**XXXX**

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

# Программирование графики на Java

Методические указания к лабораторной работе

Рязань 2010

УДК 681.3

Программирование графики на Java: методические указания к лабораторной работе / Рязан. гос. радиотехн. ун-т.; сост. А.А. Митрошин, А.В.Бакулев. – Рязань, 2010. – 16 с.

Содержат описание лабораторной работы, используемой в курсе «Инженерная и компьютерная графика».

Предназначены для студентов дневной и заочной форм обучения направления «Информатика и вычислительная техника».

Ил. 4. Библиогр.: 3 назв.

*Программирование, графика, Java.*

Печатается по решению редакционно-издательского совета Рязанского государственного радиотехнического университета.

Рецензент: кафедра САПР вычислительных средств Рязанского государственного радиотехнического университета (зав. кафедрой засл. деят. науки и техники РФ В.П.Корячко)

Программирование графики на Java

Составители: Митрошин Александр Александрович
Бакулев Александр Валерьевич

Редактор Р.К. Мангутова
Корректор С.В. Макушина
Подписано в печать 00.00.0000. Формат бумаги 60×84 1/16.
Бумага газетная. Печать трафаретная. Усл. печ. л. 1,0.
Уч-изд. л. 1,0. Тираж 100 экз. Заказ
Рязанский государственный радиотехнический университет.
390005, Рязань, ул. Гагарина, 59/1.
Редакционно-издательский центр РГРТУ.

## Лабораторная работа № 1

Для того чтобы начать процесс построения графических изображений в Java, необходимо создать окно, в котором и будет строиться изображение. Чаще всего для создания окна используется объект класса `JFrame`, определенный в пакете `javax.swing` [1]. Суперклассы `JFrame` показаны на рисунке 1.

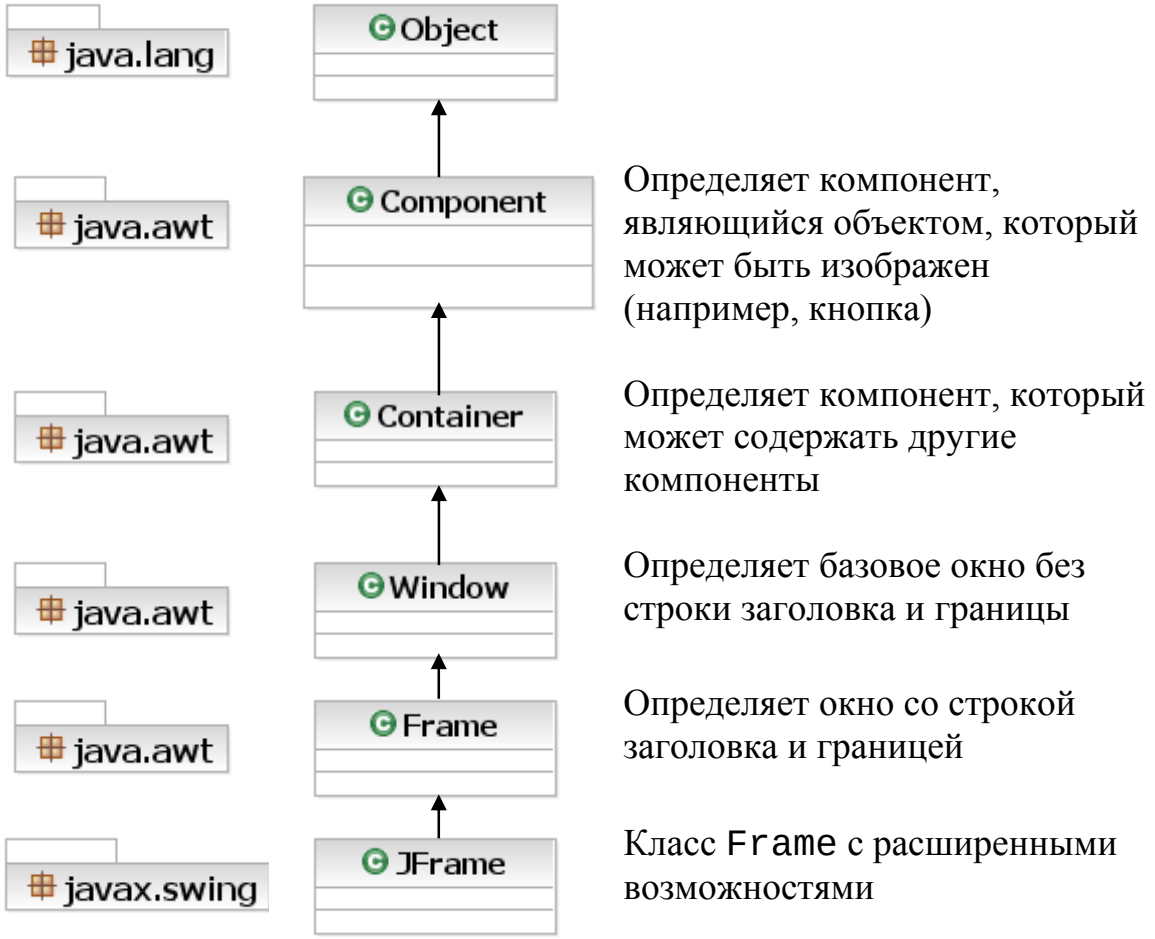



Рис. 1. Суперклассы класса `JFrame`

Класс `Component` является прародителем всех классов компонентов, то есть классов, объекты которых имеют графическое представление. Он определяет базовые свойства и методы, используемые всеми компонентами.

Класс `Container` предоставляет объекту класса `Component` возможность содержать другие компоненты. Так как `JFrame` имеет `Container` в качестве суперкласса, то есть и сам является контейнером, то объект класса `JFrame` может содержать другие компоненты.

Класс `Window` добавляет к классу `Component` методы, которые являются специфическими для окна, такие как возможность обрабатывать события, возникающие при взаимодействии с окном.

Класс `Frame` является исходным файлом в пакете `java.awt`, который предоставляет окно со строкой заголовка и границей.

Класс `JFrame` добавляет функциональность к классу `Frame` для поддержки более совершенных средств рисования и вывода других компонентов. Основное различие между объектом `JFrame` и объектом `Window` состоит в том, что объект `JFrame` представляет собой основное окно приложения, в то время как объект `Window` им не является. Объект `JFrame` необходимо иметь до создания объекта `Window`.

## Графический контекст

При создании графического компонента, то есть объекта класса `Component` или его потомка, автоматически формируется его ***графический контекст***. В контексте размещается область рисования и вывода текста и изображений. Контекст содержит текущий и альтернативный цвет рисования и цвет фона – объекты класс `Color`, текущий шрифт для вывода текста – объект класса `Font`. В контексте определена система координат, начало которой – точка с координатами (0, 0) расположена в верхнем левом углу области рисования, ось *Ox* направлена вправо, ось *Oy* – вниз (рис. 2).

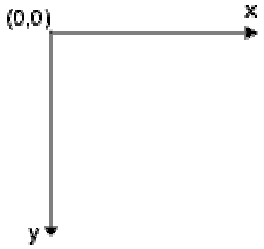



Рис. 2 Система координат графического контекста компонента

Управляет контекстом класс `Graphics` (или более новый класс `Graphics2D`, созданный в рамках библиотеки Java 2D, который мы рассмотрим на следующей лабораторной работе). Поскольку графический контекст зависит от конкретной графической платформы, то классы `Graphics` и `Graphics2D` сделаны абстрактными. Поэтому нельзя непосредственно создать экземпляры классов `Graphics` или `Graphics2D`. Однако каждая реализация виртуальной машины Java реализует методы этих классов, создает их экземпляры для компонента и предоставляет объект класса `Graphics` методом `getGraphics()` класса `Component`.

При создании контекста в нем задается текущий цвет (то есть цвет, которым будут рисоваться графические элементы) для рисования, обычно черный, и цвет фона области рисования (цвет, на котором будут рисоваться графические элементы), обычно белый или серый.

### Установка текущего цвета контекста

Существуют два режима рисования, которые различаются тем, какой цвет используется для вывода. Соответственно существуют и два метода, позволяющие устанавливать тот или иной режим рисования.

`setPaintMode()` – устанавливает режим рисования графического контекста, в котором цвет рисуемого изображения определяется текущим цветом, установленным в графическом контексте. Этот режим используется по умолчанию.

`setXORMode(Color c1)` – устанавливает режим рисования, в котором цвет получаемой фигуры определяется как результат выполнения побитовой операции XOR (сложение по модулю 2) между текущим цветом контекста и цветом `c1`, передаваемым в качестве параметра метода.

Изменить текущий цвет контекста можно методом `setColor(Color newColor)`, аргумент которого `newColor` – объект класса `Color`. Определить текущий цвет можно с помощью метода `getColor()`, возвращающего объект класса `Color`.

Класс `Color` предназначен для описания цвета. Класс имеет семь следующих конструкторов [2].

`Color(int red, int green, int blue)-` создаёт цвет, получающийся как смесь красной `red`, зелёной `green` и синей `blue` составляющей (цветовая модель RGB). Значение каждой составляющей может изменяться от 0 (отсутствие составляющей цвета) до 255 (максимальная интенсивность составляющей цвета). Например,
`Color pureRed(255, 0, 0)-` определяет ярко-красный цвет.

`Color(float red, float green, float blue)` – создаёт цвет, интенсивность составляющих которого определяется вещественными числами в диапазоне от 0 (отсутствие составляющей) до 1 (максимальная интенсивность составляющей).

`Color(int rgb)-` создаёт цвет, в котором все три составляющие задаются в одном целом числе. В битах 16 – 23 записывается красная составляющая, в битах 8 – 15 – зелёная, в битах 0 – 7 – синяя. Например, `Color c = new Color(0xFF002AFF)` определяет цвет, в котором красная составляющая определена с интенсивностью 0x00, зелёная – 0x2A, синяя – 0xFF.

Следующие три конструктора вводят четвертую составляющую цвета – альфа-канал, определяющий прозрачность цвета. Значение альфа-канала проявляется при наложении одного цвета на другой.

```
Color(int red, int green, int blue, int alpha);
Color(float red, float green, float blue, float alpha);
Color(int rgb, boolean hasAlpha);
```

Если значение альфа-канала равно 255 или 1.0, то цвет совершенно не прозрачен, цвет, который находится под этим цветом, не просвечивается сквозь него. Если значение альфа-канала равно 0 или 0.0, то цвет абсолютно прозрачен, - для каждого пикселя виден только цвет, лежащий под ним. Промежуточные значения позволяют создавать полупрозрачные изображения, сквозь которые просвечивается фон.

Конструкторы, в которых значение альфа-канала не задается, создают совершенно не прозрачный цвет.

Последний конструктор позволяет задавать цвет не только в цветовой модели RGB, но и в других цветовых моделях, таких как CMYK, HSB, CIE, XYZ, определённых объектом класса `ColorSpace`:

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

Для создания распространённых цветов можно использовать константы, описанные в классе `Color`: `Color.BLACK, Color.BLUE, Color.CYAN, Color.DARK_GRAY,Color.GREEN, Color.LIGHT_GRAY, Color.MAGENTA, Color.ORANGE, Color.PINK, Color.RED, Color.WRITE, Color.YELLOW`.

Методы класса `Color` позволяют получить составляющие текущего цвета: `getRed(), getGreen(), getBlue(), getAlpha, getColorSpace.`

Существуют два метода, которые создают более яркий `brighter()` и более тёмный `darker()` цвета по сравнению с текущим цветом. Эти методы удобно использовать при выделении активного компонента более ярким цветом или, наоборот, когда необходимо показать неактивный компонент бледнее остальных компонентов.

Создав цвет, можно рисовать им в графическом контексте.

### Установка параметров отсечения

Рисование происходит только в внутри области, называемой ***областью отсечения***. Части изображения, не попавшие в эту область, не визуализируются. Размеры этой области могут быть установлены с помощью следующих методов.

`setClip(int x, int y, int width, int height)` – устанавливает область отсечения как прямоугольник со сторонами, параллельными сторонам дисплея, с левой верхней вершиной с координатами (x, y), шириной `width` и высотой `height`.

`setClip(Shape clip)` – устанавливает область отсечения, определяемую объектом, реализующим интерфейс `Shape` пакета `java.awt`, который обеспечивает определение для объектов, представляющих собой некоторую геометрическую фигуру.

`clipRect (int x, int y, int width, int height)` – определяет область отсечения как пересечение текущей области отсечения с прямоугольником, определяемым параметрами метода.

### Методы класса `Graphics` для вывода графических примитивов

После того, как получен контекст и установлены параметры вывода в нем (например, цвет выводимого компонента), можно

приступать к созданию изображения. Для этого предназначены следующие методы класса `Graphics`.

`drawLine(int x1, int y1, int x2, int y2)` – вычерчивает текущим цветом отрезок прямой между точками с координатами *(x1, y1)* и *(x2, y2)*.

`drawRect(int x, int y, int width, int height)` – выводит прямоугольник со сторонами, параллельными краям экрана, задаваемый координатами верхнего левого угла *(x, y)*, шириной `width` и высотой `height` пикселов.

`fillRect(int x, int y, int width, int height)` – выводит прямоугольник, закрашенный текущим цветом.

`draw3DRect(int x, int y, int width, int height, boolean raised)-` выводит прямоугольник, «приподнимающийся» над плоскостью рисования, если параметр `raised=true`, или «вдавленный» в плоскость если `raised=false`.

`fill3DRect(int x, int y, int width, int height, boolean raised)-` выводит закрашенный текущим цветом `draw3DRect`.

`drawOval(int x, int y, int width, int height)-` выводит эллипс, вписанный в прямоугольник, заданный параметрами метода; если `width== height`, то выводится окружность.

`fillOval(int x, int y, int width, int height)-` выводит закрашенный текущим цветом эллипс.

`drawArc(int x, int y, int width, int height, int startAngle, int arc)` – выводит часть эллипса, вписанного в прямоугольник, заданный первыми четырьмя параметрами. Дуга имеет величину `arc` градусов и отсчитывается от угла `startAngle`. Угол отсчитывается в градусах от оси *Ox*. Положительный угол отсчитывается

против часовой стрелки, отрицательный – против часовой стрелки. Чтобы вывести часть окружности, достаточно положить `width== height.`

`fillArc(int x, int y, int width, int height, int startAngle, int arc)` – выводит закрашенную текущим цветом часть эллипса. Параметры аналогичны параметрам `drawArc()`.

`drawRoundRect(int x, int y, int width, int height, int arcWidth, int acrHeight)` – выводит прямоугольник с закруглёнными углами. Закругления выводятся как четверть эллипса, вписанного в прямоугольник шириной `arcWidth` и высотой `arcHeight`, построенные в углах основного прямоугольника.

`fillRoundRect(int x, int y, int width, int height, int arcWidth, int acrHeight)` – выводит закращенный прямоугольник с закруглёнными углами. Параметры аналогичны параметрам `drawRoundRect()`.

`drawPolyline(int[] xPoints, int[] yPoints, int nPoins)` - выводит ломаную линию с вершинами в точках (`xPoints[i], yPoints[i]`) и числом вершин `nPoins.`

`clearRect(int x, int y, int width, int height)`- очищает (то есть закрашивает цветом фона) прямоугольник, определяемый параметрами метода.

`copyArea(int x, int y, int width, int height, int dx, int dy)`- копирует прямоугольную область изображения, определяемую параметрами `x, y, width, height,` в область, смещенную относительно координаты левого верхнего угла копируемой области на `dx` по оси x и на `dy` по оси y.

`drawPolygon(int[] xPoints, int[] yPoints, int nPoins)-` выводит замкнутую ломаную, проводя замыкающий отрезок прямой между первой и последней точками.

`fillPolygon(int[] xPoints, int[] yPoints, int nPoins)-` выводит закрашенную текущим цветом область, ограниченную замкнутой ломаной линий. Параметры аналогичны параметрам `drawPolygon()`.

`drawPolygon(Polygon p)` – выводит замкнутую линию, вершины которой задаются объектом класса `Polygon`.

`fillPolygon(Polygon p)-` выводит закрашенную текущим цветом область, ограниченную замкнутой ломаной линий. Параметры аналогичны параметрам `drawPolygon()`.

Класс `Polygon` предназначен для работы с произвольными многоугольниками, в том числе с треугольниками и произвольными четырёхугольниками. Объекты этого класса можно создать с помощью одного из двух конструкторов.

`Polygon()` – создаёт пустой объект.

`Polygon(int[] xPoints, int[] yPoints, int nPoins)-` задаются координаты вершин многоугольника (`xPoints[i], yPoints[i]`) и число вершин `nPoins.` Несоответствие параметров вызывает исключительную ситуацию.

После создания объекта класса `Polygon` в него можно добавить вершины, используя вызов метода `addPoint(int x, int y)`.

В классе `Polygon` определён очень полезный метод `contains()`, который позволяет проверить, лежат ли точка, отрезок прямой или прямоугольник со сторонами, параллельными сторонам экрана, передаваемый в качестве параметров этого метода, внутри полигона. Определены следующие варианты метода `contains()`.

`boolean contains(int x, int y)`,

`boolean contains(double x, double y)`,

`boolean contains(Point p)` и

`boolean contains(Point2D p)`- возвращают `true`, если передаваемая в качестве аргумента точка лежит внутри полигона, и `false` в противном случае.

Классы `Point` и `Point2D` используются для определения точек. Взаимоотношения между этими классами представлены на рис. 3.

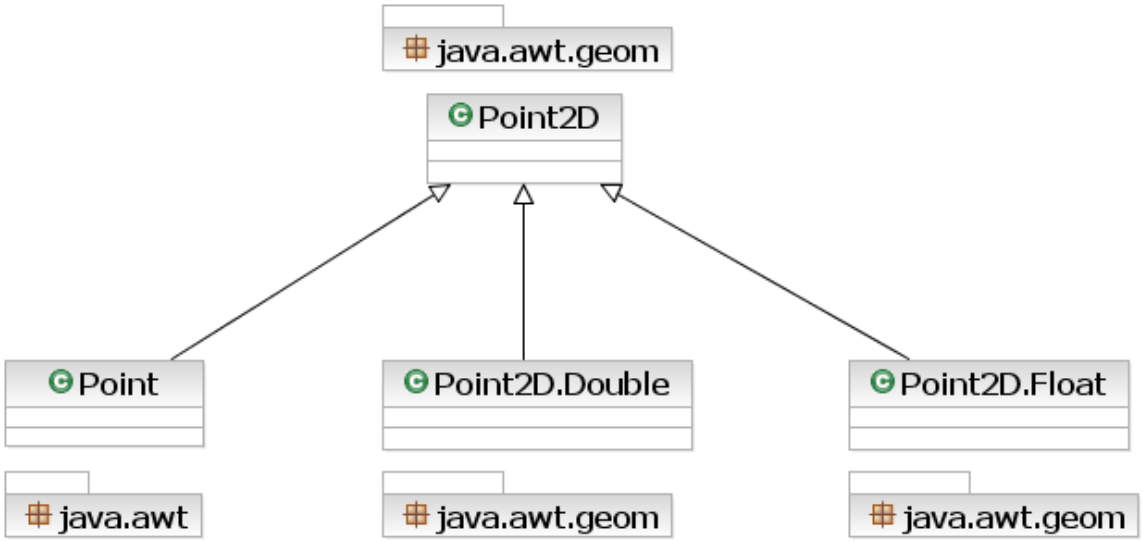



Рис. 3. Классы `Point` и `Point2D`

Классы `Point2D.Double` и `Point2D.Float` являются внутренними классами класса `Point2D`. Он (`Point2D`) же является абстрактным базовым классом для `Point2D.Double` и `Point2D.Float`. Класс `Point2D.Float` определяет точку из пары координат *(x, y)* типа `float`, класс `Point2D.Double` – точку из пары координат типа `double`. Класс `Point` из пакета `java.awt` также определяет точку, но в терминах координатной пары типа `int`. Класс `Point` является потомком `Point2D`. Три подкласса класса `Point2D` определяют используемый по умолчанию конструктор, который создаёт

точку с координатами (0, 0), и конструктор, который получает пару координат типа, соответствующего типу класса.

Методы `intersects()`класса `Polygon` позволяют проверить, пересекается ли с многоугольником отрезок прямой, заданный параметрами метода, или прямоугольник со сторонами, параллельными сторонам экрана. Определены два варианта этого метода:

`boolean intersects(double x, double y, double width, double height)` и

`boolean intersects(Rectangle2D rectangle).`

Методы `getBounds()` и `getBounds2D()` возвращают прямоугольники классов `Rectangle` и `Rectangle2D` соответственно, которые целиком содержат в себе данный многоугольник. Классы `Rectangle` и `Rectangle2D` определяют прямоугольники. Отношения между этими классами аналогичны отношениям между классами `Point` и `Point2D`.

**Установка текущего шрифта**

Для установки текущего шрифта используется метод `setFont(Font newFont)` класса `Graphics`, где `newFont` – объект класса `Font`, определяющий шрифт.

Объекты класса `Font` хранят начертания символов шрифта. Для создания объекта класса `Font` используются два конструктора.

`Font(Map attributes)` – задаёт шрифт с атрибутами `attributes`. Ключи атрибутов и некоторые их значения задаются константами класса `TextAttributes` из пакета `java.awt.font`. Этот конструктор редко используется при определении шрифта для установки текущего значения объекта класса `Graphics`, его используют при работе с `Graphics2D`, поэтому мы рассмотрим его при описании

работы с `Graphics2D`. Заметим только, что это чрезвычайно мощный конструктор `Font.`

`Font(String name, int style, int size)` – определяет шрифт по имени `name`, со стилем `style` и размером `size` типографских пунктов.

Именем шрифта `name` может быть строка с физическим именем шрифта, например “Courier New”, или одна из строк “`Dialog`”, “`DialogInput`”, “`Monospaced`”, “`Serif`”, “`SansSerif`”,“`Symbol`”. Это логические имена шрифтов. При выводе текста шрифтом, определяемым логическим именем, ему (логическому имени) сопоставляется с физическим именем шрифта или именем семейства шрифтов. Это имена реальных шрифтов, имеющихся в операционной системе пользователя. Например, логическому имени “`Serif`” может быть сопоставлено имя семейства шрифтов “`Times New Roman`”. Если `name==null`, то используется шрифт по умолчанию.

В качестве стиля шрифта `style` может использоваться одна из следующих констант: `Font.PLAIN` (обыкновенный шрифт), `Font.BOLD` (**жирный**), `Font.ITALIC` (*курсив*) или выражение `Font.BOLD+Font.Italic` (***жирный курсив***).

Размер шрифта `size`, как отмечалось выше, задаётся в типографских пунктах. Пункт равен 1/72 английского фунта, что составляет 0,351 мм. Чтобы установить шрифт размером 12 пунктов, укажите число 12 в качестве значения параметра `size`.

Подробнее со шрифтами и их параметрами можно познакомиться, например, в [3].

### Методы класса `Graphics` для вывода текста

Для вывода текста текущим цветом и шрифтом, начиная с точки *(x, y)* в классе `Graphics` существуют несколько методов.

`drawString(string s, int x, int y)`– выводит строку s.

`drawBytes(byte[] b, int offset, int length, int x, int y)` – выводит `length` элементов массива байтов `b`, начиная с индекса `offset`.

`drawChars(char[] ch, int offset, int length, int x, int y)` – выводит `length` элементов массива символов `ch`, начиная с индекса `offset`.

`drawString(AttributedCharacterIterator iter, int x, int y)` – выводит текст, занесённый в объект класса, реализующего интерфейс `AttributedCharacterIterator`, что позволяет задавать свой шрифт для каждого выводимого символа.

Точка *(x, y)* – это левая нижняя точка первой буквы текста, расположенная на базовой линии шрифта.

### Пример простейшей программы с использованием Graphics

```
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import javax.swing.JFrame;
public class GraphDemo extends JFrame {
     private static final long serialVersionUID = 1L;
// Переопределяем метод paint, который вызывается
// автоматически,когда перерисовается окно
     @Override
     public void paint(Graphics g) {
          super.paint(g);
          // Устанавливаем текущий цвет
          g.setColor(new Color(255,0,0));
          // Устанавлмваем текущий шрифт
          g.setFont(new Font("Arial",Font.BOLD,24));
          // Выводим строку
          g.drawString("ФВТ", 120, 80);
          // Изменяем текщие цвет и шрифт
          g.setColor(new Color(0,0,255));
          g.setFont(new Font("Arial",Font.ITALIC,24));
          // Выводим ещё три строки
          g.drawString("ВПМ", 20, 170);
```

```
        g.drawString("ЭВМ", 100, 170);
        g.drawString("САПР ВС", 180, 170);
        // Выводим три отрезка
        g.drawLine(120, 100, 60, 140);
        g.drawLine(150, 100, 140, 140);
        g.drawLine(180, 100, 220, 140);
        // Рисуем прямоугольники с закруглёнными краями
        g.drawRoundRect(15, 200, 270, 85, 20, 20);
        g.drawRoundRect(20, 205, 260, 75, 20, 20);
        // Изменяем текущий цвет контекста
        g.setColor(new Color(150,10,10));
        // Рисуем эллипс
        g.drawOval(25, 210, 250, 65);
        // Рисуем закрашенные окружности
        g.fillOval(40, 228, 30, 30);
        g.fillOval(75, 228, 30, 30);
    }
    public static void main(String[] args) {
        GraphDemo gd = new GraphDemo();
        // Определяем заголовок окна
        gd.setTitle("Пример графики на Java");
        // Определяем размер окна
        gd.setSize(300, 300);
        // Запрещаем пользователю изменять размеры окна
        gd.setResizable(false);
        // Определяем, что при закрытии окна заканчивается
        работа
        // программы
        gd.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Делаем окно видимым
        gd.setVisible(true);
    }
}
```

Программа создаёт изображение, показанное на рис. 4.

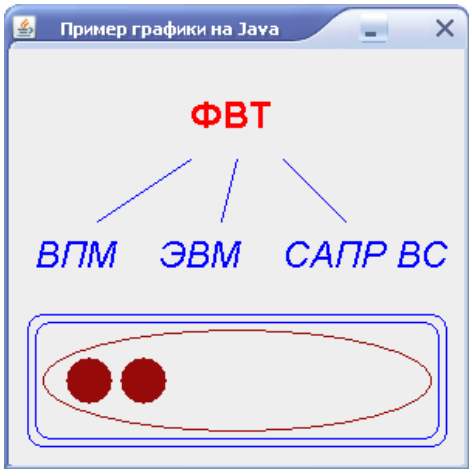



Рис. 4 Пример выполнения программы

## Порядок выполнения работы

1. Изучите теоретический материал.
2. Выполните задание в соответствии с номером варианта.
3. Выполните дополнительное задание преподавателя.
4. Ответьте на контрольные вопросы.

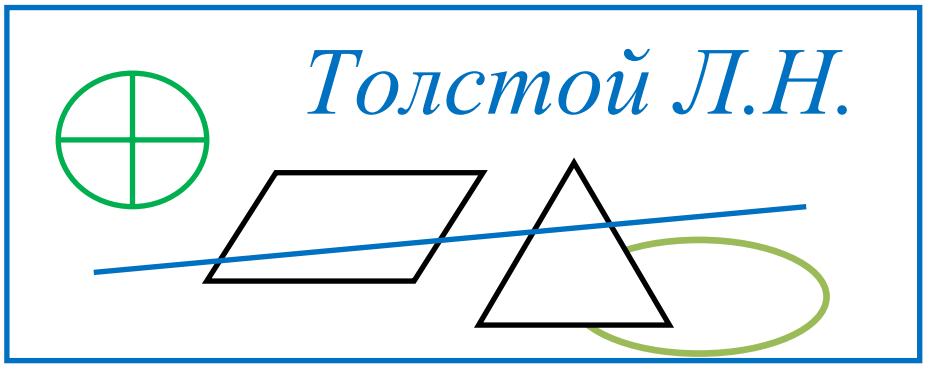
## Варианты заданий

Напишите программу, рисующую приведенное изображение. В программе обязательно должны быть использованы методы, имена которых приведены ниже рисунка.

**Вариант 1**

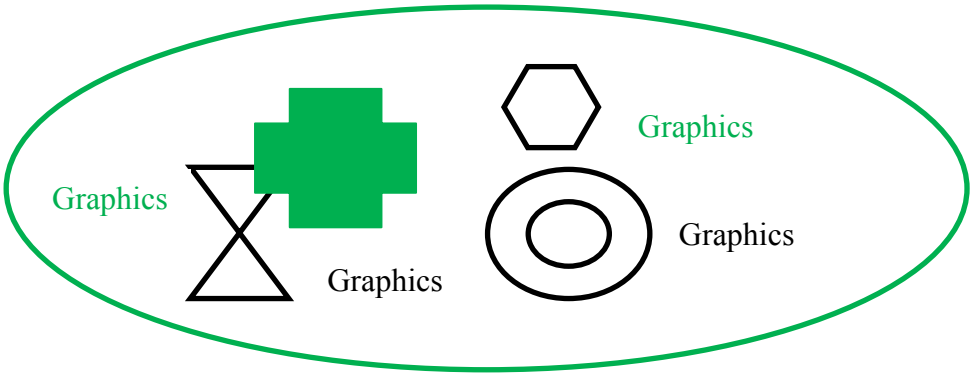

`drawPolygon(), fillOval(), drawRect().`

**Вариант 2**



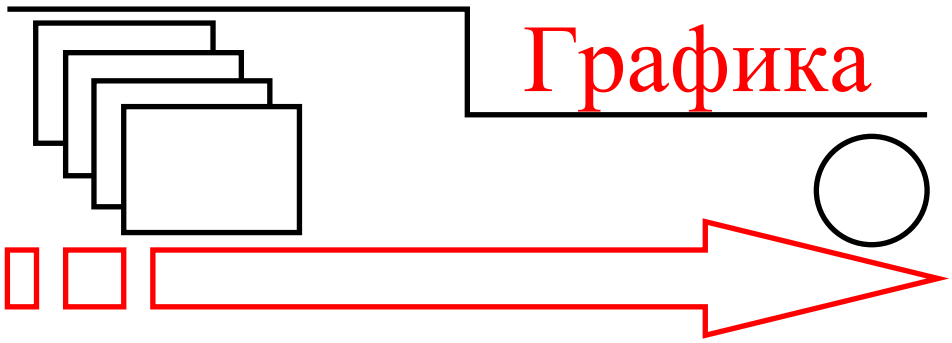

`drawPolygon(), drawOval(), drawLine().`

**Вариант 3**



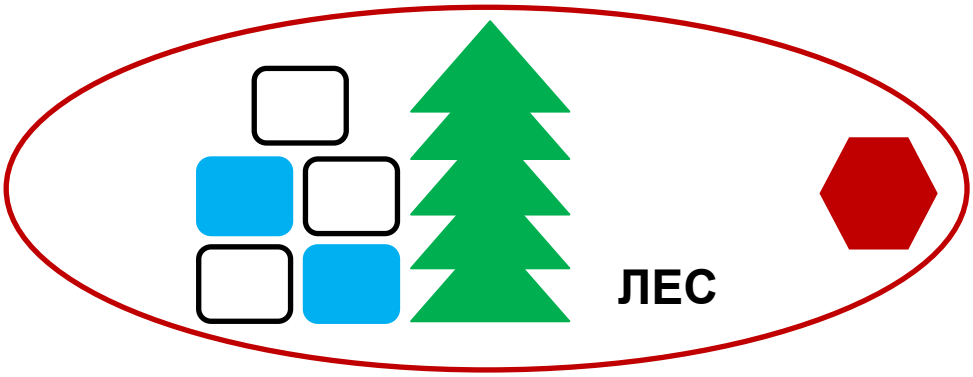

`fillPolygon(), drawPolygon(), drawOval().`

**Вариант 4**



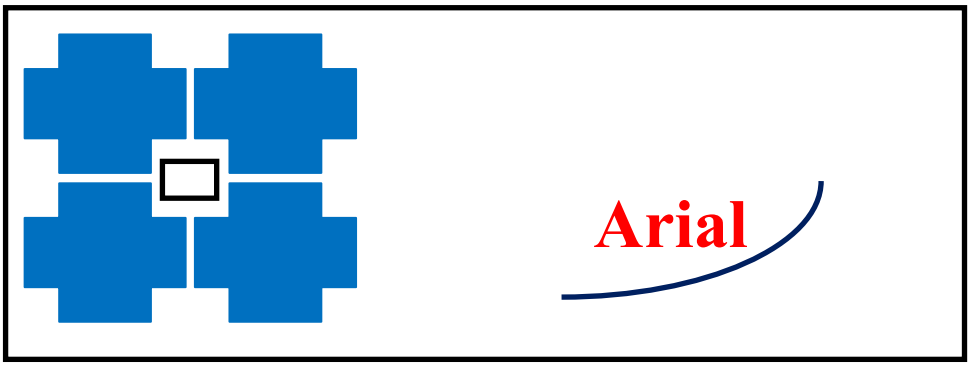

`drawPolyline(); drawRect(), drawOval().`

**Вариант 5**



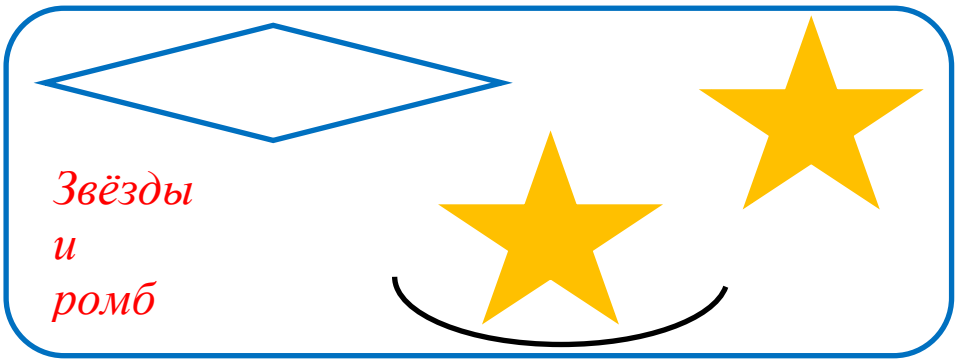

`fillRoundRect(), drawRoundRect(), fillPolygon().`

**Вариант 6**




`fillPolygon(), drawRect(),drawArc().`

**Вариант 7**

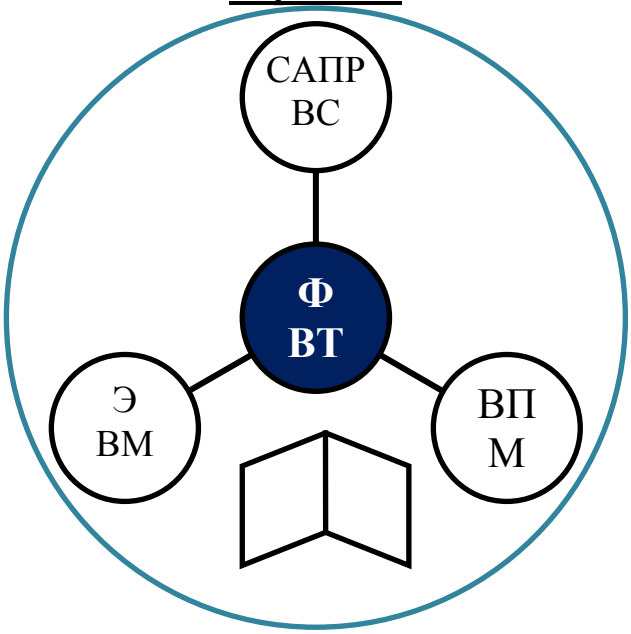



`fillPolygon(), drawpolygon(), drawArc().`

**Вариант 8**

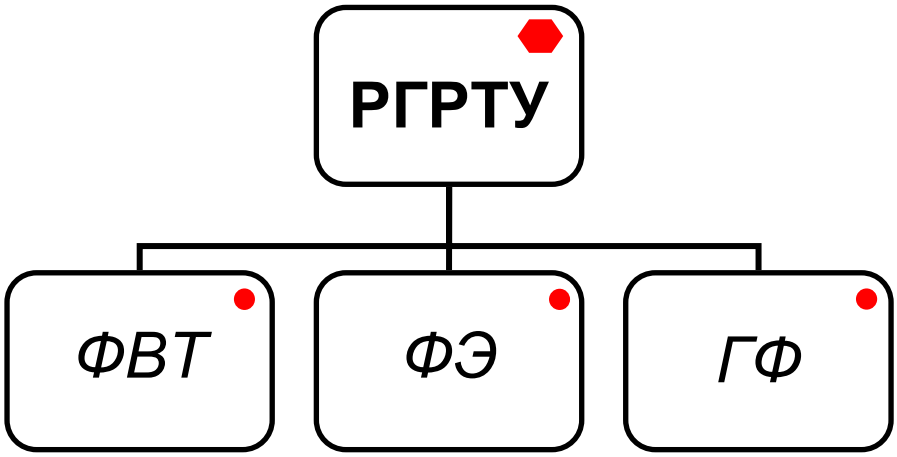

`drawOval(), drawLine(), drawArc().`

**Вариант 9**




`drawOval(), drawLine(), copyArea().`

**Вариант 10**

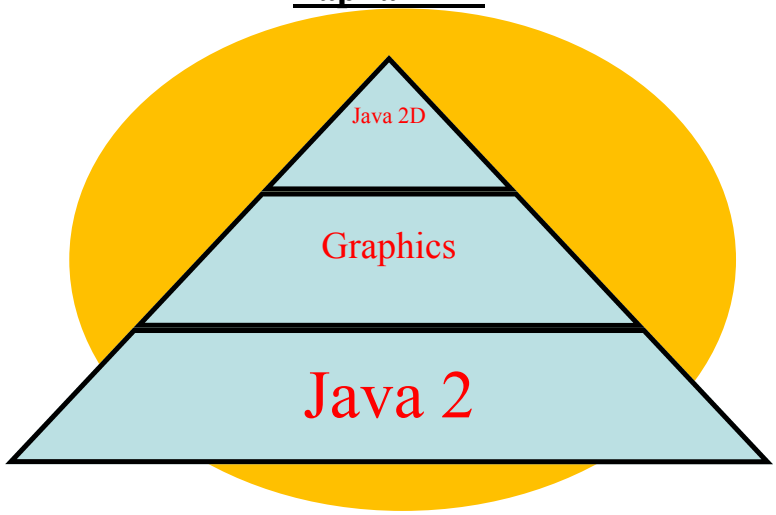



`drawRoundRect(),fillOval(),fillPolygon(),copyArea().`

**Вариант 11**


`copyArea(), fillRoundRect(), drawArc().`

**Вариант 12**




`fillOval(), drawLine(), drawString()`

**Контрольные вопросы**

1. Расскажите о предках класса JFrame.
2. Что представляет собой графический контекст?
3. Почему невозможно непосредственно создать объект класса Graphics?
4. Для чего предназначен класс Color? Опишите конструкторы этого класса.
5. Расскажите о методах класса Graphics, предназначенных для вывода текста.
6. Расскажите о методах класса Graphics, предназначенных для вывода графических примитивов.
7. Для чего предназначен класс Font? Опишите конструкторы этого класса. Расскажите о физических и логических именах шрифтов.
8. Заметили ли вы недостатки класса Graphics? Расскажите о них.

**Библиографический список**

1. Хортон А. Java 2. – М.: Лори, 2008.

2. Хабибуллин И. Самоучитель Java. – СПб.: БХВ-Петербург, 2008.

3. Юань Фень. Программирование графики для Windows. – СПб., Питер, 2002.