Лабораторная работа №11

Графика в Java

AWT и Swing

До версии Java 1.2 графический интерфейс пользователя (GUI) создавался на основании технологии AWT (Abstract Window Toolkit). В AWT каждой компоненте ставится в соответствие компонента операционной системы (ОС), в которой исполняется приложение. Связь осуществляется через т.н. peer-объекты, которые создаются глубоко внутри реализации GUI и пользователь их не «видит». Большая часть вызовов в этих peer-объектах обращена к «родным» (native) компонентам данной ОС и аппаратному обеспечению, соответственно, требуется реализация под них.

При отрисовке пользовательского интерфейса AWT используется подход «нарисуй себя сам». У каждой графической компоненты есть метод paint (Graphics), который вызывается для ее рисования. Система координат у этого графического контекста установлена так, что верхний левый угол окна рисования находится в точке [0,0] и ось Y направлена вниз.

В AWT отрисовка устроена так, что, если не переопределять метод paint, то в конечном итоге управление уйдет на уровень peer-объектов. Опора на native-компоненты ОС порождает ряд проблем. Первая – внешний вид приложений. Разные цветовые схемы, разный вид элементов и, что самое главное – разная компоновка окна. Все это способно крайне затруднить работу с приложением при переходе под другую ОС. Т.е. налицо противоречие главному принципу Java – WORA (Write Once, Run Anywhere) – «написать однажды, выполнять везде».

Для решения всех этих проблем был разработан Swing, построенный на основе JFC (Java Foundation Classes). Swing позволяет создавать пользовательский интерфейс, который способен выглядеть и вести себя одинаково во всех ОС. С другой стороны, при необходимости можно с легкостью переключиться на вид и поведение приложения, характерные для текущей ОС.

Начиная с версии Java 1.2 библиотека swing входит в стандартную поставку jre (java runtime environment) и может использоваться вместе с AWT, хотя это делать не рекомендуется.

Основой для Swing остается AWT, что видно из приведенной ниже схемы наследования классов:

Component(AWT) - > Container (AWT) -> JComponent (Swing) -> Компоненты Swing

Во многих последних версиях, как правило, используется экземпляр наследника класса Graphics – java.awt.Graphics2D, возможности которого существенно выше.

Component — это абстрактный класс, который инкапсулирует все атрибуты визуального интерфейса: обработка ввода с клавиатуры, управление фокусом, взаимодействие с мышью, уведомление о входе/выходе из окна, изменения размеров и положения окон, прорисовка своего собственного графического представления, сохранение текущего текстового шрифта, цветов фона и переднего плана.

Подклассы класса Component.

Container — это абстрактный подкласс класса Component, определяющий дополнительные методы, которые дают возможность помещать в него другие компоненты, что дает возможность построения иерархической системы визуальных объектов. Container отвечает за расположение содержащихся в нем компонентов с помощью интерфейса LayoutManager,.

Panel — это очень простая специализация класса Container. В отличие от последнего, он не является абстрактным классом. Поэтому о Panel можно думать, как о допускающем рекурсивную вложенность экранном компоненте. С помощью метода add в объекты Panel можно добавлять другие компоненты. После того, как в него добавлены какие-либо компоненты, можно вручную задавать их положение и изменять размер с помощью методов move, resize и reshape класса Component.

Window во многом напоминает Panel за тем исключением, что он создает свое собственное окно верхнего уровня. Чаще всего используется не непосредственно класс Window, а его подкласс Frame.

Frame — это как раз то, что обычно и считают окном на рабочей поверхности экрана. У объекта Frame есть строка с заголовком, управляющие элементы для изменения размера и линейка меню. Для того, чтобы вывести/спрятать изображение объекта Frame, нужно использовать методы show и hide.

Applet - один из подклассов класса Panel.

Dialog (JDialog) - специальный вид окон для создания диалоговых окон. Создаются для непосредственного ввода информации пользователем и удаляются после завершения процесса ввода. Диалоговые окна часто используют для вывода сообщений об ошибках или для запроса дополнительной информации, которую должен ввести пользователь. Обычно это "модальные" (от слова mode — режим) диалоговые окна. После вывода их на экран система переходит в специальный режим и не позволяет работать с другими окнами, пока это не будет закрыто.

В Swing контейнером верхнего уровня может быть javax.swing.JApplet, javax.swing.JWindow, javax.swing.JDialog, javax.swing.JFrame и javax.swing.JInternalFrame. Такой контейнер содержит всего одну корневую панель – экземпляр javax.swing.JRootPane.

JApplet, JWindow, JDialog и JFrame – компоненты Swing, отрисовка которых проходит на уровне peer-объектов. Все остальные компоненты унаследованы от javax.swing.JComponent.

Основное отличие Swing от AWT – **в Swing отрисовка компонент происходит исключительно в java-коде.**

А метод paint(Graphics) в Swing переопределен так, что в native-код управление не уходит. Он делегирует отрисовку трем protected-методам – paintComponent(Graphics), paintBorder(Graphics) и paintChildren(Graphics). Каждый из этих методов выполняет следующее:

- `paintComponent` – этот метод позволяет рисовать саму компоненту. Т.е. делает то, что в AWT делал `paint(Graphics`.
- `paintBorder` – рисует рамку компоненты. Это нововведение Swing – в AWT рамок у компоненты не было.

- `paintChildren` – рисует дочерние элементы. Это еще одно изменение поведения `paint(Graphics)` – в AWT он дочерние элементы не отрисовывал.

Класс Graphics

В классе Graphics существуют разнообразные методы для работы с графическими примитивами. Средства языка Java позволяют изменять цвет и создавать изображения линий, прямоугольников, эллипсов и многоугольников. В большинстве случаев методу paint передается объект класса Graphics. Для построения изображений необходимо обращаться к его методам. Можно также создать собственный объект этого класса, например, для ускорения вывода изображений при воспроизведении сложной анимации.

Некоторые методы класса Graphics:

clearRect - очищает указанный прямоугольник, заполняя цветом фона;
clipRect - задает область ограничения вывода;
copyArea - копирует область экрана;
create - создает новый объект, который является копией исходного объекта;
draw3DRect - рисует прямоугольник с объемным эффектом;
drawArc - рисует дугу текущим цветом;
drawBytes - рисует указанные байты текущим шрифтом и цветом;
drawChars - рисует указанные символы текущим шрифтом и цветом;
drawImage - рисует указанное изображение типа Image;
drawLine - рисует линию между точками;
drawOval - рисует овал внутри указанного прямоугольника текущим цветом;
drawPolygon - рисует многоугольник текущим цветом;
drawRect - Рисует контур прямоугольника текущим цветом;
drawRoundRect - Рисует контур прямоугольника с закругленными краями;
drawString - Рисует указанную строку текущим шрифтом и текущим цветом;
fill3DRect - раскрашивает цветом прямоугольник с объемным эффектом;
fillArc - заполняет дугу текущим цветом;
fillOval - заполняет овал текущим цветом;
fillPolygon - заполняет многоугольник текущим цветом;
fillPolygon - заполняет объект класса Polygon текущим цветом;
fillRect - заполняет прямоугольник текущим цветом;
fillRoundRect - заполняет прямоугольник с закругленными краями;
setPaintMode - устанавливает режим заполнения текущим цветом.

Ниже приведены примеры использования некоторые из этих методов.
Для изображения линии в метод drawLine необходимо передать четыре параметра: координаты (x1, y1) начала и (x2, y2) конца линии.
Пример:

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

public class Graphic extends Frame {
     public Graphic(){
          super("Графика");
          addWindowListener(new WindowAdapter()
          {public void windowClosing(WindowEvent e){
               System.exit(0);
          }});

          setSize(400,400);
          setVisible(true);
     }
public void paint(Graphics g) {
     for(int i=0; i<100; i+=10)
          g.drawLine (i, i, i+10, i) ;

}
public static void main(String[] args) {
     new Graphic();
}
}
```

Можно нарисовать и закрасить прямоугольник. Углы прямоугольника могут быть округлены, а изображение может быть псевдотрехмерным.

```
g.drawRect(left, top, width, height);
   // Рисование прямоугольника с верхним левым углом в
   // (left, top), и нижним правым углом в
   // (left+width, top+height).
g.fillRect(left, top, width, height);
   // Рисование такого же прямоугольника, как в drawRect,
   // но закрашенного цветом линии контура.
g.drawRoundRect(left, top, width, height,horizontal-radius,vertical-radius);
g.fillRoundRect(left, top, width, height,horizontal-radius, vertical-radius);
// Рисование обычного и закрашенного прямоугольников с
// закругленными углами.
// Радиусы закругления по горизонтали и вертикали
// определяются значениями horizontal-radius
```

```
// и vertical-radius.
g.draw3DRect (left, top, width, height, polarity) ;
// Рисование прямоугольника с затенением по краям для
// создания эффекта трехмерного изображения.
// Если значение polarity установлено в true, то
// прямоугольник "выступает" из экрана,
// а при false - "вдавлен" в экран.
```

Функции рисования прямоугольников содержат функции рисования линий и дуг, а также функции закрашивания (заполнения) объектов. Углы закругленного прямоугольника формируются из дуг эллипса. Сначала эллипс разделяется на четыре части, части "растаскиваются" и далее соединяются прямыми линиями. Каждый угол закругленного прямоугольника является четвертью эллипса, пропорции которого определяются шириной и высотой прямоугольника.

Метод drawOval рисует эллипс, а метод drawArc — дугу эллипса. Закрашивание производится соответственно методами fillOval и fillArc. Параметрами служат координаты наименьшего описанного прямоугольника. Эллипс определяется четырьмя параметрами: двумя координатами вершины описанного прямоугольника, его шириной и высотой.
Для drawArc существуют два дополнительных параметра, определяющих начало и длину дуги в угловых градусах. Длина дуги отсчитывается по направлению часовой стрелки от полудня, так что 90 градусов соответствуют цифре 3 на циферблате :-).
Пример:

```
public void paint(Graphics g){
for(int i=0; i<140; i+=20)
g.drawOval(i,5,15,45);
for(int i=0; i<140; i+=20)
g.fillOval(i,55,15,45);
for(int i=0; i<140; i+=20)
g.drawArc(i,100,20,45, i+10, i+100);
for(int i=0; i<140; i+=20)
g.fillArc(i, 150, 20, 45, i+10, i+100);
}
```

Перед выводом на экран строки текста с помощью drawString целесообразно создать объект Font, определяющий имя, стиль и размер шрифта для этой строки. Например:

```
Font k =new Font("TimesRoman", Font.PLAIN, 24);
g.setFont(k);
g.drawString("Hi!",10,20);
```

В первой строке применен конструктор класса Font для создания объекта k, который затем передается в объект класса Graphics. Первым параметром конструктора является имя шрифта.
С каждым шрифтом можно использовать следующие стили: Font.BOLD (полужирный), Font.ITALIC (курсив) или оба стиля одновременно с помощью команды:

```
Font k = new Font ("Courier", Font.ITALIC+Font.BOLD, 12).
```

Константы стилей определены в классе Font, фактически они являются целыми числами.

Метод drawString языка Java не имеет многих необходимых возможностей. При выводе строки на экран игнорируются символы перевода строки и возврата каретки, следовательно, необходимы дополнительные расчеты при размещении строк на экране. Поэтому существуют специальные методы для вывода нескольких строк. Объекты класса FontMetrics (возвращаемые функцией getFontMetrics класса Graphics) имеют методы, с помощью которых можно узнать размеры символов, межстрочный интервал и высочу строки. Верхняя линия строки определяется с учетом высоты прописных (больших) букв. Нижняя линия строки проходит по нижней точке символов типа "g". Межстрочный интервал задает расстояние между базовыми линиями двух соседних строк. А общая высота строки измеряется от нижней точки символов типа "y" до верхней точки прописных букв (например, "M").

Несколько методов из класса FontMetrics:

```
public int getLeading() - возвращает значение межстрочного интервала.
public int getAscent() -возвращает расстояние между базовой и верхней линиями строки.
public int getDescent() - возвращает расстояние между базовой и нижней линиями строки. (Нижняя линия определяется нижней
точкой символов типа "g").
public int getHeight() -возвращает сумму общей высоты строки и межстрочного интервала. public int charWidth(int ch) -
возвращает ширину символа ch.
public int stringWidth(String str) -возвращает ширину строки (сумма ширины всех символов).
public int charsWidth(char data [], int off, int len) - возвращает ширину массива символов (сумму ширины len символов
массива, начиная с off).
public int[] getWidths() - возвращает ширину каждого из 256 базовых символов.
```

Функции вычисления ширины особенно полезны при разработке текстовых редакторов или для красивого форматирования строки на экране.

Пример форматирования текста:

```
int w = 200;
int h = 150;
g.drawRect(10,10,w,h);
Font f=new Font("Courier", Font.PLAIN, 18);
g.setFont(f);
FontMetrics fm = g.getFontMetrics();
g.drawString("Courier", 11, 10+fm.getHeight());
f=new Font("TimesRoman", Font.ITALIC,12);
g.setFont(f);
```

```
fm=g.getFontMetrics();
g.drawString("TimesRoman", w-fm.stringWidth("TimesRoman"),10+fm.getHeight());
f=new Font("Arial", Font.PLAIN,18);
g.setFont(f);
fm=g.getFontMetrics();
g.drawString("Arial",10,h);
f=new Font("Dialog", Font.PLAIN,18);
g.setFont(f);
fm=g.getFontMetrics();
g.drawString("Dialog",w-fm.stringWidth("Dialog"),h);
}
```

Задание цвета

В языке Java для работы с цветом применяется базовая модель RGB, в которой каждый компонент цвета (красный, зеленый и голубой) задается целым числом в диапазоне от 0 до 255. Любой цвет определяется тремя числами, например: белый-(255,255,255), красный-(255,0,0), зеленый - (0,255,0), черный -(0,0,0).

Класс `java.awt.Color` является главным классом для работы с цветом. Просмотрев исходные тексты этого класса можно увидеть, что в языке Java поддерживается ряд стандартных цветов.

Стандартные цвета Java:

Цвет	Объект Java	R	G	B
Белый	Color.white	255	255	255
Светло-серый	Color.lightGray	192	192	192
Серый	Color.gray	128	128	128
Темно-серый	Color.darkGray	64	64	64
Черный	Color.black	0	0	0
Красный	Color.red	255	0	0
Зеленый	Color.green	0	255	0
Пурпурный	Color.magenta	255	0	255
Голубой	Color.cyan	0	255	255
Синий	Color.blue	0	0	255

В классе `Color` существуют три конструктора. Аргументами первого из них являются три целых числа в диапазоне от 0 до 255, определяющих интенсивность красного, зеленого и синего цветов.

```
Color lineColor=new Color(140,23,190);
```

Аргументы второго конструктора - три числа с плавающей точкой в диапазоне от 0.0 до 1.0, которые преобразуются к целым числам из диапазона 0-255.

Аргумент третьего конструктора - 32-разрядное число, в котором синему цвету соответствуют биты с 0 по 7, зеленому-с 8 по 15, а красному - с 16 по 32:

```
Color lineColor=new Color(1423424);
```

Класс Graphics использует объекты класса Color для определения цвета отображаемого объекта. Для вывода в цвете объектов, создаваемых такими методами, как drawOval() или drawRect(), следует использовать метод setColor(). Например, для того чтобы установить текущий цвет для рисования красным:

```
g.setColor(Color.red);
```

Текущий цвет можно получить, воспользовавшись функцией getColor().

Фон окна апплета по умолчанию серый. Для его изменения используется метод setBackground(), определенный в классе Component(подклассом которого является Applet). Существует парный ему метод getBackground(). Метод setForeground() действует на все объекты апплета, устанавливая для них заданный цвет.

Задание размеров графических изображений

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

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

Простые методы класса Graphics

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

drawLine

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

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

drawArc и fillArc

Форма методов drawArc и fillArc следующая:

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

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

drawPolyson и fillPolyson

Прототипы для этих методов:

drawPolygon(int[], int[], int)

fillPolygon(int[], int[], int)

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

Цвет

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

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

Color(int, int, int)

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

Color(int)

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

Color(float, float, float)

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

Методы класса Color

HSBtoRGB(float, float, float)

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

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

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

getRedQ, getGreenO, setBlue()

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

getRGB()

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

```
red = Oxff & (getRGB() >> 16);
green = Oxff & (getRGB() >> 8);
blue = Oxff & getRGB();
```

setPaintMode() и setXORMode(Color)

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

Шрифты

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

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

drawString

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

Использование шрифтов

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

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

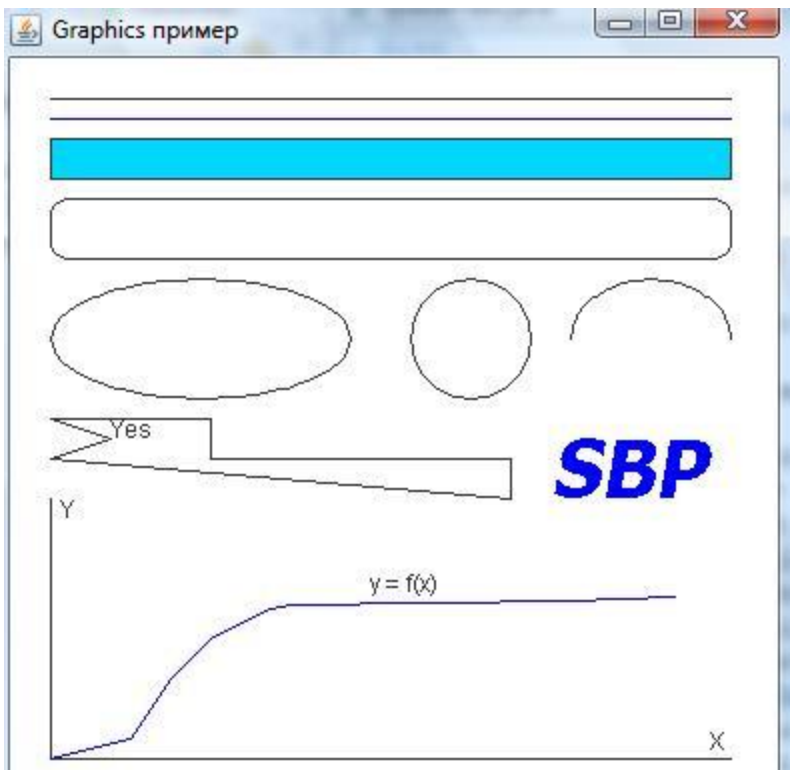
В настоящее время доступны следующие имена шрифтов: Dialog, Helvetica, TimesRoman, Courier и Symbol. Для указания стиля шрифта внутри данного семейства предусмотрены три статические переменные. — Font.PLAIN, Font.BOLD и Font.ITALIC, что соответствует обычному стилю, курсиву и полужирному.
Теперь давайте посмотрим на несколько дополнительных методов.
getFamily и getName
Метод getFamily возвращает строку с именем семейства шрифтов. С помощью метода getName можно получить логическое имя шрифта.
getSize
Этот метод возвращает целое число, представляющее собой размер шрифта в пунктах.
getStyle
Этот метод возвращает целое число, соответствующее стилю шрифта. Полученный результат можно побитово сравнить со статическими переменными класса Font: — PLAIN, BOLD и ITALIC.
isBold, isItalic, isPlain
Эти методы возвращают true в том случае, если стиль шрифта — полужирный (bold), курсив (italic) или обычный (plain), соответственно.

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

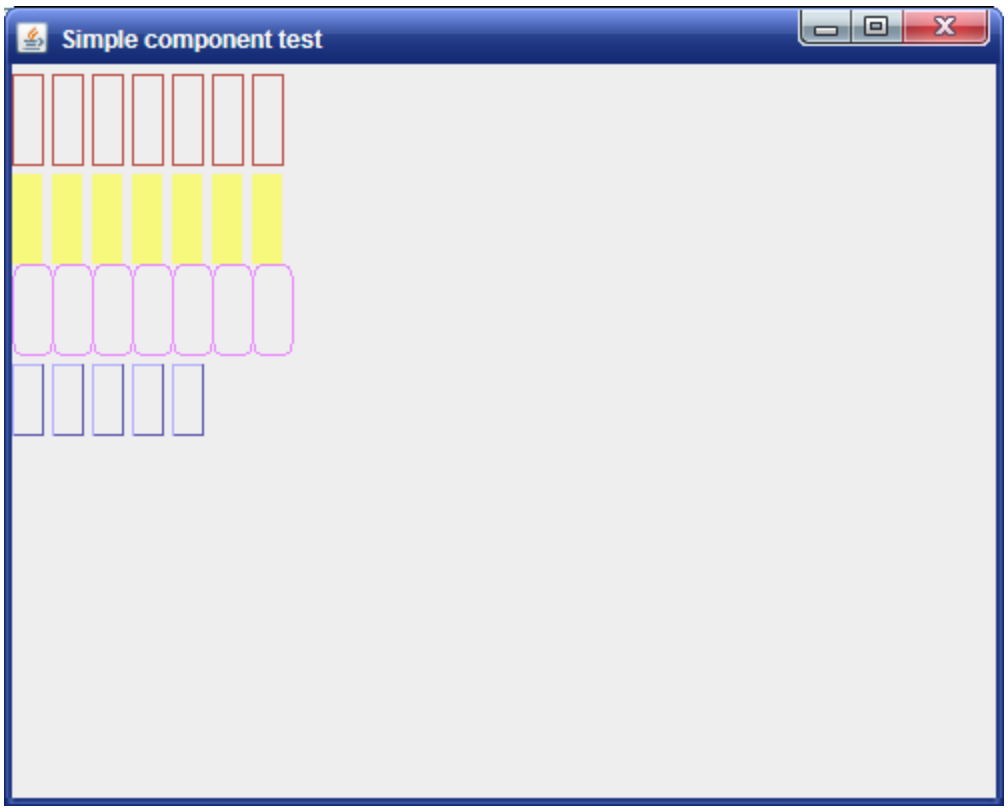
- *Высота* (height) — размер от верхней до нижней точки самого высокого символа в шрифте.
- *Базовая линия* (baseline) — линия, по которой выравниваются нижние границы символов (не считая снижения (descent)).
- *Подъем* (ascent) — расстояние от базовой линии до верхней точки символа.
- *Снижение* (descent)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

Использование FontMetrics
Ниже приведены некоторые методы класса FontMetrics:
stringWidth
Этот метод возвращает длину заданной строки для данного шрифта.
bytesWidth, charsWidth
Эти методы возвращают ширину указанного массива байтов для текущего шрифта.
getAscent, getDescent, getHeight
Эти методы возвращают подъем, снижение и ширину шрифта. Сумма подъема и снижения дают полную высоту шрифта. Высота шрифта — это не просто расстояние от самой нижней точки букв g и y до самой верхней точки заглавной буквы T и символов вроде скобок. Высота включает подчеркивания и т.п.
getMaxAscent и getMaxDescent
Эти методы служат для получения максимальных подъема и снижения всех символов в шрифте.

Задание 1. Нарисовать следующие фигуры (см.рис.):



Задание 2.
Используя оператор цикла вывести четыре ряда прямоугольников (см.рис.). Для каждого ряда задавать случайный цвет с помощью функции **random.**



Задание 3. Нарисовать следующие фигуры