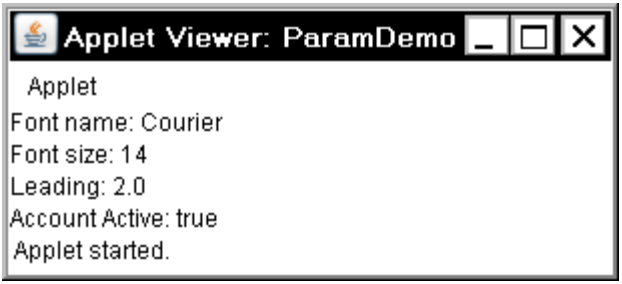

Рис. 9. Параметры получены апплетом

Усовершенствованный апплет заголовка

Передачу параметра можно использовать для усовершенствования показанного ранее апплета с бегущим заголовком. В предыдущей версии бегущее сообщение было жестко закодировано в апплете. Однако с помощью методики передачи параметра можно отображать различные сообщения при каждом прогоне апплета. Ниже показана версия апплета с бегущим заголовком, улучшенная таким способом. Обратите внимание, что в теге <applet> в верхней части исходного файла теперь определяется параметр param с именем message (сообщение), который имеет значение бегущей строки.

Программа 87. Передача в апплет строки

```
// Файл ParamBanner.java
// Параметризованный заголовок.
import java.awt.*;
import java.applet.*;
/*
<applet code="ParamBanner" width = 300 height = 50>
   <param name = message value = "Java makes the Web move!">
</applet>
*/
public class ParamBanner extends Applet implements Runnable {
   String msg;
   Thread t = null;
   int state;
   boolean stopFlag;
   // Устанавливает цвета и инициализирует поток.
   public void init() {
      setBackground(Color.cyan);
      setForeground(Color.red);
   }
   // Запустить поток.
   public void start () {
      msg = getParameter("message");
      if(msg == null)
         msg = "Message not found.";
```

```
      msg = " " + msg;
      t = new Thread(this);
      stopFlag = false;
      t.start();
   }
   // Точка входа для потока, который выполняет заголовок.
   public void run() {
      char ch;
   // Показать заголовок на экране.
      for(;;) {
         try {
            repaint();
            Thread.sleep(250);
            ch = msg.charAt(0);
            msg = msg.substring(1, msg.length());
            msg += ch;
            if(stopFlag)
               break;
         }
         catch(InterruptedException e) {}
      }
   }
   // Приостановить заголовок.
   public void stop() {
      stopFlag = true;
      t = null;
   }
   // Показать заголовок на экране.
   public void paint(Graphics g) {
      g.drawString(msg, 50, 30);
   }
}
```

Создадим следующий файл RunParamBanner.html для запуска апплета:

```
html>
<body>
<applet code="ParamBanner" width = 300 height = 250>
   <param name = message value = "Java makes the Web move!">
</applet>

</applet>
</body>
</html>
```

Окно апплета приведено на рис.13. Видно, что параметр, указанный в теге <applet> передан в апплет.

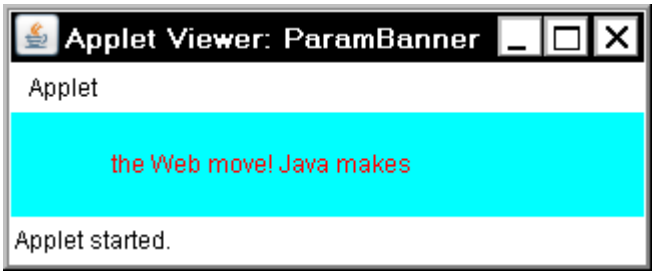

Рис. 13. Передача параметра в апплет

Методы getDocumentBase() и getCodeBase()

Часто необходимо создавать апплеты, которые будут явно загружать среду и текст, Java позволяет загружать в апплет данные из каталога, содержащего HTML-файл, запустивший апплет (т. е. из *базы документа*), и каталога, из которого был загружен class-файл апплета (т. е. из *базы кода*). Методы getDocumentBase() и getCodeBase() возвращают указанные каталоги в виде URL-объектов. Их можно сцепить со строкой, именующей загружаемый файл. Чтобы загружать другие файлы, нужно использовать другой метод — showDocument(), определенный в интерфейсе AppletContext (он обсуждается в следующем разделе).

Рассмотрим апплет, иллюстрирующий использование этих методов:

Программа 88. Отображение баз кода и документа

```
// Файл Bases.java
// Отображение баз кода и документа.
import java.awt.*;
import java.applet.*;
import java.net.*;
/*
<applet code = "Bases" width = 300 height = 50>
/applet>
*/
public class Bases extends Applet{    // Отобразить базы кода и документа.
   public void paint(Graphics g) {
      String msg;
      URL url = getCodeBase();     // Получить базу кода
      msg = "Code base:  " + url.toString();
      g.drawString(msg, 10, 20);
      url = getDocumentBase();     // Получить базу документа
      msg = "Document base:  " + url.toString();
      g.drawString (msg, 10, 40);
   }
}
```

Создадим для запуска этого апплета html-файл с названием RunBases.html:

```
<html>
   <applet code=Bases.class width="500" height="50" >
   </applet>
</html>
```

Пример вывода этой программы представлен на рис.9.

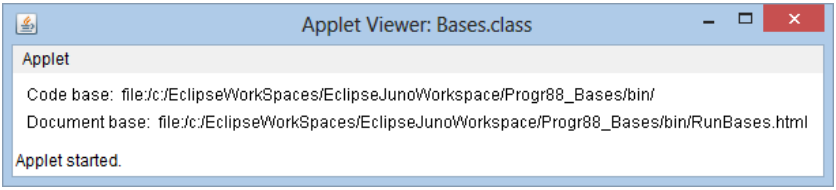

Рис. 9. Вывод базы кода и документа

Интерфейс AppletContext и метод showDocument()

На языке Java удобно программировать *графические* средства Web-навигации — изображения и мультипликацию, которые более интересны, чем текстовые адресные гиперссылки, широко используемые для этой цели в гипертекстовых документах. Чтобы разрешить апплету передавать управление на другой URL, следует использовать метод showDocument(), определенный интерфейсом AppletContext.

Интерфейс AppletContext позволяет получать информацию из среды выполнения апплета. Методы, определенные в AppletContext, показаны в табл. 19.2. Контекст выполняющегося апплета можно получить, обращаясь к методу getAppletContext(), который определен в классе Applet.

Получив свой контекст, апплет может перейти к показу другого документа, вызывая метод showDocument(). Данный метод не имеет возвращаемого значения и не выбрасывает исключение, если терпит неудачу, так что используйте его с осторожностью. Существует две формы метода showDocument() (табл. 19.2). Метод первого формата (с одним параметром — url) отображает документ из каталога, указанного в аргументе его вызова. Метод второго формата (с двумя параметрами — url и where) отображает документ, определенный первым аргументом вызова, в той области окна браузера, на которую указывает второй аргумент. Параметр where может принимать следующие (строковые) значения: "_seif" (показ в текущем фрейме апплета), "_parent" (показ в родительском фрейме апплета), "top" (показ в самом верхнем фрейме

апплета) и "blank" (показ в новом окне браузера). Можно также указать имя, под которым документ будет отображен в новом окне браузера.

Таблица 12.2. Абстрактные методы, определенные в интерфейсе AppletContext

Метод	Описание
Applet getApplet(String appletName)	Возвращает Applet-объект, имя которого специфицирует параметр ***appletName***, (если это имя находится в текущем контексте апплета). В противном случае возвращается null (пустой указатель)
Enumeration getApplets()	Возвращает перечисление, которое содержит все апплеты из текущего контекста апплета
AudioClip getAudioClip(URL url)	Возвращает AudioClip-объект (который инкапсулирует *аудиоклип*), находящийся по адресу, указанному в аргументе вызова
Image getImage (URL url)	Возвращает Image-объект (который инкапсулирует *изображение*), находящийся по адресу, указанному в аргументе вызова
void showDocument (URL url)	Начинает показ документа, находящегося по адресу, указанному в аргументе вызова. Этот метод может не поддерживаться средствами просмотра апплета
void showDocument (URL url, string where)	Начинает показ документа, находящегося по адресу, указанному в первом аргументе вызова. Этот метод может не поддерживаться средствами просмотра апплета. Место размещения документа определяется параметром ***where,*** как описано в тексте раздела
void showstatus (string str)	Показывает строку, указанную в аргументе вызова, в окне состояния

Следующий апплет демонстрирует AppletContext() и showDocument(). Во время выполнения, он получает текущий контекст апплета и использует его для передачи управления файлу с именем Test.html. Данный файл должен быть в том же каталоге, что и апплет. Test.html может содержать любой правильный гипертекст.

Программа 89. Использование контекста апплета

```
// Файл ACDemo.java
/* Использование контекста апплета,
getCodeBase (), и showDocument()
для просмотра HTML-файла.
*/
import java.awt.*;
```

```
import java.applet.*;
import java.net.*;
/*
<applet code = "ACDemo" width = 300 height = 50>
</applet>
   */
public class ACDemo extends Applet{
   public void start() {
      AppletContext ac = getAppletContext();
      URL url = getCodeBase(); // Получить url данного апплета
      try {
         ac.showDocument(new URL(url+"Test.html"));
      }
      catch(MalformedURLException e)
      {
         showStatus("URL not found");
      }
   }
}
```

Интерфейс AudioClip

Интерфейс AudioClip определяет следующие методы: piay() (проигрывает клип с начала), stop() (останавливает проигрывание) и loop() (выполняет непрерывное циклическое проигрывание). Их можно использовать для воспроизведения аудиоклипа после его загрузки методом getAudioClip().

Интерфейс AppletStub

Интерфейс Appletstub обеспечивает средства, с помощью которых апплет и браузер (или программа просмотра апплета) взаимодействуют между собой. Прикладные программисты редко реализуют этот интерфейс.

Вывод на консоль

Хотя вывод в окно апплета должен быть организован через AWT-методы типа drawString(), все еще можно использовать и консольный вывод, особенно для целей отладки. Когда вызывается метод, такой как System.out.println(), вывод не посылается в окно апплета. Вместо этого он появляется или в консольном сеансе, где вы запустили программу просмотра апплета, или в консоли Java, которая доступна в некоторых браузерах. Использование консольного вывода рекомендуется только для целей отладки, т. к. он нарушает основной принцип проектирования приложений — использование *графического интерфейса*.