**ЛАБОРАТОРНАЯ РАБОТА №5 часть 1**

# РАЗРАБОТКА АППЛЕТОВ, РАБОТА С ГРАФИКОЙ

**Цель**: Научиться создавать приложения с использованием апплетов.

Как известно на *Java* можно создавать два типа приложений: приложения и апплеты. В предшествующих лабораторных работах №№1-4 показывался пример листинга кода *Java*-приложений. Другой тип программ представлен *апплетом. Апплеты –* это небольшие приложения, которые доступны на *Internet*-сервере, транспортируются по *Internet*, автоматически устанавливаются и выполняются как часть *Web*-документа. После того как апплет прибывает к клиенту, он имеет ограниченный доступ к ресурсам системы, которые использует для создания произвольного мультимедийного интерфейса пользователя и выполнения комплексных вычислений без риска заражения вирусами или нарушения целостности данных. Работа с апплетами поддерживается пакетом *AWT*. Этот пакет достаточно большой, что позволяет работать не только с апплетами. Пакет *AWT* содержит многочисленные классы и методы, которые позволяют создавать окна и управлять ими.

Рассмотрим более подробно указанный пакет *AWT*.

## Пакет AWT

*AWT* – *Abstract Window Toolkit* – абстрактный оконный интерфейс. Основное назначение пакета *AWT*: поддержка окон апплета и создание обычных

*GUI*-приложений. Классы пакета *AWT* определяют интерфейсные окна и работу с визуальными компонентами окон и содержатся в пакете *java.awt*.

На рис. 1.1 представлен фрагмент иерархии классов *AWT*. Практически все классы пакета *AWT* являются потомками абстрактного класса *Component*.



Рис. 1.1. Фрагмент иерархии классов AWT

Рассмотрим особенности программирования апплетов. В последних лабораторных работах рассмотрим более подробно другие классы пакета *AWT*.

## Класс Applet

Для работы с апплетами предназначен класс *Applet*, который определяет методы, представленные в табл. 1.1. *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.

Таблица 1.1

Методы, определенные в классе Applet

|  |  |
| --- | --- |
| Метод  1 | Описание  2 |
| void destroy() | Освобождает все ресурсы, занятые апплетом. Вызывается браузером непосредственно перед тем, как апплет завершается.  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. |
| String getParameter(String paramName) | Возвращает параметр, указанный в *paramName*. Если указанный параметр не найден, возвращается *null* (пустой указатель) |

Окончание табл. 1.1

|  |  |
| --- | --- |
| 1 | 2 |
| void init() | Вызывается, когда апплет начинает выполнение. Это первый метод, который вызывается для любого апплета. В нем необходимо инициализировать переменные. Вызывается этот метод один раз в течение времени выполнения апплета. |
| boolean isActive() | Возвращает *true*, если апплет был запущен. Возвращает *false*, если апплет был остановлен |
| void resize(Dimension dim*)* | Изменяет размеры апплета согласно измерениям, указанным в *dim* |
| void start() | Вызывается, чтобы перезапустить апплет после его остановки. В то время как *init()* вызывается один раз (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()*. |
| void 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 *stop()* всегда вызывается перед *destroy()*. |

## Инициализация и завершение апплета

Важно понять порядок, в котором вызываются различные методы апплета. 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()*;

## Запуск апплетов

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>*-тeг, который она находит, в отдельном окне, в то время как *Web*-браузеры *Netscape Navigator*, *Internet Explorer* и *HotJava* допускают много апплетов на одной странице.

Покажем синтаксис тега *<applet>*. Параметры в квадратных скобках не обязательны.

<applet

[CODEBASE = codebaseURL]

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>

*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*).

*CODE* − обязательный параметр, который задает имя файла, содержащего откомпилированный файл (с расширением *.class*) вашего апплета.

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

*ALIGN* − необязательный параметр, который определяет выравнивание апплета. Возможные значения*: left, right, top, bottom, middle, BASELINE, TEXTTOP*, *ABSMTODLE И ABSBOTTOM*.

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

Пример *Web*-странички, на которой размещен апплет *DemoApplet.class*:

Пример 1.1

<html>

<head>

<title> DemoApplet</title>

</head>

<applet CODE="DemoApplet.class" WIDTH ="500" HEIGTH="500">

</applet> </html>

## Рисование линий в апплете

Линии рисуются методом *drawLine()* формата:

*void drawLine (int startx, int startY, int endX, int endY)*

Данный метод отображает линию (в текущем цвете рисования), которая начинается в координатах *startX*, *startY*  и заканчивается в *endX, endY.* Пример использования метода:

Пример 1.2

public void paint(Graphics g) {

g.drawLine(0, 0, 100, 100);

}

## Рисование прямоугольников в апплете

Методы *drawRect()* и *fillRect()* отображают соответственно рисованный и заполненный прямоугольник. Их форматы:

*void drawRect(int top, int left, int width, int height)*

*void fillRect(int tap, int left, int width, int height)*

Координаты левого верхнего угла прямоугольника задаются в параметрах *top* и *left, width* и *height*,указывающих размеры прямоугольника (в пикселах).

Пример 1.3

public void paint(Graphics g)

{ g.drawRect(10, 10, 60, 50);

g.fillRect(100, 10, 60, 50);

}

## Рисование эллипсов и кругов в апплете

Для рисования эллипса используется *drawOval (),* а для его заполнения – *fillOval()*. Эти методы имеют форматы:

*void drawOval(int top, int left, int width, int height)*

*void fillOval(int top, int left, int width, int height)*

Пример 1.4

public void paint(Graphics g)

{ g.drawOval(10, 10, 50, 50);

g.fillOval(100, 10, 75, 50);

}

## Рисование дуг в апплете

Дуги можно рисовать методами *drawArc()* и *fillArct()*, используя форматы:

*void drawArc(int top, int left, int width, int height, int начало, int конец)*

*void fillArc(int top, int left, int width, int height, int начало, int конец)*

Дуга ограничена прямоугольником; левый верхний угол прямоугольника определяется параметрами *top***,** *left****,*** а ширина и высота — параметрами *width* и *height.* Дуга рисуется от *начала*до углового расстояния, указанного в *конец.* Углы указываются в градусах и отсчитываются от горизонтальной оси против часовой стрелки. Дуга рисуется против часовой стрелки, если *конец* положителен, и по часовой стрелке, если *конец* отрицателен. Поэтому, чтобы нарисовать дугу от 12-часового до 6-часового положений, начальный угол должен быть 90° и угол развертки 180°.

Пример 1.5

public void paint(Graphics g) {

g.drawArc(0, 40, 70, 70, 0, 75);

g. fillArc (0, 40, 70, 70, 0, 75); }

## Работа с цветом

Работа с цветом поддерживается классом *Color*. В *Color* определено несколько цветовых констант (например *color.black*), специфицирующих ряд обычных цветов. Возможно также создание собственных цветов, применением одного из цветовых конструкторов. Обычно используются следующие его форматы:

*Color (int red, int green, int blue)*

*Color (int rgbValue)*

*Color(float red, float green, float blue)*

Пример 1.6

new Color(255, 100, 100); // светло-красный

По умолчанию графические объекты рисуются в текущем цвете переднего плана. Можно изменить этот цвет, вызывая метод *setColor ()* класса *Graphics*:

*void setColor(Color newColor)//* параметр *newColor* определяет новый                                                             цвет рисунка.

Вызывая метод *getColor ()*, возможно получение текущего цвета,:

*Color getColor()*

Следующий пример демонстрирует рисование «Домика» в апплете.

Пример 1.7

Листинг файла DrawHouseApplet.java

import java.awt.\*;

import java.applet.\*;

public class DrawHouseApplet extends Applet {

//функция прорисовки апплета

public void paint(Graphics g) {

g.setColor(Color.DARK\_GRAY);

g.drawLine(50, 150, 200, 50);

g.drawLine(200, 50, 350, 150);

g.drawLine(350, 150, 50, 150);

g.drawLine(50, 150, 200, 50);

g.drawLine(200, 50, 350, 150);

g.drawLine(350, 150, 50, 150);

g.drawRect(100, 150, 200, 200);

g.drawLine(50, 150, 200, 50);

g.drawLine(200, 50, 350, 150);

g.drawLine(350, 150, 50, 150);

g.drawRect(100, 150, 200, 200);

g.drawRect(170, 200, 60, 100);

g.drawLine(50, 150, 200, 50);

g.drawLine(200, 50, 350, 150);

g.drawLine(350, 150, 50, 150);

g.drawRect(100, 150, 200, 200);

g.drawRect(170, 200, 60, 100);

g.drawLine(200, 200, 200, 300);

g.drawLine(170, 250, 230, 250);

g.setColor(Color.MAGENTA);

g.drawString("Домик", 190, 30); }

}

Откомпилируйте файл DrawHouseApplet.java. Для этого можно использовать команду javac DrawHouseApplet.java. Потом создайте файл, листинг которого приведен ниже. Запустите его с помощью браузера.

Листинг DrawHouseApplet.html

<HTML>

<HEAD>

</HEAD>

<BODY BGCOLOR="000000">

<CENTER>

<APPLET code = "DrawHouseApplet.class" width = "500"

height = "300">

</APPLET>

</CENTER>

</BODY>

</HTML>

Результаты работы апплета показаны на рис. 1.2.

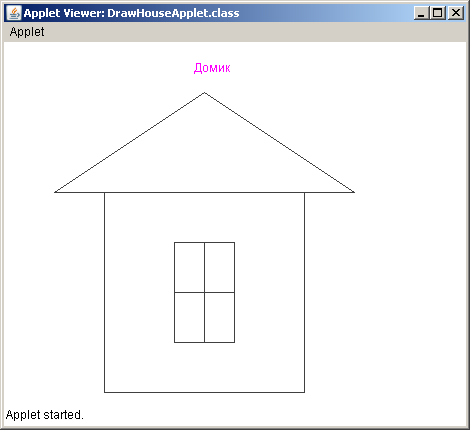


Рис. 1.2. Результаты работы апплета DrawHouseApplet

В программе ниже демонстрируется рисование строк и графических объектов разными цветами в апплете.

Пример 1.8

Листинг AppletSample.java

import java.awt.\*;

import java.applet.\*;

public class AppletSample extends Applet {

int poly\_x[]={140,180,180,140,100,100,140};//x-координаты

//для полигона

int poly\_y[]={205,225,245,265,245,225,205};//y-координаты

//для полигона

public void paint(Graphics g) {

g.setColor(Color.yellow);//выбрать желтый цвет

g.drawString("Yellow Color", 10, 30 );//нарисовать текст желтым

// цветом

g.drawLine(100, 30, 100, 100);//нарисовать линию

g.drawRect(200, 30, 60, 50);//нарисовать прямоугольник

g.fillRect(200, 30, 60, 50);//нарисовать заполненный прямоугольник

g.setColor(Color.red);//выбрать красный цвет

g.drawString("Red Color", 10, 130 );//нарисовать текст желтым

//цветом

g.drawOval(100, 130, 50, 50); //нарисовать овал

g.fillOval(200, 130, 75, 50);//нарисовать заполненный овал

g.drawArc(300, 130, 70, 70, 0, 75);//нарисовать дугу окружности

g.fillArc(400, 130, 70, 70, 0, 75);//нарисовать заполненную дугу

Color c1 = new Color(100, 100, 255);//создать собственный цвет

g.setColor(c1);//выбрать собственный цвет

g.drawString("Own Color", 10, 200 );//нарисовать текст собственным

//цветом

g.drawPolygon(poly\_x,poly\_y,poly\_x.length);//нарисовать многоугольник

}

}

Листинг AppletSample.html

<HTML>

<HEAD>

</HEAD>

<BODY BGCOLOR="000000">

<CENTER>

<APPLET

code = "AppletSample.class"

width = "500"

height = "300"

>

</APPLET>

</CENTER>

</BODY>

</HTML>

Результат работы показан на рисунке 1.3.

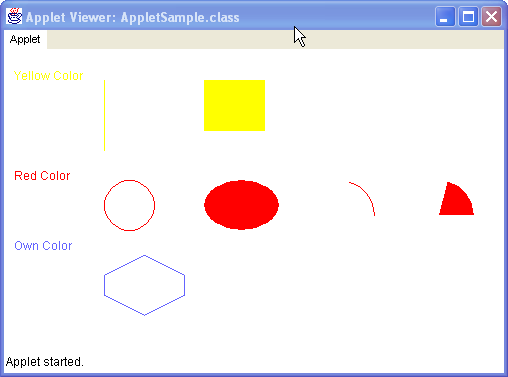


Рис. 1.3. Результат работы апплета

## Задания для самостоятельного выполнения

В следующих заданиях выполнить соответствующий рисунок в окне апплета:

1. Создать классы *Point* и *Line*. Объявить массив из n объектов класса *Point*. Для объекта класса *Line* определить, какие из объектов *Point* лежат на одной стороне от прямой линии и какие − на другой. Реализовать ввод данных для объекта *Line* и случайное задание данных для объектов *Point*.

2. Создать классы *Point* и *Line*. Объявить массив из n объектов класса *Point* и определить в методе, какая из точек находится дальше всех от прямой линии, и пометить ее другим цветом.

3.  Создать классы *Point* и *Triangle*. Объявить массив из n объектов класса *Point*. Написать функцию, какие из точек находятся внутри, а какие – снаружи треугольника (нарисовать их разными цветами).

4. Создать классы *Point* и *Rectangle*. Объявить массив из n объектов класса *Point*. Написать функцию, какие из точек находятся внутри, а какие – снаружи прямоугольника (нарисовать их разными цветами).

5. Определить класс *Line* для прямых линий, проходящих через точки A(x1,y1) и B(x2,y2). Создать массив объектов класса *Line*. Определить, используя функции, какие из прямых линий пересекаются, а какие − совпадают. Нарисовать все пересекающиеся прямые одним цветом, непересекающиеся − другим.

6.   Создать класс *Triangle*. Определить, какие из m-введенных треугольников прямоугольные (нарисовать их другим цветом).

7.   Создать класс *Triangle*. Определить, какие из m-введенных треугольников имеют площадь, больше заданной (прорисовать их другим цветом). Sтреуг=1/2((x2-x1)\*(y3-y1)-(y2-y1)\*(x3-x1))

8.   Создать классы *Point* и *Circle*. Объявить массив из n объектов класса *Point*. Для объекта класса *Circle* определить, какие из объектов *Point* лежат внутри окружности, а какие − вне. Реализовать ввод данных для объекта *Circle* и случайное задание данных для объектов *Point*.

9.   Создать свой собственный класс рисования трехмерных прямоугольников( выпуклых, вогнутых, с заливкой и без нее).

10. Привести графическое доказательство теоремы Пифагора.

11. Создать классы *Rectangle* и *Circle*. Объявить массивы из n объектов класса *Circle*. Для объекта класса *Rectangle* определить, какие из объектов *Circle* лежат внутри прямоугольника, а какие − вне. Реализовать ввод данных для объекта *Rectangle* и случайное задание данных для объектов *Circle*.

12.  Создать классы *Line* и *Circle*. Объявить массивы из n объектов класса *Line*. Для объекта класса *Circle* определить, какие из объектов *Line* пересекают окружность в двух местах, какие – в одном, и какие вообще не пересекают. Реализовать ввод данных для объекта *Circle* и случайное задание данных для объектов *Line*.

13.  Создать класс *HumanFace* с различными возможностями: улыбающийся, печальный, злой и т.п. (разное положение бровей, губ, волос).

14. Разработать апплет, выполняющий роль справочной таблицы по химическим элементам, в соответствии с периодической системой Д. И. Менделеева.

15.  Визуализация решения квадратного уравнения (построить на экране график квадратичной функции с заданными коэффициентами, отметить точки пересечения с осью абсцисс, надписать на оси значения корней).

**ЛАБОРАТОРНАЯ РАБОТА №5 часть 2**

# РАЗРАБОТКА АППЛЕТОВ С ИСПОЛЬЗОВАНИЕМ ПОТОКОВ И ИХ СИНХРОНИЗАЦИИ

**Цель**: Научиться разрабатывать апплеты с использованием потоков.

## Многопоточное программирование

В отличие от большинства других машинных языков *Java* обеспечивает встроенную поддержку для многопоточного программирования. Многопоточная программа содержит две и более части, которые могут выполняться одновременно, конкурируя друг с другом. Каждая часть такой программы называется потоком, а каждый поток определяет отдельный путь выполнения (в последовательности операторов программы).

Потоки существуют в нескольких состояниях. Поток может быть в состоянии выполнения. Может находиться в состоянии готовности к выполнению, как только он получит время *CPU*. Выполняющийся поток может быть приостановлен, что временно притормаживает его действие. Затем приостановленный поток может быть продолжен (возобновлен) с того места, где он был остановлен. Поток может быть блокирован в ожидании ресурса. В любой момент выполнение потока может быть завершено, что немедленно останавливает его выполнение. После завершения поток не может быть продолжен.

## Приоритеты потоков

*Java* назначает каждому потоку приоритет, который определяет порядок обработки этого потока относительно других потоков. Приоритеты потоков — это целые числа, которые определяют относительный приоритет одного потока над другим. Приоритет потока используется для того чтобы решить, когда переключаться от одного выполняющегося потока к другому. Это называется переключением контекста*.*

Потоку можно назначить приоритет от 1(константа *MIN\_PRIORITY*) до 10(*MAX\_PRIORITY*) с помощью метода *setPriority()*, получить значение приоритета можно с помощью метода *getPriority()*.

Правила переключения контекста следующее:

* поток может добровольно отказаться от управления;
* поток может быть приостановлен более приоритетным потоком.

## Класс Thread. Интерфейс Runnable

Многопоточная система Java построена на классе *Thread*, его методах и связанном с ним интерфейсе *Runnable*. *Thread* инкапсулирует поток выполнения. Так как невозможно непосредственно обращаться к внутреннему состоянию потока выполнения, то взаимодействие с ним осуществляется через его полномочного представителя — экземпляр (объект) класса *Thread*, который его породил. Чтобы создать новый поток, выбранная программа должна будет или расширять Класс *Thread* или реализовывать интерфейс *Runnable*.

Класс *Thread* определяет несколько методов, которые помогают управлять потоками. Таблица 2.1 содержит описание методов класса *Thread*.

Таблица 2.1

Некоторые методы класса Thread

|  |  |
| --- | --- |
| Метод | Значение |
| getName () | Получить имя потока |
| getPriority() | Получить приоритет потока |
| isAlive () | Определить, выполняется ли еще поток |
| join () | Ждать завершения потока |
| runO | Указать точку входа в поток |
| sleep () | Приостановить поток на определенный период времени |
| start () | Запустить поток с помощью вызова метода *run ()* |

## Создание потоков. Жизненный цикл потока

Программа всегда имеет один поток, который создается при выполнении программы. В программе этот поток запускается в начале метода *main()*. В апплете браузер является основным потоком. Программа создает поток в дополнение к потоку выполнения, который его создает. Для создания дополнительного потока используется объект класса *Thread*. Каждый дополнительный поток, который задает программа, представлен объектом класса *Thread* или подкласса *Thread*. Если программа имеет три дополнительных потока, то необходимо создать три таких объекта.

Для начала выполнения потока вызывается метод *start()* объекта *Thread*. Код, который выполняется в новом потоке, всегда является методом с именем *run()*, который является открытым (*public*), не имеет аргументов и не возвращает значение. Потоки, отличные от основного потока программы, всегда запускаются методом *run()* объекта, который представляет поток.

При реализации интерфейса *Runnable* необходимо определить его единственный метод *run()*.

Программа, которая создает три потока, схематически проиллюстрирована на рис. 2.1. Приведенная иллюстрация показывает метод *main()*, создающий три потока. Однако не обязательно так должно быть всегда. Любой поток может создавать дополнительные потоки.





Рис. 2.1. Консольная программа, порождающая три потока

## Синхронизация потоков

Основное различие между потоками и процессами состоит в том, что процессы защищены от воздействия друг на друга средствами операционной системы (каждый процесс выполняется в своем адресном пространстве). Использование потоков, лишенных подобной защиты, позволяет быстро запускать новые потоки и способствует их производительности. Однако здесь есть и отрицательный эффект – любой из потоков может получить доступ и даже внести изменения в данные, которые другой поток считает принадлежащими только ему. Решение этой проблемы состоит в синхронизации потоков. Ситуация, когда много потоков, обращающихся к некоторому общему ресурсу, начинают мешать друг другу, очень часта. Например, когда два потока записывают информацию в файл/объект/поток. Синхронизация кода реализуется двумя основными способами.

1. Если критическим участком является метод, то можно просто указать ключевое слово *sychronized* в объявлении метода, т.е.

*synchronized void myMethod ( ) { … }*

Эквивалентный код можно представить в виде

*void myMethod ()*

*{synchronized (this)*

*{…………….}}*

2. Если нет доступа к классу, в котором объявлен метод, то для его синхронизации можно использовать следующий прием:

*synchonized (object)*

*{ // операторы критического участка, в том числе и вызовы метода}*

Здесь *object* – ссылка на объект, который нужно синхронизировать, т.е. на объект элементом которого является вызываемый метод. При синхронизации одного оператора фигурные скобки можно опускать.

В Java кроме использования блока *synchonized* разработаны и эффективные средства межпроцессового взаимодействия. Например, метод

*public final void wait ( ) throws InterruptedException;*

осуществляет перевод вызывающего потока в режим ожидания, пока

некоторый другой поток не введет *notify()*.

Существуют и другие варианты метода, например:

*public final void wait (long timeout ) throws InterruptedException;*

осуществляет задержку на определенное время.

Метод

*public final void notify();*

«пробуждает» на том же объекте первый поток, который вызвал ожидание – *wait( )*;

Следующий метод

*public final void notifyAll();*

пробуждает все потоки, для которых вызван – *wait( )* и первым будет выполняться поток с наибольшим приоритетом.

Рассмотрим несколько примеров программ с использованием потоков.

Пример 2.1 является модификацией примера апплета, приведенного в лабораторной работе №1 (см. пример 1.7). Листинг кода ниже демонстрирует, как можно нарисовать тот же домик с использованием потоков.

Пример 2.1

Листинг файла DrawHouseThreadApplet.java

import java.awt.\*;

import java.applet.\*;

public class DrawHouseThreadApplet extends Applet implements Runnable

{

int level = 0;

Thread t;

//функция инициализации апплета

public void init()

{

this.setBackground(Color. white);

t = new Thread(this);

t.start();

}

//функция перерисовки апплета

public void paint(Graphics g)

{

g.setColor(Color.DARK\_GRAY);

if(level == 1)

{

g.drawLine(50, 150, 200, 50);

g.drawLine(200, 50, 350, 150);

g.drawLine(350, 150, 50, 150);

}

if(level == 2)

{

g.drawLine(50, 150, 200, 50);

g.drawLine(200, 50, 350, 150);

g.drawLine(350, 150, 50, 150);

g.drawRect(100, 150, 200, 200);

}

if(level == 3)

{

g.drawLine(50, 150, 200, 50);

g.drawLine(200, 50, 350, 150);

g.drawLine(350, 150, 50, 150);

g.drawRect(100, 150, 200, 200);

g.drawRect(170, 200, 60, 100);

}

if(level == 4)

{

g.drawLine(50, 150, 200, 50);

g.drawLine(200, 50, 350, 150);

g.drawLine(350, 150, 50, 150);

g.drawRect(100, 150, 200, 200);

g.drawRect(170, 200, 60, 100);

g.drawLine(200, 200, 200, 300);

g.drawLine(170, 250, 230, 250);

g.setColor(Color.MAGENTA);

g.drawString("Домик", 190, 30);

}

}

//запуск потока

public void run()

{

System.out.println("Run");

while(true)

{

level ++;

repaint();

try{

Thread.currentThread().sleep(3000);

}

catch(Exception ex){}

if(level == 4)

{

return;

}

}

}

}

Файл DrawHouseThreadApplet.html

<HTML>

<HEAD>

</HEAD>

<BODY bgcolor=white>

<CENTER>

<h1><font color=lightblue> <I>House</I></font></h1>

<APPLET

code = "DrawHouseThreadApplet.class"

width = "400"

height = "400"

>

</APPLET>

</CENTER>

</BODY>

</HTML>

Следующий пример демонстрирует управление приоритетами.

Пример 2.2

Листинг TryPriorThread.java

public class TryPriorThread extends Thread{

public TryPriorThread(String threadName){

super(threadName);

System.out.println("Thread '"+threadName+"' created!");

}

public void run(){

for(int i=0;i<10;i++){

System.out.println("Thread '"+getName()+"' "+i);

try{

sleep(1); //ожидать одну миллисекунду

}

catch(InterruptedException e){

System.out.print("Error:"+e);

}

}

}

public static void main(String [ ] args){

// создать три потока выполнения

Thread min\_thr = new TryPriorThread("ThreadMin");

Thread max\_thr = new TryPriorThread("ThreadMax");

Thread norm\_thr = new TryPriorThread("ThreadNorm");

System.out.println("Starting threads...");

min\_thr.setPriority(Thread.MIN\_PRIORITY); //задать потоку

//минимальный приоритет

//задать потоку максимальный приоритет

max\_thr.setPriority(Thread.MAX\_PRIORITY);

//задать потоку нормальный приоритет

norm\_thr.setPriority(Thread.NORM\_PRIORITY);

min\_thr.start(); // запустить первый поток

max\_thr.start(); // запустить второй поток

norm\_thr.start(); // запустить третий поток

}

}

Результат работы программы:

Thread 'ThreadMin' created!

Thread 'ThreadMax' created!

Thread 'ThreadNorm' created!

Starting threads...

Thread 'ThreadMax' 0

Thread 'ThreadNorm' 0

Thread 'ThreadMin' 0

Thread 'ThreadMax' 1

Thread 'ThreadMax' 2

Thread 'ThreadMax' 3

Thread 'ThreadMax' 4

Thread 'ThreadMax' 5

Thread 'ThreadMax' 6

Thread 'ThreadMax' 7

Thread 'ThreadMax' 8

Thread 'ThreadMax' 9

Thread 'ThreadNorm' 1

Thread 'ThreadMin' 1

Thread 'ThreadNorm' 2

Thread 'ThreadMin' 2

Thread 'ThreadNorm' 3

Thread 'ThreadMin' 3

Thread 'ThreadNorm' 4

Thread 'ThreadMin' 4

Thread 'ThreadNorm' 5

Thread 'ThreadMin' 5

Thread 'ThreadNorm' 6

Thread 'ThreadMin' 6

Thread 'ThreadNorm' 7

Thread 'ThreadMin' 7

Thread 'ThreadNorm' 8

Thread 'ThreadMin' 8

Thread 'ThreadNorm' 9

Thread 'ThreadMin' 9

Пример, демонстрирующий синхронизацию доступа к файлу.

Пример 2.3

Листинг SynchroThreads.java

import java.io.\*;

public class SynchroThreads{

public static void main(String [ ] args){

SynchroFile sf= new SynchroFile(); //объект класса SynchroFile

FileThread ft1=new FileThread("FisrtThread",sf); //первый поток

FileThread ft2=new FileThread("SecondThread",sf); //второй поток

ft1.start(); //стартовать первый поток

ft2.start(); //стартовать второй поток

}

}

class FileThread extends Thread{

String str;

SynchroFile sf;

public FileThread(String str,SynchroFile sf){

this.str=str;

this.sf=sf;

}

public void run(){

for(int i=0;i<10;i++){

sf.writing(str,i);

}

}

}

class SynchroFile{

File f=new File("file.txt");

public SynchroFile(){

System.out.println("Object SynchroFile creating...");

try{

f.delete(); //удалить файл если он есть

f.createNewFile(); //создать новый файл

}

catch(IOException ioe){

ioe.printStackTrace();

}

}

public synchronized void writing(String str,int i){

try{

RandomAccessFile raf=new RandomAccessFile(f,"rw");

raf.seek(raf.length()); //переместить указатель в конец

System.out.print(str);

raf.writeBytes(str); //записать в файл

// на случайное значение приостанавить поток

Thread.sleep((long)(Math.random()\*15));

raf.seek(raf.length()); //переместить указатель в конец

System.out.print("->"+i+" \n");

raf.writeBytes("->"+i+" \n"); //записать в файл

}

catch(IOException ioe){

ioe.printStackTrace();

}

catch(InterruptedException ie){

ie.printStackTrace();

}

notify(); //известить об окончании работы с методом

}

}

Работа программы будет выглядеть на экране следующим образом:

Object SynchroFile creating...

FisrtThread->0

SecondThread->0

FisrtThread->1

SecondThread->1

FisrtThread->2

SecondThread->2

FisrtThread->3

SecondThread->3

FisrtThread->4

SecondThread->4

FisrtThread->5

SecondThread->5

FisrtThread->6

SecondThread->6

FisrtThread->7

SecondThread->7

FisrtThread->8

SecondThread->8

FisrtThread->9

SecondThread->9

В каталоге приложения будет создан файл *file.txt*, дублирующий информацию, выведенную на экран.

Следующий пример демонстрирует применение потоков в апплете. Создается апплет, в разных потоках осуществляется движение строки, квадрата и овала, а также зарисовка фона апплета.

Пример 2.4

Листинг AppletThreadSample.java

import java.awt.\*;

import java.applet.\*;

//создать класс апплета, который реализует интерфейс Runnable

public class AppletThreadSample extends Applet implements Runnable{

private Thread T; //создать объект потока

//объявление переменных

private ShapeString m\_ShapeString = null; //для строки

private ShapeOval m\_ShapeOval = null; //для овала

private ShapeRect m\_ShapeRect = null; //для квадрата

public void run() { //реализация метода run, точка входа в поток

setBackground(Color.yellow); //фон апплета зарисовывается желтым

while (true){ //бесконечный цикл

repaint(); //перерисовка апплета или вызов метода paint

try{

T.sleep(10); //приостановка апплета на 10 миллисекунл

}

catch (InterruptedException e){ }

}

}

public void init() { //метод инициализации апплета

T = new Thread(this); //создание потока и привязка его к текущему классу

T.start(); //запуск потока (вызывается run)

//создание объектов

m\_ShapeString= new ShapeString();

m\_ShapeOval= new ShapeOval();

m\_ShapeRect= new ShapeRect();

}

public void paint(Graphics g) { //метод прорисовки апплета

//прорисовка строки

g.drawString("This is ShapeString",

m\_ShapeString.x\_String,m\_ShapeString.y\_String);

//прорисовка квадрата

g.setColor(Color.red);

g.drawRect(m\_ShapeRect.x\_Rect,m\_ShapeRect.y\_Rect,

m\_ShapeRect.w\_Rect,m\_ShapeRect.h\_Rect);

//прорисовка овала

g.setColor(Color.CYAN);

g.fillOval(m\_ShapeOval.x\_Oval,m\_ShapeOval.y\_Oval,

m\_ShapeOval.w\_Oval,m\_ShapeOval.h\_Oval);

}

//класс ShapeString реализующий интерфейс Runnable

class ShapeString implements Runnable{

Thread T;

int x\_String, y\_String; //координаты строки

public ShapeString(){ //конструктор

T = new Thread(this); //создание объекта Thread

//установление начальных координат строки

x\_String=100; y\_String=100;

T.start(); //запуск потока (вызов метода run)

}

public void run(){ //метод run

for(;;){

x\_String+=15; //изменение координаты строки

try{

T.sleep(1000); //приостановка работы потока на 1000 миллисекунд

}

catch (InterruptedException e){}

}

}

}

//класс ShapeRect реализующий интерфейс Runnable

class ShapeRect implements Runnable{

Thread T;

int x\_Rect,y\_Rect,w\_Rect,h\_Rect; //координаты и размеры квадрата

public ShapeRect(){ //конструктор

T = new Thread(this); //создание объекта Thread

//установление начальных координат квадрата

x\_Rect=350;y\_Rect=50;w\_Rect=100;h\_Rect=100;

T.start();//запуск потока (вызов метода run)

}

public void run(){ //метод run

for(;;){

x\_Rect-=15; //изменение координаты квадрата

try{

T.sleep(500); //приостановка работы потока на 1000 миллисекунд

}

catch (InterruptedException e){}

}

}

}

//класс ShapeOval реализующий интерфейс Runnable

class ShapeOval implements Runnable{

Thread T;

int x\_Oval, y\_Oval,w\_Oval,h\_Oval; //координаты и размеры овала

public ShapeOval(){ //конструктор

T = new Thread(this); //создание объекта Thread

//установление начальных координат овала

x\_Oval=30; y\_Oval=30;w\_Oval=100;h\_Oval=90;

T.start(); //запуск потока (вызов метода run)

}

public void run(){//метод run

for(;;){//изменение координат овала

x\_Oval+=8;

y\_Oval+=7;

try{

T.sleep(100); //приостановка работы потока на 100 миллисекунд

}

catch (InterruptedException e){ }

} } } }

Откомпилируйте программу. Не забудьте создать соответствующий

*html*-файл. Результат работы программы показан на рис. 2.2.



Рис. 2.2. Результат работы программы AppletThreadSample.java

## Задания для самостоятельного выполнения

1. Составить программу вывода на экран дисплея схематичного изображения велосипедиста. При запуске программы велосипедист начинает движение, вращая ногами педали велосипеда.

2. Составить программу вывода на экран дисплея схематичного изображения человека. При запуске программы человек начинает идти, размахивая в такт движения руками.

3. Составить программу вывода в верхней части экрана дисплея изображения облака. При запуске программы облако начинает двигаться и из него начинает идти дождь. При этом размер облака постепенно уменьшается.

4. Составить программу вывода в верхнюю часть экрана дисплея изображения тучи, а в нижнюю часть экрана дисплея – емкость для воды. При запуске программы начинает идти дождь. При этом размер тучи уменьшается, a емкость наполняется водой.

5. Составить программу вывода в верхнюю часть экрана дисплея изображения тучи, а в нижнюю часть экрана дисплея – сугроб. При запуске программы начинает идти снег. При этом размер тучи уменьшается, a сугроб растет.

6. Составить программу вывода на экран дисплея изображения летящего самолета.

7. Составить программу вывода на экран дисплея изображения пушки. В правой части экрана появляется и исчезает (случайным образом) мишень. Нажатием кнопки производится выстрел из пушки. Момент попадания фиксируется в виде взрыва.

8. Составить программу вывода в верхней части экрана дисплея движущегося слева направо парусника с постоянной скоростью. Ее значение всякий раз задается генератором случайных чисел. В нижней части экрана дисплея расположена пушка. При нажатии кнопки происходит выстрел торпедой с постоянной скоростью. При попадании торпеды в пушку смоделировать взрыв парусника и его исчезновение. При промахе парусник достигает правой границы экрана дисплея и начинает движение сначала с новой постоянной скоростью.

9. Составить программу вывода на экран дисплея схематичного изображения лыжника. При нажатии кнопки он начинает движение классическим стилем.

10.  Составить программу вывода на экран дисплея схематичного человека в положении готовности осуществить прыжок в длину. При нажатии кнопки спортсмен начинает разбег и выполняет такой прыжок.

11.  Составить программу вывода изображения циферблата механических часов с секундной, минутной и часовой стрелками. Организовать срабатывание будильника в заданное время.

12.  Составить программу вывода на экран дисплея песочных часов. При нажатии кнопки моделируется процесс падения песчинок, уменьшение уровня песка в верхней части колбы и увеличение в нижней части колбы.

13.  Составить программу вывода на экран дисплея треугольника. При нажатии клавиши курсор вправо треугольник вращается по часовой стрелке, клавиши курсор влево − против часовой стрелки.

14.  Составить программу вывода на экран дисплея схематичного изображения бабочки. При нажатии кнопки бабочка начинает полет, взмахивая крыльями.

15.  Составить программу вывода на экран дисплея трех вложенных друг в друга окружностей, представляющих собой беговые дорожки. На линию старта выходят три спортсмена (произвольные фигуры). При нажатии кнопки участники стартуют с одинаковой угловой скоростью. После старта угловые скорости участников забега изменяются по случайному закону. На финише указать место, занятое каждым участником забега.