Введение в практику работы с Java апплетами

Часть 1. Что такое апплет?

Апплет - это небольшая программа на языке Java, которая может выполнятся в окне браузера. Сам по себе апплет, в отличие от других java-программ, выполнятся не может - присутствие браузера тут обязательно.

Апплетом может как небольшая программа, так и большая и сложная, взаимодействующая, например, с программой на сайте, откуда этот апплет был загружен. Одно из типичных использований апплетов - это игрушки на сайтах.

Для того, чтобы апплет мог выполнятся в браузере, в последнем должна быть поддержка java. Не все браузеры подерживают java, хотя без поддержки java браузеры встречаются достаточно редко. Иногда поддержку java браузером надо дополнительно загружать - как правило, это происходит при первом посещении какой-нибудь web-странички с java-апплетом. Иногда существует две версии браузера - с и без поддержки java (например, Opera).

Не ясна поддержка апплетов и в будущих версиях IE. Microsoft заявляла, что в дальшейшем она не будет поддерживать язык java, в том числе и в своем браузере IE. Так это или не так - покажет время.

Файл с апплетом имеет расширение *.class. Разумеется, этот файл получается компиляцией из файла *.java.

С технической точки зрения апплеты представляют из себя потомков класса java.applet.Applet. Чтобы не писать такие длинные конструкции, обычно пакет java.applet просто импортируют:

```
import java.applet
```

Часть 2. Пишем первый апплет

Первый апплет, по давно укоренившейся традиции, покажет нам некоторую надпись. В нашем примере это будет "FirstApplet". Итак, создайте файл FirstApplet.java и внесите в него следующий текст:

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

Теперь создайте в той же папке, где расположен файл FirstApplet.java, HTML-файл test.htm следующего содержания:

```
<html>
<head>
    <title>FirstApplet</title>
</head>
<body>
    <applet code="FirstApplet" width="100" height="100"></applet>
</body>
</html>
```

Компилируем файл FirstApplet.java обычным образом - набрав в командной строке javac FirstApplet.java. Если ошибок нет, то в нашей папке образуется файл FirstApplet.class (который мы и используем на HTML-страничке test.htm). Откройте теперь в браузере файл test.htm. На WEB-страничке вы увидите ваш первый апплет в действии:

Теперь будем разбирать написанный нами код. Сначала код апплета. В двух первых строчках мы импортируем нужные нам классы. Так как наш класс FirstApplet - потомок класса Applet, то мы должны сделать ссылку на то, где класс Applet расположен:

```
import java.applet.*;
...
```

Это мы потом используем в строке

```
...
public class FirstApplet extends Applet{
...
```

Если бы мы не написали первый import, то нам бы пришлось написать

```
...
public class FirstApplet extends java.applet.Applet{
...
```

что выглядит слишком неуклюже. Но при любом способе мы в этой строке объявляем наш класс потомком класса Applet, т. е. наш класс автоматически умеет делать все то, что умеет класс Applet. В частности, в нем есть метод paint, который мы просто переопределяем.

Второй import

```
...
import java.awt.*;
...
```

нужен нам для рисования (вернее для использования класса Graphics). Его мы используем при выводе надписи в наш апплет. У этого класса есть выводящий некоторый текст метод drawString. Параметры у drawString простые - строка и куда она выводится.

Обратите внимание, что класс, и метод paint мы объявили как public. Это для того, чтобы класс и его метод мы могли использовать извне.

С HTML-страничкой тоже все должно быть ясно. Для включения апплета на страницу мы используем тег <applet>. У него есть параметр code, в который мы записываем имя нашего класса с апплетом. Так как имя класса совпадает с именем файла, то можно считать, что мы записываем имя файла. При этом мы можем использовать и абсолютный, и относительный (как в примере) пути. Параметры width и height означают, естественно, ширину и высоту нашего апплета в WEB-браузере.

Часть 3. Рисуем прямоугольники и эллипсы

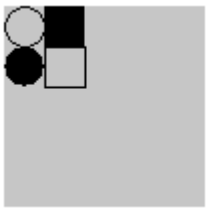
Для рисования прямоульников и эллипсов служат методы drawOval, drawRect, fillOval и fillRect класса Graphics. Первые два из них рисуют прямоугольник и эллипс соответственно, последние два служат для рисования заполненных эллипса и прямоугольника.

Вот так можно изменить код для рисования нашего апплета из прошлого частья:

```
import java.applet.*;
import java.awt.*;
public class FirstApplet extends Applet{
    public void paint(Graphics g){
        //Рисуем эллипс.
        g.drawOval(0, 0, 20, 20);
        //Рисуем прямоугольник.
        g.drawRect(20, 20, 20, 20);
        //Рисуем заполненный эллипс.
        g.fillOval(0, 20, 20, 20);
        //Рисуем заполненный прямоугольник.
        g.fillRect(20, 0, 20, 20);
    }
}
```

Параметры у этих четырех методов идентичны - первые два задают x и y левого верхнего угла рисуемой фигуры, последние два пределяют ширину и высоту.

Откомпилируйте апплет и откройте созданную на прошлом частье тестовую html-страничку. Наш апплет будет выглядеть так:



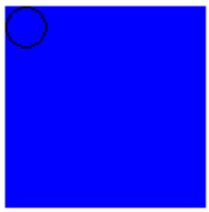
Часть 4. Закрашиваем фон

По умолчанию апплет имеет серый фон. Для изменения фона служит метод setBackground. Этот метод принимает в качестве параметра переменную типа Color.

Вот пример апплета с синим цветом фона:

```
import java.applet.*;
import java.awt.*;
public class FirstApplet extends Applet{
    public void paint(Graphics g){
        g.drawOval(0, 0, 20, 20);
        //Устанавливаем цвет фона.
        setBackground(new Color(0, 0, 255));
    }
}
```

Откомпилируйте апплет и откройте созданную на прошлом частье тестовую html-страничку. Наш апплет будет выглядеть так:

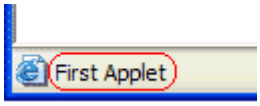


Часть 5. Информация в строке состояния браузера

Апплет может показывать информацию в строке браузера. Делается это через метод showStatus:

```
import java.applet.*;
import java.awt.*;
public class FirstApplet extends Applet{
    public void paint(Graphics g){
        showStatus("First Applet");
    }
}
```

Параметр метода showStatus выведется в строке состояния браузера:



Часть 6. Класс Color

Класс Color относится к пакету java.awt.

Переменная типа Color задает цвет в формате RGB (расшифровывается как Red, Gree, Blue - красный, зеленый, синий) - каждая цветовая составляющая может принимать значения от 0 до 255.

Для создания переменной типа Color можно использовать множество конструкторов. Мы рассмотрим три из них:

```
        Color color1=new Color(255, 255, 0);
        Color color2=new Color((float)0.2, (float)0.9, (float)0);
        Color color3=new Color(0xFFFF00);
```

Первый конструктор задает три составляющие цвета числами от 0 (отсутствие цвета) до 255 (самый интенсивный цвет), второй - тоже самое, только интенсивность каждого цвета может задаваться вещественным числом от 0 до 1, третий конструктор задает все три цвета в одним числом (которое удобнее всего задавать в шестнадцатеричном виде, что мы и сделали).

Кроме того в классе Color определены константы для стандартных цветов. Эти константы относятся ко всему классу, а не к конкретному экземпляру. Вот пример их использования:

```
        Color color=Color.blue;
```

Таких констант не слишком много, но основные цвета часто удобнее задавать ими. Вот все эти константы:

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

Часть 7. Изменяем шрифт

Для вывода некоторой надписи в апплет мы используем метод drawString объекта Graphics. Этот метод имеет три параметра (что на надпись и расположение на апплете). По умолчанию используется стандартный шрифт, для смены которого надо создать новый шрифт, указав в конструкторе начертание шрифта, его стиль и размер и щатем приписать созданный шрифт к объекту Graphics.

Вот пример:

```
...
  public void paint(Graphics g){
    //Создаем новый шрифт.
    Font font=new Font("Courier", Font.BOLD|Font.ITALIC, 20);
    //Приписываем шрифт к объекту Graphics.
    g.setFont(font);
    //Выводим надпись.
    g.drawString("Test", 20, 20);
    //Создаем новый шрифт.
    font=new Font("Courier", Font.ITALIC, 30);
    //Приписываем шрифт к объекту Graphics.
    g.setFont(font);
    //Выводим надпись.
    g.drawString("Test", 20, 50);
  }
...
```

Апплет с таким методом paint покажет две надписи "Test", выведенных соответствующими шрифтами и в соответствующем месте.

Часть 8. Мигающая надпись

Давайте посмотрим, как мы может достичь в апплете эффекта мигающей надписи (это как мигающие банеры, только не так надоедает ;)). Вот пример:

```
import java.awt.*;
import java.applet.*;

public class Applet1
    extends Applet
    implements Runnable {
  //Создаем новый поток.
  Thread t;
  //Переменная для смены цвета.
  boolean color = false;
  //Реализуем метод интерфейса Runnable.
  public void run() {
    while (true) {
      //Вызываем перерисовку.
      repaint();
      try {
```

```
        t.sleep(500);
      }
      catch (InterruptedException e) {
      }
    }
  }

  public void init() {
    //Создаем новый поток и запускаем его.
    t = new Thread(this);
    t.start();
  }

  public void paint(Graphics g) {
    //Присваиваем один из двух цветов.
    if (color) {
      g.setColor(Color.red);
    }
    else {
      g.setColor(Color.green);
    }
    //Изменяем переменную для цвета на противоположную.
    color = !color;
    g.drawString("Test", 20, 50);
  }
}
```

Перерисовку мы делаем в отдельном потоке, вызывая метод repaint(). Для создания отдельного потока мы производим наш класс от интерфейса Runnable, для которого мы должны реализовать только один метод run. В этом методе мы как раз и вызываем метод для перерисовки repaint(). Новый поток мы запускаем при инициализации апплета - в методе init().

После запуска апплета вы увидите надпись test то красного, то зеленого цветов.

Часть 9. Бегущая строка

Бегущую строку в апплете можно реализовать аналогично мигающей строке, рассмотренной на прошлом частье. Отличие только в том, что в методе paint мы вместо изменения цвета изменяем координату x нашей надписи. В остальноем же действуем аналогично - в методе run класса (этод метод реализует метод интерфейса Runnable) мы перерисовываем апплет путем вызова repaint() и делаем паузу путем вызова метода sleep().

```
import java.applet.*;
import java.awt.*;

public class TextMove
    extends Applet
    implements Runnable {

  int x, y; //Координаты строки.

  Thread t;
  //Реализуем интерфейс Runnable.
  public void run() {
    while (true) {
      //Перерисовываем.
      repaint();
      try {
        //Определяем скорость передвижения.
        t.sleep(10);
      }
      catch (InterruptedException e) {
      }
    }
  }

  public void init() {
    //Задаем начальные координаты надписи.
    x = 10;
    y = 30;
    //Создаем и запускаем новый поток.
```

```
    t = new Thread(this);
    t.start();
  }

  public void paint(Graphics g) {
    //Увеличиваем координату x.
    x += 1;
    //Рисуем строку.
    g.drawString("Test", x, y);
  }
}
```

При запуске апплета появится движущаяся направо надпись.

Часть 10. Работаем с мышью

Работа с мышкой происходит на основе модели делегирования событий. Дело тут происходит приблизительно так: есть некоторый источник событий - например, мышка. Этот источник генерирует определенные события. Те классы, которые должны обрабатывать события, должны в обязательном порядке реализовывать определенные интерфейсы именно для этих событий. Т. е. интерфейсы и события (в том числе и мышиные) однозначно соответствуют друг другу.

Когда происходит некоторое событие (например, щелчок мышкой или ее движение), то управление передается методу класса, в котором реализован интерфейс именно для события.

Для использования событий в некотором классе мы должны использовать пакет java.awt.event, т. е. мы должны включить в текст нашей программы следующую строку:

```
import java.awt.event.*;
```

События в библиотеке AWT классифицированы. Они представляют из себя иерархию классов. Во главе этой иерархии стоит класс EventObject. Когда генерируется некоторое событие, то фактически создается экземпляр определенного класса из этой иерархии. Этот экземпляр хранит конкретные параметры произошедшего события - например, для мышки это могут быть ее координаты. Созданный экземпляр класса события передается в качестве параметра в метод класса, который реализует интерфейс, определенный для нашего события.

Часть 11. Интерфейс MouseListener

Для работы с мышью сушествуют два интерфейса. На этом частье мы рассмотрим один из них - а именно MouseListener.

В этом интерфейсе существует следующие пять методов:

- mouseClicked (щелчок мышью)
- mouseEntered (ввод мыши в пределы апплета)
- mouseExited (вывод мыши из пределов апплета)
- mousePressed (нажатие кнопки мыши)
- mouseReleased (отпускание кнопки мыши)

Все эти методы имеют тип void и в них передается единственный параметр типа MouseEvent.

Для того, чтобы апплет подерживал указанные события, необходимо, во-первых, указать интерфейс MouseListener в качестве предка для нашего апплета:

```
...
public class FirstApplet extends Applet implements MouseListener{
...
```

Во-вторых, необходимо реализовать в классе апплета каждый из перечисленных методов интерфейса MouseListener:

```
public class FirstApplet extends Applet implements MouseListener{
    ...
    public void mouseClicked(MouseEvent me)
```

```
    {
        ...
    }
    ...
```

И в-третьих, мы должны зарегистрировать класс нашего апплета в качестве получателя сообщений:

```
public class FirstApplet extends Applet implements MouseListener{
    ...
    public void init()
    {
        addMouseListener(this);
    }
    ...
```

Часть 12. Интерфейс MouseMotionListener

Второй интерфейс, который позволяет реализовать работу с мышью - это MouseMotionListener.

В этом интерфейсе только два метода:

- mouseMoved (перемещение мыши)
- mouseDragged (перетаскивание мыши)

Как и для интерфейса MouseListener, эти оба метода имеют тип void и один параметр типа MouseEvent.

Вот пример использования этого интерфейса:

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;

public class FirstApplet extends Applet implements MouseMotionListener{
    int x, y;
    String s;
    public void paint(Graphics g){
        g.drawString(s, 10, 40);
    }
    //Метод ничего не делает, но должен быть реализован.
    public void mouseDragged(MouseEvent me)
    {
    }
    //Реализация метода mouseMoved.
    public void mouseMoved(MouseEvent me)
    {
        x=me.getX();
        y=me.getY();
        s="x="+x+", y="+y;
        repaint();
    }
    public void init()
    {
        addMouseMotionListener(this);
    }
}
```

В этом примере при движении мышки над апплетом в нем будут показываться текушие мышиные координаты.

Часть 13. Класс MouseEvent

В предыдущих частьах мы рассматривали работу с мышкой. Напомним, что для этого мы объявляли класс нашего апплета потомком интерфейсов MouseListener и MouseMotionListener. Во всех методах интерфейсов, которые мы должны были реализовать для нашего апплета, присутствовал параметр типа MouseEvent. Рассмотрим этот класс более подробно.

Указанный класс позволяет получить такие данные, как координаты мыши (x и y), нажатую кнопку мыши (левая, правая или средняя), была ли нажата одна из трех клавиш-модификаторов на клавиатуре - Ctrl, Alt или Shift.

Вот пример реализации метода mouseClicked интерфейса MouseListener:

```
public void mouseClicked(MouseEvent e){
   if(e.getButton()==MouseEvent.BUTTON1){
     x=e.getX();
     y=e.getY();
     repaint();
   }
}
```

В этом примере мы определяем, что за кнопка мыши была нажата, используя при этом метод getButton() и статическую константу BUTTON1 класса MouseEvent (для остальных кнопок мыши константы будут BUTTON2 и BUTTON3 соответственно). Если нажата левая кнопка мыши, то мы получаем координаты мыши через вызов методов getX() и getY() класса MouseEvent и вызываем перерисовку. Наряду с методами getX() и getY() можно использовать и метод getPoint() - он возвращает объект типа Point.

А вот пример, в котором мы определяем, была ли при нажатии кнопки мыши удержана клавиша Alt:

```
public void mouseClicked(MouseEvent e){
   if(e.isAltDown()){
     p = e.getPoint(); // p - переменная типа Point.
     repaint();
   }
}
```

Тут для опеределения, была ли нажата клавиша Alt, мы используем метод isAltDown класса MouseEvent. Соответствующие методы для других кнопок - это isControlDown() и isShiftDown().

Часть 14. Метод update()

Часто при перерисовке апплета надо отрисовывать заново не весь апплет, а только его часть. Например, у вас в апплете может быть неподвижный фон, по которому что-то там движеться. Для перерисовки в этом случае не надо перерисовывать весь фон, а только ту его часть, на которой находился и находится движущийся объект. При таком способе перерисовки мы избавимся и от неприятного мерцания нашего апплета.

Для такой частичной перерисовки мы использем не метод paint (который перерисовывает все), а метод update, который позволяет задать для перерисовки только часть апплета.

Вот пример, иллюстрирующий это:

```
public void update(Graphics g){
  // Задаем прямоугольник для перерисовки.
  g.clipRect(0, 0, 50, 50);
  // Вызываем перерисовку.
  paint(g);
}
```

Здесь мы задаем прямоугольник для перерисовки с помощью метода clipRect класса Graphics. Параметры у него такие - x и y начальной точки и ширина с высотой. После задания прямоугольника для перерисовки мы вызываем метод paint, в который в качестве параметра передаем тот же самый g, для которого мы только что вызвали метод clipRect. Разумеется метод paint должен быть тоже определен в нашем апплете (например, такой, как в части 8).

Часть 15. Работаем с клавиатурой

Для работы с клавиатурой используется интерфейс KeyListener. Т. е. класс нашего апплета должен реализовывать этот интерфейс для работы с клавиатурой. В этом интерфейсе имеется три метода: keyPressed, keyReleased и keyTyped - и наш апплет должен их всех реализовать. Эти методы вызываются соответственно когда пользователь нажимает и отпускает клавишу на клавиатуре и в промежутке между нажатием/отпусканием.

Вот пример апплета, реагирующего на нажатия клавиш на клавиатуре.

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
public class FirstApplet
    extends Applet
    implements KeyListener {

    String s; // Выводимая строка.

    //Реализуем интерфейс KeyListener.
    public void keyPressed(KeyEvent k_e) {
    }

    public void keyReleased(KeyEvent k_e) {
    }
    public void keyTyped(KeyEvent k_e){
      s+=k_e.getKeyChar();
      repaint();
    }
    public void init() {
      // Добавление слушателя для мишиных событий.
      addKeyListener(this);
      // Установка фокуса на апплет.
      requestFocus();
    }

    public void paint(Graphics g) {
      //Рисуем строку.
      g.drawString(s, 10, 10);
    }
  }
```

После запуска апплета в нем будет появляться текст, который мы будем набирать на клавиатуре (для этого, возможно, придеться сначала щелкнуть на апплете мышкой). Также обратите внимание, что нажатие разных специальных клавиш будет обратываться некорректно - например, клавиша backspace не будет стирать последний символ, а будет, наоборот, добавлять еще один символ (который будет отображаться в виде квадратика). Ни и, конечно, методы keyPressed и keyReleased мы тут добавили просто потому, что они должны быть в нашем классе, так как они присутствуют в нашем интерфейсе KeyListener. В этих обработчиках мы ничего не пишем.

Часть 16. Вставляем изображение в апплет

Разумеется, рисовать в апплете с использованием примитивов - задача не из легких. Поэтому части поступают так - рисуют качественные изображения в каком-нибудь графическом редакторе, сохраняют их в формате gif или jpg, и затем показывают в апплете изображение из этого файла.

Вот пример такого использования графического файла:

```
import java.awt.*;
import java.applet.*;
import java.awt.image.*;

public class FirstApplet
    extends Applet{

    Image img; // Изображение.

    public void init() {
     img = getImage(getDocumentBase(), "1.gif");
    }

    public void paint(Graphics g) {
      //Выводим изображение.
      g.drawImage(img, 10, 10, this);
    }
  }
```

Первым делом мы тут импортируем нужное пространство имен java.awt.image. Далее путем вызова метода getImage мы загружаем в переменную класса img типа Image нужный графический файл. Этод метод имеет два параметра - первый задает расположение файла, второй параметр - имя файла. Для первого параметра мы используем метод getDocumentBase, возвращающий местоположение документа (html-страницы в данном случае).

Потом загруженное изображение мы выводим в методе paint путем вызова метода drawImage.

Апплет будет выглядеть приблизительно так:

Часть 17. Заполение фона градиентом

Для заполнения фона апплета градиентом мы просто будем рисовать вертикальные линии. Каждая такая линия будет иметь цвет, незначительно отличющийся от цвета соседей. Разумеется, что все линии будут располагаться вплотную друг к другу.

А вот и код:

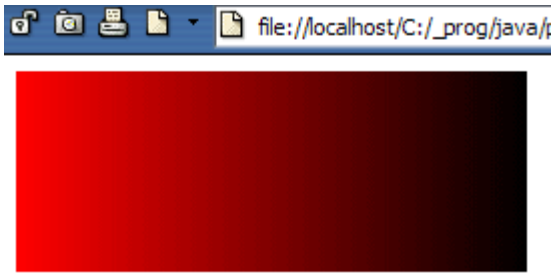
```
import java.awt.*;
import java.applet.*;
public class FirstApplet
    extends Applet{

    int a, b; // Ширина и высота окна.

    public void init() {
      // Получаем ширину и высоту апплета.
      a = getSize().width;
      b = getSize().height;
    }

    public void paint(Graphics g) {
      // Заполняем градиентом.
      // Двигаемся по ширине апплета.
      for(int i=0; i < a; i++){
        // Установка цвета в зависимости от x.
        g.setColor(new Color(255-i, 0, 0));
        // Рисование вертикальной линии.
        g.drawLine(i, 0, i, b);
      }
    }
  }
```

Апплет будет выглядеть приблизительно так:



Часть 18. Передача данных в апплет

Часто нам надо настроить апплет в соответствии с некоми параметрами. Параметры могут браться из различных источников - например из внешнего файла или из базы данных. На этом частье мы рассмотрим, как брать параметры из html-файла.

Вот пример код апплета, берущего параметры из html-страницы:

```
import java.applet.*;
import java.awt.*;
public class Applet1
    extends Applet {
  String myString;
  int fontSize;
  Font font;
  public void init() {
      // Берем строковый параметр.
      myString = this.getParameter("str");
      // Берем целочисленный параметр.
      fontSize = Integer.parseInt(this.getParameter("size"));
      // Устанавливаем размер шрифта
      // в соответствии с полученным параметром.
      font = new Font("Courier", Font.BOLD | Font.ITALIC, fontSize);
  }
  public void paint(Graphics g) {
    // Устанавливаем шрифт.
    g.setFont(font);
    // Выводим надпись нужным шрифтом.
    g.drawString(myString, 10, 40);
  }
}
```

Как вы видите, основной метод для взятия кода - это getParameter. Возвращает он строку - которую при необходимости надо превратить в нужный тип (как мы превратили в целое число параметр size).

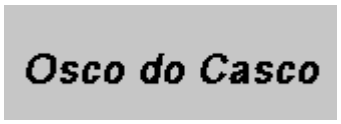
Откуда же наш апплет берет значения параметров? Как уже говорилось, из самой html-страницы. Вот код для такой страницы:

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=windows-1251">
<title>
HTML Test Page
</title>
</head>
<body>
<applet
  codebase = "."
  code     = "project1.Applet1.class"
  name     = "TestApplet"
  width    = "200"
  height   = "100"
  hspace   = "0"
  vspace   = "0"
  align    = "middle"
>
<param name = "str" value = "Igor Alexeev">
<param name = "size" value = "14">
</applet>
</body>
</html>
```

Как нетрудно заметить, значения параметров передаются в теге param, который вложен в тег applet. Для каждого параметра мы должны задать имя (name) и значение (value).

Указанный апплет при данных параметрах будет выглядеть так:

А если изменить параметры str и size, то он может выглядеть, например, так:



Часть 19. Начинаем писать игру "Убей муху"

Этот и следующий частьи мы посвятим написанию простой игры - а именно на нашем апплете будет появляться изображение мухи, которое будет менять свое положение сначала раз в секунду. Игрок должен попасть по изображению мышкой. Если попал, то, во-первых, начисляется одно очко и, во-вторых, муха начинает менять свое положение в два раза чаще (2 раза в секунду после первого попадания, 4 раза в секунду после второго и т. д.).

Начинаем писать. Вот код этого частья:

```
import java.awt.*;
import java.applet.*;

// Основной класс апплета.
public class KillFly
    extends Applet
    implements Runnable{
  // Поток.
  Thread t;
  // Координаты мыши.
  int x, y;
  // Два изображения мыши.
  Image img, img2;
  // Частота перемещения мухи.
  int period = 1000;
  //Число очков.
  int points = 0;
  // Номер показываемой картинки.
  int imgNumber = 0;
  //Реализуем метод интерфейса Runnable.
  public void run() {
    while (true) {
      try {
        // Меняем случайным образом координаты мыши.
        x = (int) (Math.random() * 400);
        y = (int) (Math.random() * 300);
        //Вызываем перерисовку.
        repaint();
        t.sleep(period);
      }
      catch (InterruptedException e) {
      }
    }
  }
```

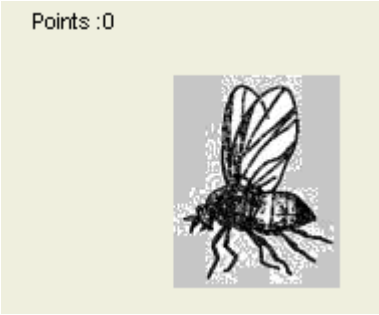
```
 // Инициализация апплета.
 public void init() {
   img = getImage(getDocumentBase(), "1.jpg");
   img2 = getImage(getDocumentBase(), "2.jpg");
   t = new Thread(this);
   t.start();
 }

 // Перерисовка.
 public void paint(Graphics g) {
   // Выбираем картинку.
   if (imgNumber == 0) {
     g.drawImage(img, x - 50, y - 50, this);
   }
   else {
     g.drawImage(img2, x - 50, y - 50, this);
   }
   // Выводим очки.
   g.drawString("Points :" + points, 20, 20);
 }
}
```

Как вы видите, анимацию мы получаем через наследование нашего класса от интерфейса Runnable (подробности см. в частье 8). Для реализации этого интерфейса мы добавляем в наш класс метод run. Внутри этого метода мы запускаем бесконеный цикл, внутри которого мы получаем новое случайное мето для нашей мухи и вызываем перерисовку.

После запуска наш апплет будет показывать муху, раз в секунду меняющую свое положение. Код, нужный для того, чтобы муха реагировала на щелчки мыши, мы добавим в следующем частье.

Наш апплет будет выглядеть приблизительно так:



Часть 20. Продолжение игры

На этом частье мы закончим наш апплет с игрой "Убей муху".

Нам осталось только добавить обработчики для щелчка мыши. При нажатии на кнопку мыши мы будет проверять, не попали ли мы на муху и если попали, то будем начислять очки, уменьшать интервал и показывать другое изображение для мухи. Кроме того, мы будем вызывать перерисовку. При отпускании кнопки мыши мы будем возвращать старое изображение и опять вызывать перерисовку.

Приступаем к коду.

Для добавдения возможности реагиирования на события мыши надо добавить строку, импортирующую соответствующие классы:

```
// Подключаем события мыши.
import java.awt.event.*;
```

Далее надо объявить наш класс потомком интерфейса MouseListener (подробности см. в частье 11):

```
public class KillFly
   extends Applet
   implements Runnable, MouseListener {
```

Так как именно наш класс и будет обрабатывать сообщения от мыши, то мы должны добавить следующие строки в метод init:

```
 public void init() {
   // Регистрируем наш класс в качестве получателя сообщений.
   addMouseListener(this);
   ...
```

Далее остается только реализовать методы интерфейса MouseListener:

```
 // Реализуем методы интерфейса MouseListener.
 // Обработчик нажатия кнопки мыши.
 public void mousePressed(MouseEvent me) {
   // Если попали на муху.
   if (x - 50 < me.getX() & me.getX() < x + 50 &
       y - 50 < me.getY() & me.getY() < y + 50) {
     // Начисляем очки.
     points++;
     // Уменьшаем интервал.
     period /= 2;
     // Изменяем номер текущей картинки.
     imgNumber = 1;
     // Перерисовываем.
     repaint();
   }
 }

 // Обработчик отпускания кнопки мыши.
 public void mouseReleased(MouseEvent me) {
   // Возвращаем старую картинку.
   imgNumber = 0;
   // Перерисовываем.
   repaint();
 }

 public void mouseClicked(MouseEvent me) {
 }

 public void mouseEntered(MouseEvent me) {
 }

 public void mouseExited(MouseEvent me) {
```

```
}
```

Теперь после запуска мы получим мини-игру - муха будет менять свое положение, мы можем пытаться попасть по ней мышкой и, если попадем, то получаем очки и муха начинает двигаться в два раза чаще.