Государственное бюджетное общеобразовательное учреждение средняя общеобразовательная школа №270
Красносельского района Санкт-Петербурга

Учебно-методическое пособие
«Графические возможности языка программирования Java»

Кириллова Ирина Анатольевна
учитель информатики и ИКТ, педагог дополнительного образования
ГБОУ СОШ №270 Санкт-Петербурга

Оглавление

Введение 3

Лекция 1. 3

Лекция 2. 4

Лекция 3. 5

Лекция 4. 6

Лекция 5. 8

Лекция 6. 11

Лекция 7. 12

Лекция 8. 16

Лекция 9. 19

Лабораторная работа №1. 28

Лабораторная работа №2. 30

Лабораторная работа №3. 31

Лабораторная работа №4. 31

Лабораторная работа №5. 33

Список литературы 35

Введение

Учебно-методическое пособие «Графические возможности языка программирования Java» входит в УМК по программе «Язык программирования Java». Пособие содержит 9 тем, на изучение которых отводится по одному практическому (кроме первого вводного урока) и одному теоретическому занятию.

В пособие включен цикл лабораторных работ, основными целями проведения которых являются:

- закрепление навыков программирования на языке Java;
- знакомство с графическими возможностями Java;
- закрепление уже полученных и приобретенных навыков работы на персональном компьютере в среде JBuilder;
- повышение интереса к теме “Языки программирования”.

Лекция 1.

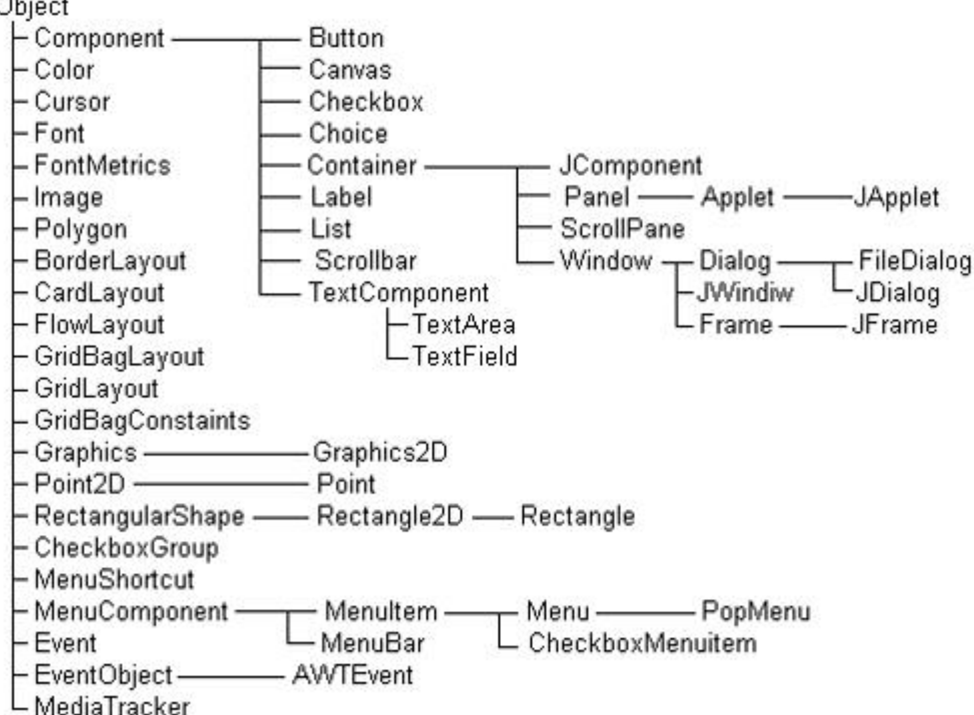
Введение в AWT.

Библиотеку графических средств Java, названную JFC (Java Foundation Classes), составляют следующие библиотеки: AWT, Swing, Java 2D (средства рисования, вывода текстов и изображений), DnD (средства, реализующие перемещение текста), Input Method Framework и Accessibility. Мы ограничимся изучением только основных средств библиотеки AWT.

AWT (Abstract Window Toolkit) содержит многочисленные классы и методы, которые позволяют создавать окна и управлять ими.

Классы AWT содержаться в пакете java.awt. Это один из самых больших пакетов Java. К счастью, он организован нисходящим, иерархическим способом, поэтому легок в понимании и использовании.

Иерархия классов AWT.



Описание некоторых классов.

Класс	Описание
AWTEvent	Инкапсулирует AVT-события
Canvas	Пустое, свободное от семантики окно

Color	Управляет цветами переносимым, независимым от платформы способом
Component	Абстрактный суперкласс для различных AWT-компонентов
Container	Подкласс Component, который может содержать другие компоненты
Font	Инкапсулирует шрифт печати
FontMetrics	Инкапсулирует различную информацию, связанную с шрифтом. Эта информация помогает отображать текст в окне
Frame	Создает стандартное окно (фрейм), которое имеет строку заголовка, углы, изменяющие размеры и строку меню
Graphics	Инкапсулирует графический контекст. Этот контекст используется различными методами вывода для отображения
Image	Инкапсулирует графическое изображение
MediaTracker	Управляет объектами среды
Panel	Самый простой конкретный подкласс класса Container
Polygon	Инкапсулирует многоугольник
Window	Создает окно без границы, строки меню и заголовка

Лекция 2.

Основы оконной графики.

AWT определяет окна согласно иерархии классов, которая с каждым уровнем добавляет функциональные возможности и специфику. Два наиболее общих типа окон являются производными от типа Panel, который пользуется апплетами, и от типа Frame, который создаёт стандартное окно.

Основное окно приложения, активно взаимодействующее с операционной системой, необходимо построить по правилам графической системы. Оно должно перемещаться по экрану, изменять размеры, реагировать на действия мыши и клавиатуры. В окне должны быть, как минимум, следующие стандартные компоненты.

- Строка заголовка (title bar), с левой стороны которой необходимо разместить кнопку контекстного меню, а с правой — кнопки сворачивания и разворачивания окна и кнопку закрытия приложения.
- Необязательная строка меню (menu bar) с выпадающими пунктами меню.
- Горизонтальная и вертикальная полосы прокрутки (scrollbars).
- Окно должно быть окружено рамкой (border), реагирующей на действия мыши.

2.1.Работа с фреймовыми окнами.

Тип окна, который вы будете чаще всего создавать, является производным от Frame. Он используется для создания окон верхнего уровня и дочерних окон и приложений. Фрейм – это окно со стандартным стилем (с заголовком, меню, обрамлением управляющими уголками). Frame поддерживает два конструктора:

Frame() – создает стандартное окно, которое не содержит заголовка;

Frame(String заголовок) – создает окно с заголовком, указанным в параметре **заголовок**.

2.2.Установка размеров окна.

Чтобы установить размеры окна, используется метод setSize(). Существует две формы этого метода (с разными списками параметров):

void setSize(int newWidth, int newHeight)

void setSize(Dimension newSize)

Новый размер окна специфируется параметрами **newWidth и newHeight** (первая форма), или полями width и height объекта класса Dimension, передаваемыми параметру **newSize.** Размеры задаются в пикселах.

Метод getSize() используется для получения текущего размера окна. Его сигнатура:

Dimension getSize().

Данный метод возвращает текущий размер окна в полях width и height объекта класса Dimension.

2.3.Скрытие и показ окна.

После создания фрейм-окно остается невидимым до тех пор, пока вы не вызовете метод setVisible(). Сигнатура этого метода имеет вид:

void setVisible(boolean visibleFlag).

Компонент становится видимым, если параметр этого метода получает значение true, иначе он остается скрытым (невидимым).

2.4.Установка заголовка окна.

Можно изменить заголовок фрейм-окна, если вызвать метод setTitle(). Он имеет следующий формат:

void setTitle(string newTitle)

где **newTitle** – новый заголовок окна.

2.5.Создание фрейм-окна в апплете.

Хотя и возможно построить фрейм-окно, просто создавая Frame-объект, но поступают так редко, потому что мало что можно с ним делать. Например, нет способности принимать и обрабатывать события, которые происходят внутри него, или выводить в него информацию. Чаще всего будем создавать подкласс Frame. Это позволит переопределить методы класса Frame и обработку событий.

Создать новое фрейм-окно внутри апплета очень просто. Сначала создайте подкласс Frame. Затем переопределите необходимые для работы с окном стандартные методы, такие как init(), start(), stop() и paint().

Определив подкласс Frame, нужно создать объект этого класса. Однако вновь построенное фрейм-окно первоначально не будет видимым (на экране). Чтобы сделать его видимым, нужно вызвать setVisible() с аргументом false. При создании окна задают высоту и ширину по умолчанию. Чтобы установить необходимый размер окна, требуется вызвать метод setSize().

Лекция 3.

Работа с графикой.

AWT поддерживает богатый набор графических методов. Вся графика рисуется относительно окна. Это может быть главное или дочернее окно апплета, а также окно автономного приложения. Начало координат каждого окна – в его верхнем левом углу и обозначается как (0, 0). Координаты определяются в пикселях. Весь вывод в окно выполняется через графический контекст. Графический контекст инкапсулирует в классе и получается двумя способами:

- Передается апплету, когда вызывается один из его многочисленных методов, таких как paint() и update();
- Возвращается методом getGraphics() класса Component.

Класс Graphics определяется ряд функций рисования. Каждая форма может быть рисованной или заполненной. Объекты рисуются и заполняются выбранным в текущий момент графическим цветом, который по умолчанию является черным. Когда графический объект превышает размеры окна, вывод автоматически усекается. Рассмотрим несколько методов рисования.

3.1. Рисование линий.

Линии рисуются методом drawLine() формата:

void drawLine(int startX, int startY, int endX, int endY)

DrawLine() отображает линию (в текущем цвете рисования), которая начинается в координатах **startX, startY** и заканчивается в **endX, endY.**

3.2. Рисование прямоугольников.

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

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

void fillRect(int top, int left, int width, int height)

Координаты левого верхнего угла прямоугольника – в параметрах **top** и **left**, **width** и **height** – указывают размеры прямоугольника (в пикселях).

Чтобы рисовать округленный прямоугольник, используйте drawRoundRect() или fillRoundRect() с форматами:

void drawRoundRect(int top, int left, int width, int height, int xDiam, int yDiam)

void fillRoundRect(int top, int left, int width, int height, int xDiam, int yDiam)

Округленный прямоугольник имеет закругленные углы. Левый верхний угол прямоугольника задается параметрами **top** и **left**. Размеры прямоугольника определяются в **width** и **height**. Диаметр округляющейся дуги по оси X определяется в **xDiam**. Диаметр округляющейся дуги по оси Y определяется в **yDiam**.

3.3. Рисование эллипсов и кругов.

Для рисования элипса используйте drawOval(), а для его заполнения - fillOval(). Эти методы имеют формат:

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

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

Эллипс рисуется в пределах ограничительного прямоугольника, чей левый верхний угол определяется параметрами **top** и **left,** а ширина и высота указываются в **width** и **height**. Чтобы нарисовать круг, в качестве ограничительного прямоугольника указывайте квадрат.

3.4. Рисование дуг.

Дуги можно рисовать методами drawArc() и fillArc(), 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, чей левый верхний угол определяется параметрами **top** и **left,** а ширина и высота – параметрами **width** и **height.** Дуга рисуется от **начала** до углового расстояния, указанного в **конец**. Углы указываются в градусах и отчитываются 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 двенадцатичасового до шести часового положения, начальный угол должен быть 90° и угол развертки 180°.

3.5. Рисование многоугольников.

Фигуры произвольной формы можно рисовать, используя методы drawPolygon() и fillPolygon() со следующими форматами:

void drawPolygon(int x[], int y[], int numPoints)

void fillPolygon(int x[], int y[], int numPoints)

Оконечные точки многоугольника определяются координатными парами, содержащимися в массивах **x[]** и **y[]**. Число точек, определенных в этих массива, указывается параметром **numPoints.** Имеются альтернативные формы этих методов, в которых многоугольник определяется объектом класса Polygon.

Лекция 4.

Работа с цветом.

4.1. Как задать цвет.

Java обеспечивает переносимость цвета, вне зависимости от устройства вывода объекта. Цветовая система AWT позволяет указывать в программе любой желаемый цвет.

Более того, AWT находит наилучшее согласование этого цвета с заданными аппаратными ограничениями дисплея, выполняющего вашу программу или апплет. Поэтому ваш код не имеет никакого отношения к различиям в способах поддержки цветов разными аппаратными устройствами. Работа с цветом поддерживается классом Color.

Основу класса составляют семь конструкторов цвета. Самый простой конструктор:

Color(int red, int green, int blue)

создает цвет, получающийся как смесь красной red, зеленой green и синей blue составляющих. Эта цветовая модель называется RGB. Каждая составляющая меняется от 0 (отсутствие составляющей) до 255 (полная интенсивность этой составляющей).

Например:

```
Color pureRed = new Color(255, 0, 0);
Color pureGreen = new Color(0, 255, 0);
```

определяют чистый ярко-красный pureRed и чистый ярко-зеленый pureGreen цвета.

Во втором конструкторе интенсивность составляющих можно изменять более гладко вещественными числами от 0.0 (отсутствие составляющей) до 1.0 (полная интенсивность составляющей):

Color(float red, float green, float blue)

Например:

```
Color someColor = new Color(0.0f, 0.4f, 0.95f);
```

Третий конструктор

Color(int rgb)

задает все три составляющие в одном целом числе. В битах 16—23 записывается красная составляющая, в битах 8—15 — зеленая, а в битах 0—7 — синяя составляющая цвета.

Например:

```
Color c = new Color(OXFF8F48FF);
```

Здесь красная составляющая задана с интенсивностью 0X8F, зеленая — 0X48, синяя — 0XFF.

Следующие три конструктора

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

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

Color(int rgb, boolean hasAlpha)

вводят четвертую составляющую цвета, так называемую "альфу", определяющую прозрачность цвета. Эта составляющая проявляет себя при наложении одного цвета на другой. Если альфа равна 255 или 1.0, то цвет совершенно непрозрачен, предыдущий цвет не просвечивает сквозь него. Если альфа равна 0 или 0.0, то цвет абсолютно прозрачен, для каждого пикселя виден только предыдущий цвет. Последний из этих конструкторов учитывает составляющую альфа, находящуюся в битах 24—31, если параметр hasAlpha равен true. Если же hasAlpha равно false, то составляющая альфа считается равной 255, независимо от того, что записано в старших битах параметра rgb. Первые три конструктора создают непрозрачный цвет с альфой, равной 255 или 1.0.

Седьмой конструктор

Color(ColorSpace cspace, float[] components, float alpha)

позволяет создавать цвет не только в цветовой модели (color model) RGB, но и в других моделях: CMYK, HSB, CIEXYZ, определенных объектом класса ColorSpace.

Для создания цвета в модели HSB можно воспользоваться статическим методом **getHSBColor(float hue, float saturation, float brightness).**

Если нет необходимости тщательно подбирать цвета, то можно просто воспользоваться одной из тринадцати статических констант типа color, имеющихся в классе Color.

Вопреки соглашению "Code Conventions" они записываются строчными буквами: **black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, yellow**.

Методы класса Color позволяют получить составляющие текущего цвета: **getRed(), getGreen(), getBlue(), getAlph(), getRGB(), getColorSpace(), getComponents().** Каждый из этих методов возвращает цветовой компонент RGB, найденный в вызывающем Color-объекте в нижних восьми битах целого числа.

Два метода создают более яркий **brighter()** и более темный **darker()** цвета по сравнению с текущим цветом. Они полезны, если надо выделить активный компонент или, наоборот, показать неактивный компонент бледнее остальных компонентов. Два статических метода возвращают цвет, преобразованный из цветовой модели RGB в HSB и обратно:

static float[] RGBtoHSB(int red, int green, int blue, float values[])

static int HSBtoRGB(float hue, float saturation, float brightness)

Создав цвет, можно рисовать им в графическом контексте.

4.2. Установка текущего цвета графики.

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

void setColor(Color newColor)

где параметр **newColor** определяет новый цвет рисунка.

Вы можете получить текущий цвет, вызывая метод getColor():

Color getColor()

Лекция 5.

Работа с текстом.

5.1. Работа со шрифтами.

Пакет AWT поддерживает множество типов шрифтов. Шрифты появились из области традиционного набора текстов и стали важной частью компьютерных документов и дисплеев. AWT обеспечивает гибкость программирования за счет того, что берет на себя операции манипулирования шрифтами и допускает их динамический выбор.

Шрифты инкапсулированы в классе Font. Некоторые методы, определенные в Font, перечислены в табл. 5.1.

Таблица 5.1. Некоторые методы, определенные в Font.

Метод	**Описание**
static Font decode(String str)	Возвращает шрифт по заданному (в параметре) имени
boolean equals(Object FontObj)	Возвращает true, если вызывающий объект содержит тот же самый шрифт, что указан в FontObj. Иначе возвращает false.
String getFamily()	Возвращает имя семейства шрифта, которому вызывающий шрифт принадлежит
static Font getFont(String property)	Возвращает шрифт, связанный с системным свойством, указанным в параметре property. Возвращает указательnull,если свойство не существует
static Font getFont(String property, Font defaultFont)	Возвращает шрифт, связанный с системным свойством, указанным в параметре property.возвращает шрифт, указанный в defaultFont, если свойство не существует
String getFontName()	Возвращает имя гарнитуры вызывающего шрифта
String getName()	Возвращает логическое имя вызывающего шрифта
int getSize()	Возвращает размер, в пунктах вызывающего шрифта
int getStyle()	Возвращает значение стиля (начертания) вызывающего шрифта
int hashCode()	Возвращает код мусора, связанный с вызывающим

	объектом
boolean isBold()	Возвращает true, если шрифт имеет Bold-начертание, иначе - false
boolean isItalic()	Возвращает true, если шрифт имеет Italic-начертание, иначе - false
boolean isPlain()	Возвращает true, если шрифт имеет Plain-начертание, иначе - false
String toString()	Возвращает строчный эквивалент вызывающего шрифта

В классе Font определены переменные, представленные в табл. 5.2.

Таблица 5.2. Переменные класса Font

Переменная	**Значение**
String name	Имя шрифта
Float pointSize	Размер шрифта в пунктах (дробный)
Int size	Размер шрифта в пунктах (целый)
Int style	Стиль (начертание) шрифта

5.1.1.Создание и выбор шрифта.

Перед выбором шрифта нужно сначала создать объект класса Font, который описывает этот шрифт. Одна из форм конструктора класса Font имеет формат:

Font (String fontName, int fontStyle, int pointSize)

Здесь **fontName** определяет имя желательного шрифта. Имя можно указывать, используя либо логическое имя, либо имя гарнитуры[1]. Все среды Java поддерживают следующие шрифты: Dialog, DialogInput, Sans Serif, Serif, Monospaced и Symbol. Шрифт Dialog используется диалоговыми окнами системы. Dialog применяется по умолчанию, если явно не устанавливается шрифт. Можно использовать любые другие шрифты, поддерживающиеся вашей специфической средой, но они могут быть не всегда доступными.

Стиль шрифта указывается параметром **fontStyle.** Он может состоять из одной или нескольких констант: Font.PLAIN, Font.BOLD, Font.ITALIC. Стили можно комбинировать, объединяя эти константы операцией OR. Например, выражение Font.BOLD | Font.ITALIC определяет стиль полужирный курсив.

Размер шрифта указывается параметром **pointSize.**

Чтобы использовать шрифт, который создали, следует выбрать его с помощью метода setFont(). Он определен в классе Component и имеет общую форму:

void setFont(Font fontObj)

Здесь **fontObj** – объект, который содержит желательный шрифт.

5.1.2.Получение информации о шрифте.

Для получения информации о текущем шрифте необходимо вызвать метод getFont(). Он определен в классе Graphics как:

Font getFont()

Как только получили текущий шрифт, можно извлекать информацию о нем, используя различные методы, определенные в классе Font.

5.2. Управление текстовым выводом.

5.2.1. Вывод текста.

Для вывода текста в область рисования текущим цветом и шрифтом, начиная с точки (x, y), в классе Graphics есть несколько методов:

drawstring (String s, int x, int y) — выводит строку s;

---

[1] Под гарнитурой шрифта принято понимать характер рисунка литер

drawBytes(byte[] b, int offset, int length, int x, int y) — выводит length элементов массива байтов b, начиная с индекса offset;
drawChars(chart] ch, int offset, int length, int x, int y) — выводит length элементов массива символов ch, начиная с индекса offset.

Четвертый метод выводит текст, занесенный в объект класса, реализующего интерфейс AttributedCharacteriterator. Это позволяет задавать свой шрифт для каждого выводимого символа:

drawstring(AttributedCharacteriterator iter, int x, int y)

Точка (x, y) — это левая нижняя точка первой буквы текста на базовой линии (baseline) вывода шрифта.

5.2.2. Класс FontMetrics.

Для большинства шрифтов не все символы имеют одинаковую ширину (такие шрифты называют пропорциональными). Кроме того, высота каждого символа, длина выносных элементов (свисающих частей, как у символов g или p) и величина пробела между горизонтальными строками изменяется от шрифта к шрифту. Далее, может быть изменен размер шрифта. Переменный характер этих атрибутов не имел бы слишком больших последствий, если бы Java не требовал от программиста ручного управления фактически всем текстовым выводом.

Учитывая что размеры каждого шрифта могут отличатся и что шрифты могут быть изменены во время выполнения программы, должен существовать некоторый способ для определения размеров и различных других атрибутов текущего шрифта. Например, для записи одной строки текста после другой необходимо как-то узнавать, какова высота и сколько пикселов необходимо иметь между строками. Чтобы заполнить эту потребность, AWT включает класс FontMetrics, который инкапсулирует различную информацию о шрифте. Некоторые термины, используемые при описании шрифтов:

- Высота (Height) – размер (от верха до низа) самого высокого символа в шрифте.
- Базовая линия (Baseline) – линия, по которой выровнен низ всех символов (не считая десцендера).
- Высота надстрочного элемента, асцендер (Ascent) – расстояние от базовой линии до верха символа.
- Высота подстрочного элемента, десцендер (Descent) – расстояние от базовой линии до низа символа.
- Интерлиньяж (Leading) – расстояние между самым низом одной строки текста и самым верхом следующей строки.

Класс FontMetrics определяет несколько методов, которые помогают управлять текстовым выводом.

Таблица 5.3. Некоторые методы класса FontMetrics.

Метод	**Описание**
int bytesWidth(byte b[], int start, int numBytes)	Возвращает ширину строки, состоящей из numBytes-символов, содержащихся в массиве b. Параметр start указывает номер начального символа этой строки в массиве b
int charWidth(char c[], int start, int numChars)	Возвращает ширину строки, состоящей из numChar-символов, содержащихся в массиве c. Параметр start указывает номер начального символа этой строки в массиве b
int charWidth(char c)	Возвращает ширину c
int charWidth(int c)	Возвращает ширину c
int getAscent()	Возвращает асцендер шрифта
int getDescent()	Возвращает десцендер шрифта
Font getFont()	Возвращает шрифт
int getHeight()	Возвращает высоту строки текст. Это значение можно

	использовать для вывода в окно многострочного текста
int getLeading()	Возвращает размер интерлиньяжа
int getMaxAdvance()	Возвращает ширину самого широкого символа. Возвращает –1, если это значение недоступно
int getMaxAscent()	Возвращает максимальный асцендер
int getMaxDescent()	Возвращает максимальный десцендер
int [] getWidths()	Возвращает ширины первых 256 символов
int stringWidth(String str)	Возвращает ширину строки, указанной в параметре str
String toString()	Возвращает строчный эквивалент вызывающего объекта

Лекция 6.

Работа с изображением.

Далее будем рассматривать AWT-класс Image и пакет java.awt.image. Вместе они поддерживают работу с изображениями (отображение и манипуляции с графическими изображениями). Под *изображением* понимают прямоугольный *графический объект.* Изображения являются ключевым компонентом Web-дизайна. Включение тега <img> в браузер Mosaic NCSA (National Center for Supercomputer Applications, Национальный Центр Суперкомпьютерных Приложений) привело к началу взрывного роста Web в 1993 г. Этот тег был использован, чтобы встраивать изображение в поток гипертекста. Java расширяет данную базовую концепцию, допуская программное управление изображениями. Java обеспечивает интенсивную поддержку работы с изображениями.

Изображения — это объекты класса Image, который является частью пакета java.awt. Для манипулирования изображениями используются классы пакета java.awt.image, который содержит большое количество классов и интерфейсов изображений. Рассмотреть их всех сразу невозможно. Поэтому мы сосредоточимся только на той части пакета, которая формирует основу работы с изображениями.

6.1.Форматы графических файлов.

Первоначально, Web-изображения могли быть только в формате GIF. Формат растровых изображений GIF (Graphics Interchange Format, формат обмена графическими данными) был создан в *CompuServe Incorporat*ion в 1987 г., для возможности просмотра встроенных изображений, что хорошо подходило для Internet. Каждое GIF-изображение может иметь не больше 256 цветов. Это ограничение заставило главных поставщиков браузеров в 1995 г. добавить поддержку изображений в формате JPEG. Формат JPEG (Joint Photographic Expert Group) был созданpгруппой фотографических экспертов для хранения изображений с полным цветовым спектром и непрерывным тоном. Эти изображения, если они созданы должным образом, могут иметь намного более высокую точность цветовоспроизведения и более высокую степень сжатия по сравнению с GIF-кодированием. В большинстве случаев вас не будет даже интересовать, какой формат вы используете в своих программах. В языке Java все различия в кодировании изображений скрыты за ясными и удобными интерфейсами их классов.

6.2.Создание, загрузка и просмотр изображений.

Существует три общие операции, которые используются для работы с любыми изображениями: создание, загрузка и просмотр изображения на экране. Класс Image языка Java имеет средства для создания нового *объекта изображения* и его загрузки, и средства, с помощью которых изображение можно отобразить на экране. Image обслуживает как изображения, находящиеся. в памяти, так и изображения, которые загружаются из внешних источников.

6.2.1.Создание объекта изображения

Можно было бы ожидать, что для создания изображения в памяти достаточно записать что-то вроде следующей операции:

```
Image test = new Image(200, 100);        // Ошибка — не работает!
```

Но это не так. Чтобы изображения стали видимыми, их нужно рисовать в окне. Однако класс Image не имеет достаточной информации для того, чтобы создать надлежащий формат данных для экрана. Поэтому класс Component (из пакета java.awt) содержит специальный "производственный" (factory) метод с именем createImage(), который используется для создания Image-объектов. (Напомним, что все AWT-компонент являются подклассами Component, поэтому все они поддерживают данный метод). Метод createImage() имеет две формы:

Image createImage (ImageProducer imgProd)

Image createImage (int width, int height)

Первая форма возвращает изображение, изготовленное параметром **imgProd,** который является объектом класса, реализующего интерфейс ImageProducer. Вторая форма возвращает пустое изображение, которое имеет указанную ширину и высоту. Например:

```
Canvas c = new Canvas();
Image test = c.createImage(200, 100);
```

Здесь создается экземпляр (объект) класса Canvas и затем вызывается производственный метод createImage(), чтобы фактически построитьт объект типа Image. В этом случае изображение будет пустым.

6.2.2.Загрузка изображения.

Другой способ получения изображения — его загрузка. Для этого используется метод getImage(), определенный классом Applet. Он имеет следующие формы:

Image getImage(URL ur1)

Image getImage(URL ur1, String imageName)

Первая версия возвращает Image-объект, который инкапсулирует изображение, найденное по (универсальному) адресу, указанному в параметре ur1. Вторая версия возвращает Image-объект, который инкапсулирует изображение, найденное по адресу, указанному в url, и имеющему имя, указанное в **imageName.**

6.2.3.Просмотр изображения.

Имея изображение, вы можете выводить его (на экран), используя метод drawImage(), который является членом класса Graphics. Он содержит несколько форм. Мы будем использовать метод в следующей форме:

boolean drawImage (Image imgObj, int left, int top, ImageObserver imgOb)

Он выводит изображение, переданное ему параметром **imgObj**, размещая его левый верхний угол с позиции, указанной в left и top. imgObj— ссылка на класс, который реализует интерфейс ImageObserver. Этот интерфейс реализуется всеми AWT-компонентами**.** *Наблюдатель изображения* (image obserчer) — это объект, который может контролировать изображение, пока оно загружается.

Наблюдение загрузки изображения довольно информативно, но было бы лучше, если бы вы использовали время загрузки изображения, чтобы что-то делать параллельно. Полностью сформированное изображение появляется на экране только в тот момент, когда оно целиком загружено. Для контроля загрузки изображения во время прорисовок экрана с другой информацией можно использовать интерфейс ImageObserver.

Лекция 7.

Интерфейс ImageObserver и класс MediaTracker.

7.1. Интерфейс ImageObserver.

ImageObserver — это интерфейс, используемый для приема уведомлений о том, как генерируются изображения. ImageObserver определяет только один метод: imageUpdate(). Использование наблюдателя изображения позволяет выполнять (параллельно с загрузкой изображения) другие действия, такие как показ индикатора хода работы (progress-индикатора) или дополнительного экрана, которые информируют вас о ходе загрузки. Подобный вид уведомления очень полезен, когда изображение загружается по сети, где

проектировщик содержимого редко принимает во внимание, что люди часто пробуют загружать апплеты через медленный модем. Метод ImageUpdate() имеет следующую общую форму:

boolean imageUpdate(Image imgObj, int flags, int left, int top, int width, int height).

Здесь i**mgQbj** — загружаемое изображение, а **flags** — целое число, которое сообщает состояние отчета обновления. Четыре целых параметра **left, top, width** и **height** представляют прямоугольник, который содержит различные значения в зависимости от передаваемых в **flags**-значений. ImageUpgate() должен возвратить false, если он завершил загрузку, и true, если еще имеется остаток изображения для обработки.

Параметр flags содержит один или несколько разрядных флажков, определенных как статические переменные внутри интерфейса ImageObserver. Эти флажки и информация, которую они обеспечивают, перечислены в табл. 7.1.

Таблица 7.1. Разрядные флажки параметра Flags метода ImageUpdate()

Флажок	**Значение**
WIDTH	Параметр width правилен и содержит ширину изображения
HEIGHT	Параметр *height* правилен и содержит высоту изображения
PROPERTIES	Свойства, связанные с изображением могут теперь быть получены через imgObj.getProperty ()
SOMEBITS	Получена следующая порция пикселов, необходимых для вывода изображения. Параметры left, top, width, height определяют прямоугольник, содержащий новые пикселы
FRAMEBITS	Получен полный фрейм, являющийся частью многофреймового изображения, которое было предварительно нарисовано. Данный фрейм может быть отображен. Параметры left, top, width и height не используются
ALLBITS	Изображение выведено целиком. Параметры left, top, width и height не используются
ERROR	Произошла ошибка с изображением, которое прослеживалось асинхронно. Изображение неполно и не может быть отображено. Никакая дальнейшая видеоинформация не будет получена. Для удобства будет также установлен флажок ABORT, чтобы указать, что производство изображения было прервано
ABORT	Изображение, которое прослеживалось асинхронно, было прервано прежде, чем оно было закончено. Однако, если ошибка не произошла, доступ к любой части данных изображения перезапустит производство изображения

Класс Applet имеет реализацию метода imageUpdate() интерфейса ImageObserver, который используется для перерисовки изображений во время их загрузки. Его можно переопределить в вашем классе, чтобы изменить поведение метода.

Простой пример метода imageUpdate():

```
public boolean imageUpdate(Image img, int flags, int x, int y, int w, int h) {
    ((flags & ALLBITS) == 0) {
        System.out.println("Загрузка изображения...");
        return true;
  } else {
        System.out.println("Загрузка изображения завершена. ");
        return false;
        }
}
```

7.2. Класс MediaTracker.

Интерфейс ImageObserver слишком сожный для понимания и управления загрузкой множественных изображений. Необходимо более простое решение, которое позволило бы программистам загружать все их изображения синхронно, не беспокоясь относительно imageUpdate(). К java.awt добавлен класс с именем MediaTracker. MediaTracker создает объект, который будет параллельно проверять состояние произвольного числа изображений.

Для использования MediaTracker создаем его новый экземпляр и применяем его метод addImage(), чтобы прослеживать состояние загрузки изображения. Общий формат addImage():

void addImage(Image imgObj, int imgID)

void addImage(Image imgObj, int imgID, int width, int height)

Здесь **imgObj** – прослеживаемое изображение**.** Его идентификационный номер передается в **imgID**. ID (идентификатор) номера не должны быть уникальными. Можно использовать номер с несколькими изображениями, как средство их идентификации в качестве части группы. Во второй форме параметры **width** и **height** определяют размеры отображаемого объекта.

Как только зарегистрировали изображение, можно проверить загружено ли оно, или можно ждать пока оно полностью загрузится. Для проверки состояния изображения вызывают метод checked(). Ее формат:

boolean checkID(int imgID)

Здесь **imgID** определяет ID изображения, которые хотим проверить. Метод возвращает true, если все изображения, которые имеют указанный идентификатор, были загружены (или, когда загрузка закончилась с ошибкой или пользователь выполнил аварийное завершение работы). Иначе он возвращает false. Чтобы увидеть, были ли все прослеживаемые изображения загружены, можно использовать метод checkAll().

MediaTracker следует применять при загрузке группы изображений.

Рассмотрим пример, который загружает слайд-показ с семью изобажениями и отображает прогресс полоску процесса загрузки.

```
/*
*<applet code=TrackedImageLoad.class width=300 height=400>
*<param name="img" value="1+2+3+4+5+6+7">
*</applet>
*/
import java.util.*;
import java.applet.*;
import java.awt.*;
public class TrackedImageLoad extends java.applet.Applet implements Runnable{
	MediaTracker tracker;
	int tracked;
	int frame_rate=5;
	int current_img=0;
	Thread motor;
	static final int MAXIMAGES=10;
	Image img[]=new Image[MAXIMAGES];
	String name[]=new String[MAXIMAGES];
	boolean stopFlag;

	public void init()
	{
		tracker=new MediaTracker(this);
		StringTokenizer st=new StringTokenizer(getParameter("img"),"+");
		while (st.hasMoreTokens()&&tracked<=MAXIMAGES){
```

```
                name[tracked]=st.nextToken();
                img[tracked]=getImage(getDocumentBase(),name[tracked]+".jpg");
                tracker.addImage(img[tracked],tracked);
                tracked++;
                }
        }

        public void paint(Graphics g){
                String loaded="";
                int donecount=0;
                for (int i=0;i<tracked;i++){
                  if(tracker.checkID(i,true)){
                        donecount++;
                        loaded+=name[i]+" ";
                        }
                }
            Dimension d =getSize();
                int w=d.width;
                int h=d.height;
                if (donecount==tracked){
                   frame_rate=1;
                        Image i=img[current_img++];
                        int iw=i.getWidth(null);
                        int ih=i.getHeight(null);
                        g.drawImage(i,(w-iw)/2,(h-ih)/2,null);
                        if (current_img>=tracked)
                                current_img=0;
                } else {
                   int x=w*donecount;
                        g.setColor(Color.black);
                        g.fillRect(0,h/3,x,16);
                        g.setColor(Color.white);
                        g.fillRect(x,h/3,w-x,16);
                        g.setColor(Color.black);
                        g.drawString(loaded,10,h/2);
                }
        }
public void start(){
   motor=new Thread(this);
                stopFlag=false;
                motor.start();
}
public void stop(){
          stopFlag=true;
        }
public void run(){
           motor.setPriority(Thread.MIN_PRIORITY);
                while (true){
                   repaint();
                        try{
                           Thread.sleep(1000/frame_rate);
                        } catch (InterruptedException e){};
```

```
                        if (stopFlag)
                                return;
                }
        }
}
```

Представленный пример создает новый MediaTracker в методе init() и затем добавляет каждое из именованных прослеживаемых изображений с помощью метода addImage(). В методе paint() на очередном из прослеживаемых изображений вызывается checkID(). Если нет, показывается простая прогресс-полоска числа загруженных изображений с именами полностью загруженных (отображаемых ниже этой полоски).

Лекция 8.

Интерфейс ImageProducer и ImageConsumer.

8.1.Интерфейс ImageProducer.

ImageProducer — это интерфейс для объектов, которые хотят производитьданные для изображений. Объект реализующий интерфейс ImageProducer, поставляет целочисленные или байтовые массивы, которые представляют данные изображения и производят Image-объекты. Существуют два *производителя изображений* (image produsers), содержащихся в java.awt.image: MemoryImageSource и FilteredImageSource

8.1.1.Производитель изображений MemoryImageSource

MemoryImageSource — это класс, который создает новый Image-объект из массива данных. Он определяет несколько конструкторов. Тот, который мы будем использовать, имеет следующую сигнатуру:

MemoryImageSource (int width, int height, int pixel [ ], int offset, int scanLineWidth)

Объект MemoryImageSource создается из массива целых чисел (в формате умалчиваемой цветовой модели RGB), указанного в параметре **pixel** (он-то и содержит данные для воспроизведения Image-объекта). В умалчиваемой цветовой модели пиксел — это целое число формата 0xAARRGGBB, где A — Alpha, R — Red, G — Green, и B — Blue. Значение Alpha представляет степень прозрачности пиксела (0 — полностью прозрачный, 255 — полностью непрозрачный). Ширина и высота результирующего изображения передается в параметрах **width** и **height.** Исходную точку для начала чтения данных в массиве пикселов задает параметр **offset**. Ширина строки сканирования (которая часто совпадает с шириной изображения) задает параметр **scanLineWidth.**

Следующий короткий пример генерирует MemoryImagesource-объект, используя разновидность простого алгоритма (поразрядное исключающее ИЛИ (x,y)-координат каждого пиксела).

```
/*
* <applet code="MemoryImageGenerator" width=256 height=256>
* </applet>
*/
import java.applet.*;
import java.awt.*;
import java.awt.image.*;
public class MemoryImageGenerator extends Applet {
        Image img;
        public void init () {
                Dimension d = getSize();
```

```
		int w = d.width;
		int h = d.height;
		int pixels[] = new int[w * h];
		int i =0;
		for (int y=0; y<h; y++) {
			for (int x=0; x<w; x++) {
				int r = (x^y)&0xff;
				int g = (x*2^y*2)&0xff;
				int b = (x*4^y*4)&0xff;
				pixels[i++] = (255<<24) | (r<<16) | (g << 8) | b;
			}
		}
		img = createImage (new MemoryImageSource (w, h, pixels, 0, w));
	}
	public void paint(Graphics g) {
		g.drawImage (img, 0, 0, this);
	}
}
```

Данные для нового MemoryImageSource создаются в методе init(). Массив целых предназначен для хранения пикселных значений; данные генерируются во вложенных for-циклах, где значения r, g и b организуют сдвиги в пикселах массива pixels. Наконец вызывается метод createImage() с новым экземпляром MemoryImageSource, созданным из необработанных пикселных данных, в качестве последнего аргумента.

8.2.Интерфейс ImageConsumer

ImageConsumer — это абстрактный интерфейс для объектов, которые хотят получать пикселные данные изображений (скажем, от производителя) и поставлять их (скажем, на экран) уже как другой вид данных. Очевидно, что этот интерфейс является противоположностью интерфейса ImageProducer. Объект, который реализует интерфейс ImageConsumer, собирается создавать массивы **int** или **byte**, которые представляют пикселы Image-объекта. Рассмотрим класс PixelGrabber, который является простой реализацией интерфейса ImageConsumer.

8.2.1.Класс PixelGrabber.

Класс PixelGrabber определен в java.lang.image. Это инверсия класса MemoryImageSource. Вместо построения изображения из массива пикселных значений, он берет существующее изображение и строит из него массив пикселов**.** Для использования PixelGrabber вы сначала организуете int-массив достаточно большой, чтобы содержать данные пикселов, и затем создаете экземпляр PixelGrabber, передавая ему прямоугольную область, которую вы хотите преобразовать в пикселное представление. Наконец, вы вызываете метод grabPixels() этого экземпляра.

Конструктор PixelGrabbar, который будем использовать мы, имеет следующую форму:

PixelGrabber (Image imgObj, int left, int top, int width, int height,
int pixel[], int offset, int scanLineWidth)

Здесь **imgObj** — объект, чьи пикселы преобразуются. Значения **left** и **top** определяют левый верхний угол прямоугольника, а **width** и **height** — его размеры, из которого будут получены пикселы. Пикселы будут сохранены в массиве **pixel,** со смещением **offset.** Ширину строки сканирования (которая часто такая же, как ширина изображения) задают в **scanLineWidth.**

Метод grabPixels() определяется с такими сигнатурами:

boolean grabPixels ()
throws InterruptedException

boolean grabPixels (long milliseconds)
throws InterruptedException

Оба метода возвращают true, если завершаются успешно, и false — в противном случае. Во второй форме параметр **milliseconds** определяет, как долго метод будет ожидать пикселы.

Далее показан пример, который "захватывает" пикселы изображения и затем создает гистограмму их яркости. Под *гистограммой яркости* здесь понимается распределение количества пикселов с определенной яркостью по всем значениям шкалы яркости (от 0 до 255). После того как апплет рисует изображение, он выводит (поверх этого изображения) гистограмму его яркости.

```
/*
* <applet code=HistoGrab width=341 height=400>
* <param name=img value=vermeer. jpg>
* </applet>
*/
import java.applet.*;
import java.awt.*;
import java.awt.image.*;
public class HistoGrab extends Applet {
	Dimension d;
	Image img;
	int iw, ih;
	int pixels[];
	int w, h;
	int hist [] = new int [256];
	int max_hist = 0;
	public void init() {
		d = getSize ();
		w = d.width;
		h = d.height;
		try {
			img = getImage (getDocumentBase (), getParameter ("img") );
			MediaTracker t = new MediaTracker(this);
			t.addImage (img, 0);
			t.waitForID (0);
			iw = img.getWidth(null);
			ih = img.getHeight(null);
			pixels = new int[iw * ih];
			PixelGrabber pg = new PixelGrabber(img, 0, 0, iw, ih, pixels, 0, iw);
			pg.grabPixels ();
		} catch (InterruptedException e) {};
		for (int i=0; i<iw*ih; i++) {
			int p = pixels[i];
			int r = 0xff & (p >> 16);
			int g = 0xff & (p >> 8);
			int b = 0xff & (p);
			int y = (int) (.33 * r + .56 * g+ .11 * b);
			hist [y]++;
		}
		for (int i=0; i<256; i++) {
			if (hist[i] > max_hist)
```

```
                                max_hist = hist[i];
                        }
                }
        public void update () { }
        public void paint(Graphics g) {
                g.drawImage(img, 0, 0, null);
                int x = (w-256)/2;
                int lasty = h-h*hist[0]/max_hist;
                for (int i=0; i<256; i++, x++) {
                        int y = h-h*hist[i]/max_hist;
                        g.setColor(new Color(i, i, i));
                        g.fillRect (x, y, 1, h);
                        g.setColor(Color.red);
                        g.drawLine (x-1, lasty, x, y);
                        lasty = y;
                }
        }
}
```

Лекция 9.

Класс ImageFilter

На базе интерфейсов ImageProducer и ImageConsumer (и их конкретных классов MemoryImageSource и PixelGrabber), можно создатьпроизвольный набор преобразующих фильтров, каждый из которых берет источник пикселов, модифицирует сгенерированные ими пикселы и передает произвольному потребителю. Этот механизм аналогичен способу, с помощью которого создаются конкретные абстрактные классы ввода.вывода InputStream, OutputStream, Reader и Writer**.** Такя поточная модель изображений завершается введением класса ImageFilter. Класс ImageFilter пакета java.awt.image имеет несколько подклассов — AreaAveragingScaleFilter, CropImageFilter, ReplicateScaleFilter и RGBImageFilter. Кроме того, в пакете имеется специальная реализация интерфейса ImageProducer с именем FilteredImageSource. Объекты класса FilteredImageSource производят фильтрованные пикселы изображений, генерируемых объектами класса ImageProducer. Для фильтрации в них используются объекты одного из указанных выше подклассов ImageFilter. Экземпляр класса FilteredImageSource передается (в качестве аргумента) в вызов метода createImage(), который используется в качестве производителя фильтрованных (преобразованных) изображений. Мы рассмотрим два специальных фильтра: CropImageFilter и RGBImageFilter.

9.1.Фильтр CropImageFilter

Фильтр CropImageFilter просто вырезает из исходного изображения небольшую прямоугольную область. Его удобно использовать, например, когда нужно работать не с целым большим изображением, а с более мелкими его частями. Если каждое подизображение имеет один и тот же размер, то можно легко извлекать их, используя для разборки блока изображения фильтр CropImageFilter. Далее показан пример, в котором одиночное изображение делится фильтром на шестнадцать одинаковых прямоугольных частей и затем эти не перекрывающиеся изображения скрэмблируются (перемешиваются случайным образом) путем 32-кратного обмена пар, выбираемых случайно из шестнадцати вырезанных изображений.

```
/*
 * <applet code=TileImage width=288 height=399>
 * <param name=img value=picasso.jpg>
 * </applet>
 */
```

```
import java.applet.*;
import java.awt.*;
import java.awt.image.*;
public class TileImage extends Applet {
      Image img;
      Image cell[] = new Image[4*4];
      int iw, ih;
      int tw, th;
      public void init() {
                try {
                        img = getImage(getDocumentBase(), getParameter("img"));
                        MediaTracker t = new MediaTracker(this);
                        t.addImage (img, 0);
                        t.waitForID(0);
                        iw = img.getWidth(null);
                        ih = img.getHeight(null);
                        tw = iw/4;
                        th = ih/4;
                        CropImageFilter f;
                        FilteredImageSource fis;
                        t= new MediaTracker(this);
                        for (int y=0; y<4; y++) {
                                for (int x=0; x<4; x++) {
                                        f = new CropImageFilter(tw*x, th*y, tw, th);
                                        fis = new FilteredImageSource(img.getSource(), f);
                                        int i = y*4+x;
                                        cell[i] = createImage(fis);
                                        t.addImage(cell[i], i);
                                }
                  }
                  t.waitForAll();
                  for (int i=0; i<32; i++) {
                          int si = (int)(Math.random() * 16);
                          int di = (int)(Math.random() * 16);
                          Image tmp = cell[si];
                          cell[si] = cell[di];
                          cell[di] = tmp;
                  }
          } catch (InterruptedException e) { };
    }
    public void update(Graphics g) {
          paint (g);
    }
    public void paint(Graphics g) {
          for (int y=0; y<4; y++) {
                        for (int x=0; x<4; x++) {
                           g.drawImage(cell[y*4+x], x * tw, y * th, null);
                  }
          }
    }
}
```

9.2.Фильтр RGBlmageFilter

Фильтр RGBImageFilter используется для попикселного преобразования одного изображения в другое, трансформируя по пути цвет пикселов. Данный фильтр можно использовать для прояснения изображения, увеличения его контраста, или даже для преобразования цветного изображения к полутоновому.

Для демонстрации RGBImageFilter рассмотрим несколько усложненный пример, который использует динамическую встроенную стратегию для обрабатывающих изображения фильтров. Создается интерфейс для обобщенной фильтрации изображения так, чтобы апплет мог просто загружать эти фильтры, основываясь на HTML-тегах <param>, без необходимости знать подробности обо всех фильтрах изображений. Этот пример состоит из главного апплета-класса с именем ImageFilterDemo, интерфейса с именем PlugInFilter и сервисного класса с именем LoadedImage, который инкапсулирует некоторые из методов класса MediaTracker, уже использовавшихся в предыдущих лекциях. Кроме того, кроме того в программу включены три фильтра — Grayscale, Invert и Contrast, которые просто манипулируют цветовым пространством исходного изображения, используя RGBImageFilters, и еще два класса Blur и Sharpen, которые применяют более сложные фильтры “свертывания”, изменяющие данные пиксела, основываясь на пикселах, окружающих каждый пиксел исходных данных. Blur и Sharpen — это подклассы абстрактного класса-помощника называемого Convolver. Рассмотрим каждую часть нашего примера.

ImageFilterDemo.java

Класс ImageFilterDemo является каркасом апплета для других фильтров изображений. Он использует простой менеджер компоновки BorderLayout, с панелью в позиции **South**, содержащей кнопки, которые будут представлять каждый фильтр. Объект Label занимает слот North для информационных сообщений о ходе работы фильтра. Изображение (которое инкапсулировано в Canvas подклассе LoadedImage) размещается в слоте **Center.** Будем анализировать кнопки фильтров вне параметра filters тега <param> (где они отделены значками +) — с помощью класса StringTokenizer.

Метод ActionPerformed() интересен тем, что он использует метку кнопки как имя класса фильтра, который он пробует загрузить с помощью метода newInstance():

pif = (PlugInFilter) Class.forName(a).newInstance();

Данный метод устойчив и выбирает адекватное действие, даже если кнопка не соответствует подходящему классу, реализующему PlugInFilter.

```
/*
*<applet code=ImageFilterDemo.class width=300 height=300>
*<param name=img value=143-a1.jpg>
*<param name=filters value="Grayscale+Invert+Contrast+Blur+Sharpen">
*</applet>
*/
import java.awt.*;
import java.applet.*;
import java.awt.event.*;
import java.util.*;

public class ImageFilterDemo extends Applet implements ActionListener{
        Image img;
        PlugInFilter pif;
        Image fimg;
        Image curImg;
        LoadedImage lim;
        Label lab;
```

```
	Button reset;

	public void init(){
		setLayout(new BorderLayout());
		Panel p=new Panel();
		add(p,BorderLayout.SOUTH);
		reset=new Button("Reset");
		reset.addActionListener(this);
		p.add(reset);
		StringTokenizer st=new StringTokenizer(getParameter("filters"),"+");
		while(st.hasMoreTokens()){
			Button b=new Button(st.nextToken());
			b.addActionListener(this);
			p.add(b);
		}
		lab=new Label ("");
		add(lab,BorderLayout.NORTH);

		img=getImage(getDocumentBase(),getParameter("img"));
		lim=new LoadedImage(img);
		add(lim,BorderLayout.CENTER);
	}

	public void actionPerformed(ActionEvent ae){
		String a="";
		try{
			a=(String)ae.getActionCommand();
			if (a.equals("Reset")){
				lim.set(img);
				lab.setText("Normal");
			}
			else {
				pif=(PlugInFilter) Class.forName(a).newInstance();
				fimg=pif.filter(this,img);
				lim.set(fimg);
				lab.setText("Filtered: "+a);
			}
			repaint();
		}catch (ClassNotFoundException e){
		      lab.setText(a+" not found");
				lim.set(img);
				repaint();
			}catch (InstantiationException e){
   lab.setText("could't new "+a);
			}catch (IllegalAccessException e){
			   lab.setText("no access: "+a);
			};
	}
}
```

PlugInFilter.java

PlugInFilter – это простой интерфейс, используемый для абстрактной фильтрации изображений. Он имеет только один метод filter(), который берет апплет и исходное изображение и возвращает новое изображение, которое было отфильтровано некоторым способом.

```
interface PlugInFilter{
   java.awt.Image filter(java.applet.Applet a,java.awt.Image in);
}
```

LoadedImage.java

LoadedImage – это удобный Canvas подкласс, который берет изображение во время конструирования и синхронно загружает его, используя загрузчик MediaTracker. LoadedImage ведет себя должным образом под управлением менеджера компановки LayoutManager потому, что он переопределяет методы getPreferredSize() и getMinimumSize(). Он также определяет специальный установочный метод set(). С его помощью можно добиться такого вида нового изображения (Image-объекта), чтобы оно могло отображаться в данном (пустом) окне (Canvas-объекте). Именно так фильтрованное изображение отображается после того, как подключение заканчивается.

```
import java.awt.*;
public class LoadedImage extends Canvas{
        Image img;
        public LoadedImage(Image i){
                set(i);
        }
        void set(Image i){
                MediaTracker mt=new MediaTracker(this);
                mt.addImage(i,0);
                try{
                        mt.waitForAll();
                } catch (InterruptedException e){};
                img=i;
                repaint();
   }

        public void paint(Graphics g){
                if (img==null) {
                        g.drawString("no image", 10, 30);
                }else {
                        g.drawImage(img,60,40,this);
                }
        }
        public Dimension getPreferredSize(){
                return new Dimension(img.getWidth(this),img.getHeight(this));
        }
   public Dimension getMinimumSize(){
           return getPreferredSize();
        }
}
```

Grayscale.java

Фильтр Grayscale – это подкласс фильтра RGBImageFilter. Данная формулировка означает, что Grayscale может использоваться как ImageFilter-параметр для конструктора FilteredImageSource. Тогда все что нужно сделать это переопределить метод filterRGB() так, чтобы изменить входящие значения цветов. Он берет красные, зеленые и синие значения и вычисляет яркость пиксела, используя NTSC (Национальный Телевизионный Комитет Стандартов) фактор преобразования "цвет-яркость". Затем он просто возвращает серый пиксел, который имеет ту же яркость, что и цветной источник.

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

class Grayscale extends RGBImageFilter implements PlugInFilter{
	public Image filter(Applet a, Image in){
		return a.createImage(new FilteredImageSource(in.getSource(),this));
	}
	public int filterRGB(int x,int y, int rgb){
		int r=(rgb>>16)&0xff;
		int g=(rgb>>8)&0xff;
		int b=rgb&0xff;
		int k=(int)(.56*g+.33*r+.11*b);
		return (0xff00000|k<<16|k<<8|k);
	}
}
```

Invert.java

Фильтр Invert разбирает пиксел, отбирая красные, зеленые и синие каналы, и затем инвертирует их, вычитая их из 255. Эти инвертированные значения упаковываются обратно в значение пиксела.

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

class Invert extends RGBImageFilter implements PlugInFilter{
	public Image filter(Applet a, Image in){
		return a.createImage(new FilteredImageSource(in.getSource(),this));
	}
	public int filterRGB(int x,int y, int rgb){
		int r=0xff-(rgb>>16)&0xff;
		int g=0xff-(rgb>>8)&0xff;
		int b=0xff-rgb&0xff;
		return (0xff00000|r<<16|g<<8|b);
	}
	}
```

Contrast.java

Фильтр Contrast очень похож на Grayscale, за исключением того, что его переопределение метода filterRGB() немного сложнее. Алгоритм, используемый им для улучшения контраста, берет красные, зеленые и синие значения отдельно и умножая их на 1.2, если они ярче, чем 128. Если яркость ниже 128, то они делятся на 1.2. С помощью

метода multclamp() обработанные таким способом значения, если нужно, ограничиваются величиной 255.

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

public class Contrast extends RGBImageFilter implements PlugInFilter{
	public Image filter(Applet a, Image in){
		return a.createImage(new FilteredImageSource(in.getSource(),this));
	}
	private int multclamp(int in,double factor){
		in=(int)(in*factor);
		return in>255?255:in;
	}
	double gain=1.2;
	private int cont(int in){
		return (in<128)?(int)(in/gain):multclamp(in,gain);
	}
	public int filterRGB(int x,int y, int rgb){
		int r=cont((rgb>>16)&0xff);
		int g=cont((rgb>>8)&0xff);
		int b=cont(rgb&0xff);
		return (0xff00000|r<<16|g<<8|b);
	}
	}
```

Convolver.java

Абстрактный класс Convolver базовую обработку фильтра свертывания, реализуя интерфейс потребителя ImageConsumer для перемешивания исходных пикселов в массив с именем imgpixels. Он также создает второй массив с именем newimgpixels для фильтрованных данных. Фильтры свертывания производят выборку маленького прямоугольника пикселов вокруг каждого пиксела в изображении, называемого *ядром свертывания*. Эта область (3*3 пиксела в данной демонстрации) используется для принятия решения, как следует изменить центральный пиксел в этой области.

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

abstract class Convolver implements ImageConsumer,PlugInFilter{
	int width,height;
	int imgpixels[],newimgpixels[];
	abstract void convolve();
	public Image filter(Applet a, Image in){
		in.getSource().startProduction(this);
		waitForImage();
		newimgpixels=new int[width*height];
		try {
			convolve();
		}catch (Exception e){
			System.out.println("Convolver failed: "+e);
			e.printStackTrace();
```

```
		}
		return a.createImage(new
MemoryImageSource(width,height,newimgpixels,0,width));
	}
	synchronized void waitForImage(){
		try{wait();}catch (Exception e){};
	}
	public void setProperties(java.util.Hashtable dummy){}
	public void setColorModel(ColorModel dummy){}
	public void setHints(int dummy){}
	public synchronized void imageComplete(int dummy){
	notifyAll();
	}
	public void setDimensions(int x,int y){
		width=x;
		height=y;
		imgpixels=new int[x*y];
	}
	public void setPixels(int x1,int y1, int w, int h, ColorModel model,byte pixels[],int off,int
scansize){
		int pix,x,y,x2,y2,sx,sy;
		x2=x1+w;
		y2=y1+h;
		sy=off;
		for(y=y1;y<y2;y++){
			sx=sy;
			for(x=x1;x<x2;x++){
				pix=model.getRGB(pixels[sx++]);
				if((pix&0xff00000)==0)
					pix=0x00fffff;
				imgpixels[y*width+x]=pix;
			}
			sy+=scansize;
		}
	}

}
```

Blur.java

Фильтр Blur – это подкласс Convolver. Он просто пробегает через каждый пиксел в исходном изображения imgpixels и вычисляет среднее значение по всей окружающей его области 3*3. Результирующий пиксел помещается в соответствующую позицию массива newimgpixels.

```
abstract class Blur extends Convolver{
	public void convolve(){
		for (int y=1;y<height-1;y++){
			for(int x=1;x<width-1;x++){
				int rs=0;
				int gs=0;
				int bs=0;
```

```
                        for(int k=-1;k<=1;k++){
                            for (int j=-1;j<=1;j++){
                                int rgb=imgpixels[(y+k)*width+x+j];
                                int r=(rgb>>16)&0xff;
                                int g=(rgb>>8)&0xff;
                                int b=rgb*-0xff;
                                rs+=r;
                                gs+=g;
                                bs+=b;
                            }
                        }
                    rs/=9;
                    gs/=9;
                    bs/=9;
                    newimgpixels[y*width+x]=(0xff00000|rs<<16|gs<<8|bs);
                    }
            }
    }
}
```

Sharpen.java

Фильтр Sharpen – также подкласс Convolver и является инверсией Blur. Он пробегает через каждый пиксел в исходном массиве изображения imgpixels и вычисляет среднее значение по окружающему 3*3 блоку пикселов, не считая центральный. Соответствующие выводные пикселы в newimgpixels различаются между собой (в зависимости от окружения центрального пиксела) добавляемыми в них усреднениями. Грубо говоря, если пиксел в тридцать раз ярче, чем его окружение, подобная фильтрация делает примерно тридцать окружающих пикселов ярче. Если, однако, он в десять раз темнее, то примерно десять окружающих пикселов станут более темными. Это приводит к подчеркиванию граней изображения, оставляя гладкие области неизменными.

```
abstract class Sharpen extends Convolver{
    private final int clamp(int c){
        return (c>255?255:(c<0?0:c));
    }
    public void convolve(){
        int r0=0,g0=0,b0=0;
        for (int y=1;y<height-1;y++){
            for(int x=1;x<width-1;x++){
                int rs=0;
                int gs=0;
                int bs=0;
                for(int k=-1;k<=1;k++){
                    for (int j=-1;j<=1;j++){
                        int rgb=imgpixels[(y+k)*width+x+j];
                        int r=(rgb>>16)&0xff;
                        int g=(rgb>>8)&0xff;
                        int b=rgb*-0xff;
                        if (j==0&&k==0){
                            r0+=r;
                            g0+=g;
```

```
                                        b0+=b;
        }else {
                                        rs+=r;
                                        gs+=g;
                                        bs+=b;
                                  }
                              }
                    }
                    rs>>=3;
                    gs>>=3;
                    bs>>=3;
                    newimgpixels[y*width+x]=(0xff00000|clamp(r0+r0-
rs)<<16|clamp(g0+g0-gs)<<8|clamp(b0+b0-bs));
                }
        }
    }
}
```

Лабораторная работа №1.

Основы оконной графики.

Цель. Формирование навыков работы с окнами (создание окон, создание фрейм-окон в апплете).

1. Создайте простое окно приложения.

Чтобы создать окно, достаточно сделать свой класс расширением класса Frame.

```
import java.awt.*;
class TooSimpleFrame extends Frame{
  public static void main(String[] args){
    Frame fr = new TooSimpleFrame();
  fr.setSize(400, 150);
  fr.setVisible(true);
  }
}
```

Класс TooSimpleFrame обладает всеми свойствами класса Frame, являясь его расширением. В нем создается экземпляр окна fr, и устанавливаются размеры окна на экране— 400*150 пикселов— методом setSize(). Если не задать размер окна, то на экране появится окно минимального размера — только строка заголовка. Конечно, потом его можно растянуть с помощью мыши до любого размера.

Затем окно выводится на экран методом setVisible(true). Дело в том, что, с точки зрения библиотеки AWT, создать окно значит выделить область оперативной памяти, заполненную нужными пикселами, а вывести содержимое этой области на экран — уже другая задача, которую и решает метод setVisible(true).

Конечно, такое окно непригодно для работы. Не говоря уже о том, что у него нет заголовка и поэтому окно нельзя закрыть. Хотя его можно перемещать по экрану, менять размеры, сворачивать на панель задач и раскрывать, но команду завершения приложения мы не запрограммировали. Окно нельзя закрыть ни щелчком кнопки мыши на кнопке с крестиком в правом верхнем углу окна, ни комбинацией клавиш <Alt>+<F4>. Приходится завершать работу приложения средствами операционной системы, например, комбинацией клавиш <Ctrl>+<C>.

2. Добавьте в программу заголовок окна и обращение к методу позволяющему завершить приложение.

```
import java.awt.*;
import java.awt.event.*;
class SimpleFrame extends Frame{
  SimpleFrame(String s){
    super (s);
    setSize(400, 150);
    setvisible(true);
    addWindowListener(new WindowAdapter(){
      public void windowClosing(WindowEvent ev){
        System.exit (0);
      }
    });
  }
  public static void main(String[] args){
    new SimpleFrame(" Моя программа");
  }
}
```

В программу добавлен конструктор класса SimpleFrame, обращающийся к конструктору своего суперкласса Frame, который записывает свой аргумент s в строку заголовка окна.

В конструктор перенесена установка размеров окна, вывод его на экран и добавлено обращение к методу addWindowListener (), реагирующему на действия с окном. В качестве аргумента этому методу передается экземпляр безымянного внутреннего класса, расширяющего класс WindowAdapter. Этот безымянный класс реализует метод windowClosing (), обрабатывающий попытку закрытия окна. Данная реализация очень проста — приложение завершается статическим методом exit() класса system. Окно при этом закрывается автоматически.

3. Создайте фрейм-окно в апплете.

Следующий апплет создает подкласс Frame с именем SampleFrame. Оконный объект этого подкласса создается внутри метода init() класса AppletFrame. Обратите внимание, что SampleFrame вызывает Frame-конструктор. Он создает стандартное фрейм-окно с заголовком, передаваемым конструктору в аргументе title. Методы start() и stop() окна апплета переопределяются так, чтобы они показывали и скрывали дочернее окно. Это приводит к автоматическому удалению окна при завершении апплета, закрытии окна или при переходе к другой странице браузера. При возврате браузера к данному апплету показ дочернеговь окна возобновляется.

```
// Создает дочернее фрейм-окно внутри апплета.
import java.awt .*;
import java.awt.event.*;
import java.applet .*;
/*
   <applet code="AppletFrame" width=300 height=50>
    </applet>
*/
// Создать подкласс Frame.
class SampleFrame extends Frame {
  SampleFrame(String title) {
    super (title);
    // создать объект для обработки window-событий
    MyWindowAdapter adapter = new MyWindowAdapter(this);
```

```
    // регистрировать его для приема этих событий
    addWindowListener(adapter);
   }
  public void paint(Graphics g){
    g.drawString("This is in frame window", 10, 40);
   }
}
class MyWindowAdapter extends WindowAdapter {
  SampleFrame sampleFrame;
  public MyWindowAdapter(SampleFrame sampleFrame) {
    this.sampleFrame = sampleFrame;
  }
  public void windowClosing(WindowEvent we) {
    sampleГxame.setVisible(false);
  }
}
// Создать фрейм-окно.
public class AppletFrame extends Applet {
  Frame f;
  public void init () {
    f = new SampleFrame("A Frame Window");
    f.setSize(250, 250);
    f.setVisible(true);
  }
  public void start() {
    f.setVisible(true);
  }
  public void stop() {
    f.setVisible(false);
  }
  public void paint(Graphics g) {
    g.drawString("This is in applet window", 10, 20);
  }
}
```

Лабораторная работа №2.

Работа с графикой.

Цель. Формирование навыков работы с графикой: рисование линий, прямоугольников, эллипсов и многоугольников.

1. Создайте следующий апплет.

```
import java.awt.*;
import java.applet.*;
/*
<applet code=”ColorDemo” width=400 heigth=200>
</applet>
*/
public class ColorDemo extends Applet{
    public void paint(Graphics g){
//Рисовать линии
g.drawLine(0, 0, 100, 100);
g.drawLine(0, 100, 100, 0);
```

```
//Рисовать прямоугольники
g.drawRect(10, 10, 60, 50);
g.fillRect(100, 10, 60, 50);
g.drawRoundRect(190, 10, 60, 50, 15, 15);
g.fillRoundRect(70, 90, 140, 100, 30, 40);
//Рисование эллипсов
g.drawOval(10, 10 ,50, 50);
            }
          }
```

2. Создайте апплет, который рисует снеговика, домик.
3. Создайте апплет, который выводит графики тригонометрических функций.

Лабораторная работа №3.

Работа с цветом.

Цель. Формирование простейших навыков работы с цветом.

Дополните программы предыдущей лабораторной работы, задав цвета, т.е. создайте цветные картинки.

Лабораторная работа №4.

Работа с текстом.

Цель. Формирование навыков работы с текстом: установка шрифта, вывод текста в окне апплета.

1. Создайте следующий апплет, который демонстрирует возможность вывода многострочного текста.

```
//
import java.applet.*;
import java.awt .*;
/*
<applet code="MultiLine" width=300 height=100>
</applet>
*/
public class MultiLine extends Applet {
     int curX=0, curY=0;      // текущая позиции
     public void init() {
        Font f = new Font(“SansSerif", Font.PLAIN, 12);
        setFont (f);
      }
public void paint(Graphics g) {
   FontMetrics fm = g.getFontMetrics();
   nextLine("This is on line one.", g);
   nextLine("This is on line two.", g);
   sameLine (" This is on same line. ", g);
   sameLine (" This, too.", g);
   nextLine("This is on line three. ", g);
}
    // Продвинуться к следующей строке.
void nextLine(String s, Graphics g) {
   FontMetrics fm = g. getFontMetrics ();
   curY += fm. getHeight ();
   curX = 0;
```

```
   g.drawString(s, curX, curY)
   curX = fm.stringWidth(s);
   //Показать на той же линии.
void sameLine (String s, Graphics g) {
      FontMetrics fm = g. getFontetrics ( );
      g.drawString(s, curX, curY);
      curX += fm.stringWidth(s);
  }
}
```

Чтобы начать вывод с конца предыдущего вывода на той же строке, вы должны знать длину (в пикселах) каждой отображаемой строки. Для получения этого значения вызовите метод stringWidth(). Значение можно использовать для продвижения координаты **X** при отображении очередной строки.

В следующем апплете показано, как можно вывести в окно множество строк текста. Кроме того, демонстрируется, как можно вывести несколько строк текста на одной строке. Обратите внимание на переменные curx и curY. Они следят за текущей позицией текстового вывода.

2. Создайте апплет выводящий текст, выровненный по центру.

```
// Выравнивает текст по центру.
import java.applet.*;
import java.awt.*;
/*
<applet code="CenterText" width=200 height=100> </applet>
*/
public class CenterText extends Applet {
        final Font f = new Font("SansSerif", Font.BOLD, 18);
        public void paint(Graphics g) {
                Dimension d = this.getSize();
                g.setColor(Color.white);
                g.fillRect(0, 0, d.width, d.height);
                g. setColor (Color.black);
                g.setFont (f);
                drawCenteredString("This is centered.", d.width, d.height, g);
                g.drawRect(0, 0, d.width-1, d.height-1);
        }

        public void drawCenteredString(String s, int w,int h, Graphics g) {
                FontMetrics fm = g.getFontMetrics();
                int x=(w-fm.stringWidth(s))/2;
                int y=(fm.getAscent()+(h-(fm.getAscent()+fm.getDescent()))/2);
                g.drawString(s, x, y);
        }
}
```

Здесь показан пример, в котором текст выравнивается по центру в окне апплета (по горизонтали и вертикали). Программа получает асцендер, десцендер и ширину строки и с их помощью вычисляет позицию, с которой нужно отобразить в центре окна.

3. Создайте апплет, выводящий следующую информацию.

Закон отладки
Всякая последняя ошибка является предпоследней

При этом слова “Закон отладки” - красного цвета, шрифт: полужирный, Serif, размером 16; а “ Всякая последняя ошибка является предпоследней ” - синего цвета, полужирный курсив, Serif, размером 12. Линия толщиной 3 пиксела, черного цвета. Это все в простом прымоугольнике.

Лабораторная работа №5.

Работа с изображением.

Цель. Формирование навыков работы с изображением: создание, загрузка и просмотр изображений.

1. Следующий апплет, загружает и выводит одиночное изображение.

```
/*
 * <applet code=”SimpleImageLoad” width=250 height=150>
 * <param name  =”img” value=”magnify.jpg”>
 * </applet>
 */
import java.awt.*;
import java.applet.*;
public class SimpleImageLoad extends java.applet.Applet
{
	Image img;
	public void init()
	{
		resize(300,300);
		img=getImage(getDocumentBase(),getParameter("img"));
	}

	public void paint(Graphics g)
	{
		g.drawImage(img,150,150,this);
		g.drawImage(img,0,0,this);
	}
}
```

Загружается файл magnify.jpg, но вы можете заменить его любым файлом в формате GIF или JPG (только удостоверьтесь, что он находится в одном каталоге с HTML-файлом, который содержит апплет).

В методе init() переменной img назначается изображение, возвращенное методом getImage(). Метод getImage() использует строку, возвращенную методом getParameter ("img"), как имя файла изображения. Это изображение загружается из URL-адреса, в который метод getDocumentBase() возвратил URL-адрес HTML-страницы с тегом данного апплета. Имя файла, возвращенное методом getParameter ("img"), исходит из тега <param name="img" value="magnify.jpg "> данного апплета. Этот тег является эквивалентом, првда немного более медленным, HTML-тега <img src="magnify.jpg" width=250 height=150>.

Когда этот апплет выполняется, он начинает загрузку img в методе init(). На экране можно видеть изображение по мере его загрузки из сети, потому что реализация

интерфейса ImageQbserver в классе Applet вызывает метод paint () каждый раз, когда прибывает следующая порция данных изображения.

2. Попробуйте вывести два и более изображения. Посмотрите что происходит если вы отображаете одно изображение поверх другого.

Список литературы

1. Аарон И.Волш Основы программирования на Java для World Wide Web.- IDG Books Worldwide,Inc.,1996, Издательство "Диалектика",1996.
2. Баженова И.Ю. Язык программирования Java. – АО "Диалог-МИФИ", 1997.
3. Баранова Е.В., Числер Н.А. Введение в программирование на языке Java: Учебное пособие. – СПб.: Изд-во «Союз», 2001.
4. Дейкстра Э. Дисциплина программирования. – М.: Мир, 1978.
5. Лапчик М.П. Методика преподавания информатики. – Свердловск: СГПИ, 1987.
6. Ноутон П., Шилдт Г. Java™2: Пер. с англ. – СПб.: БХВ-Петербург,2001.
7. Джон Родли Создание Java-апплетов. –The Coriolis Group,Inc.,1996, Издательство НИПФ "ДиаСофт Лтд.",1996.
8. http:// ru.sun.com/win/java/
9. http://www.machaon.ru/digest/java/
10. http://microsoft.com/java/
11. CD – Java-приложения/Учебник по Java