XXXX

ФЕДЕРАЛЬНОЕ АГЕНТСТВО ПО ОБРАЗОВАНИЮ
РЯЗАНСКИЙ ГОСУДАРСТВЕННЫЙ РАДИОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Программирование графики с использованием Java 2D

Методические указания к лабораторной работе

Рязань 2010

УДК 681.3

Программирование графики с использованием Java 2D: 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».

Табл. 1. Ил. 6. Библиогр.: 2 назв.

Программирование, графика, Java, Java 2D.

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 с использованием Java 2D

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 № 2

Для создания простых рисунков можно использовать методы класса Graphics. Однако возможностей этого класса не достаточно для построения сложных изображений. Для этих целей можно использовать Java 2D – библиотеку классов для построения сложных двумерных изображений. Для создания двумерного изображения с использованием Java 2D необходимо выполнить следующие действия [1].

1) Получить объект класса `Graphics2D`. Этот класс является подклассом (потомком) класса `Graphics`. В настоящее время методы `paint()` и `paintComponent()` автоматически получают объект класса `Graphics2D`. Поэтому для получения `Graphics2D` достаточно применить следующее приведение типа.

```
public void paint(Graphics g)
{
    Graphics2D g2 = (Graphics2D)g;
}
```

2) Использовать метод `setRenderingHints()` для добавления рекомендаций по визуализации в целях достижения компромисса между скоростью и качеством рисования (установка параметров рисования).

```
RenderingHints hints = …;
g2. setRenderingHints(hints);
```

3) Использовать метод `setStroke()` для указания *штриха* (stroke), который должен применяться для прорисовки контура рисуемой фигуры. Для штриха можно выбирать толщину, а также сплошную или пунктирную линию.

```
Stroke stroke = …;
G2.setStroke(stroke);
```

4) Использовать метод `setPaint()` для указания *заливки* (paint). Заливка используется для закрашивания областей (внутренних частей фигур). Заливка может состоять из одного сплошного цвета, нескольких меняющихся оттенков или мозаичных узоров.

```
Paint paint = …;
g2.setPaint(paint);
```

5) Использовать метод setClip() для установки *области отсечения*.

```
Shape clip = …;
g2.setClip(clip);
```

6) Использовать метод `setTransform()` для преобразования координат рисунка из относительной пользовательской системы координат в абсолютную систему координат устройства. Это преобразование следует применять в тех случаях, когда проще создать рисунок в пользовательской системе координат, чем использовать координаты, выраженными в пикселах.

```
AffineTransform = transform = …;
g2.setTransform(transform);
```

7) Использовать метод `setComposite()` для задания *правил композиции* (*compositing rule*), описывающих, каким образом новые пиксели должны объединяться с уже существующими.

```
Composite composite = …;
g2.setComposite (composite);
```

8) Создать фигуру. В Java 2D предусмотрено много объектов-фигур и методов для их комбинирования.

```
Shape shape = …;
```

9) Нарисовать или залить фигуру. Под рисованием понимается очерчивание контуров фигуры, а под заливкой – закрашивание её внутренней части.

```
g2.draw(shape);
```

```
g2.fill(shape);
```

Естественно, что все эти действия во всех случаях выполнять не нужно, поскольку для многих параметров 2D-контекста предусмотрены значения по умолчанию, изменять которые не требуется.

Процесс рендеринга двумерной сцены с помощью класса `Graphics2D` показан на рис. 1.

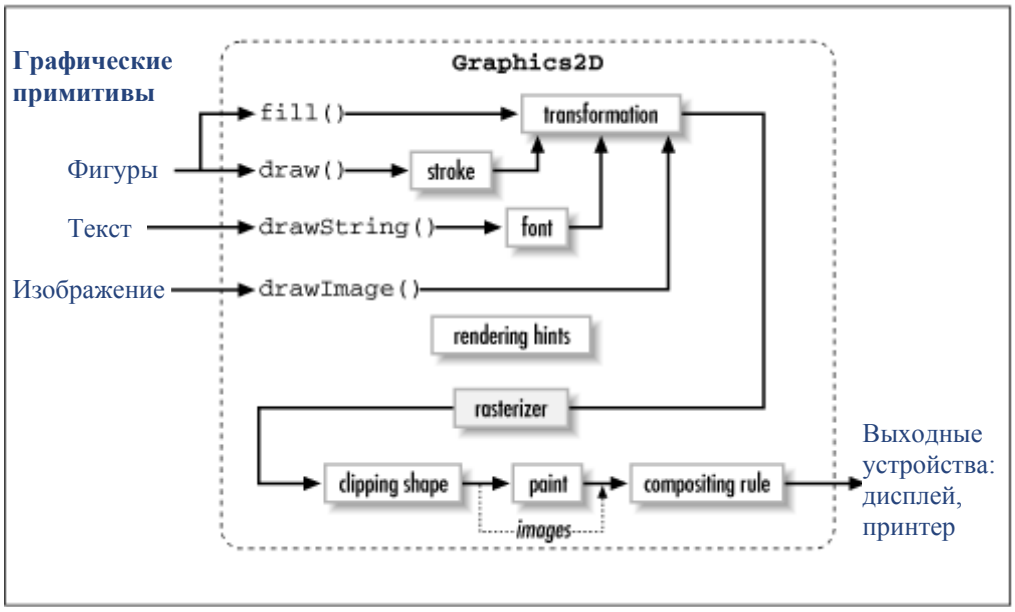


Рис.1. Рендеринг с использованием `Graphics2D`

Установка параметров рисования (rendering hints)

Для определения параметров рисования в Java 2D используется метод `setRenderHint()` класса `Graphics2D`. Он позволяет задавать лишь одно значение параметра. Ключи и значения параметров описаны в классе `RenderingHints`. В табл. 1 перечислены наиболее распространённые возможные варианты. Значения, оканчивающиеся на _DEFAULT, используются по умолчанию.

Табл.1 Ключи и значения параметров рисования

Ключ	Значение	Описание
KEY_ANTIALIASING	VALUE_ANTIALIAS_ON VALUE_ANTIALIAS_OFF VALUE_ANTIALIAS_DEFAULT	Включение или отключение сглаживания фигур

Табл.1 Ключи и значения параметров рисования (продолжение)

Ключ	Значение	Описание
KEY_TEXT_ANTIALIASING	VALUE_TEXT_ANTIALIAS_ON VALUE_TEXT_ANTIALIAS_OFF VALUE_TEXT_ANTIALIAS_DEFAULT VALUE_TEXT_ANTIALIAS_GASP VALUE_TEXT_ANTIALIAS_LCD_HRGB VALUE_TEXT_ANTIALIAS_LCD_HBGR VALUE_TEXT_ANTIALIAS_LCD_VRGB VALUE_TEXT_ANTIALIAS_LCD_VBGR	Включение и отключение сглаживания шрифтов. Значение VALUE_TEXT_ANTIALIAS_GASP подразумевает просмотр gasp-таблицы шрифта для принятия решения о том, следует ли выполнять его сглаживание в случае такого размера, а значения со словом LCD-визуализацию подпикселей для мониторов соответствующего типа
KEY_FRACTIONAL_METRICS	VALUE_FRACTALMETRICS_ON VALUE_FRACTALMETRICS_OFF VALUE_FRACTALMETRICS_DEFAULT	Включение и выключение дробных размеров шрифтов. Использование дробных размеров шрифтов улучшить их расположение
KEY_RENDERING	VALUE_RENDER_QUALITY VALUE_RENDER_SPEED VALUE_RENDER_DEFAULT	Выбор алгоритма рендеринга для достижения лучшего качества или большей скорости
KEY_STROKE_CONTROL	VALUE_STROKE_NORMALIZE VALUE_STROKE_PURE VALUE_STROKE_DEFAULT	Выбор того, должно ли размещение пера контролироваться графическим ускорителем, или вычисляться строго по правилу, которое требует, чтобы перья обязательно проходили по центру пикселей
KEY_DITHERING	VALUE_DITHER_ENABLE VALUE_DITHER_DISABLE VALUE_DITHER_DEFAULT	Включение и выключение имитации полутонов. Алгоритм имитации полутонов основан на специальном распределении рядом расположенных пикселей. Сглаживание мешать имитации полутонов.
KEY_ALPHA_INTERPOLATION	VALUE_ALPHA_INTERPOLATION_QUALITY VALUE_ALPHA_INTERPOLATION_SPEED VALUE_ALPHA_INTERPOLATION_DEFAULT	Включение и отключение точного вычисления композиции на основе значений альфа-канала

Табл.1 Ключи и значения параметров рисования (окончание)

Ключ	Значение	Описание
KEY_COLOR_RENDERING	VALUE_COLOR_RENDER_QUALITY VALUE_COLOR_RENDER_SPEED VALUE_COLOR_RENDER_DEFAULT	Выбор качества или скорости отображения цветов. Такой выбор необходимо делать только в том случае, когда используются различные цветовые пространства.
KEY_ INTERPOLATION	VALUE_INTERPOLATION_NEAREST_NEIGHBOR VALUE_INTERPOLATION_BILINEAR VALUE_INTERPOLATION_BICUBIC	Выбор правила интерполяции для масштабирования рисунка.

Например, для включения сглаживания фигур, нужно использовать следующий метод:

```
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
RenderingHints.VALUE_ANTIALIAS_ON).
```

Установка пера (stroke)

Как видно из рис. 1, метод `draw()` класса `Graphics2D` для рисования фигур использует перо, которое содержит определение того, каким образом должна выводиться граница фигуры. По умолчанию используется сплошная линия толщиной один пиксель.

Для изменения типа пера используется метод setStroke(), которому в качестве параметра передаётся экземпляр класса, реализующего интерфейс `Stroke`. В Java 2D определён только один такой класс – `BasicStroke()`.

Для штриха можно задать линии произвольной толщины. Например, для определения линии толщиной 12 пикселей можно использовать следующий код:

```
g2.setStroke(new BasicStroke(12.0F));
```

Если толщина линии превышает один пиксель, то для формирования концов линии можно использовать один из следующих стилей.

- ***Стык*** (butt cap) – линия обрывается в конечной точке.
- ***Овальное перекрытие*** (round cap) – в конце линии добавляется полукруг.
- ***Прямоугольное перекрытие*** (square cap) – в конце линии добавляется прямоугольник.

Стили концов линий показаны на рис. 2.

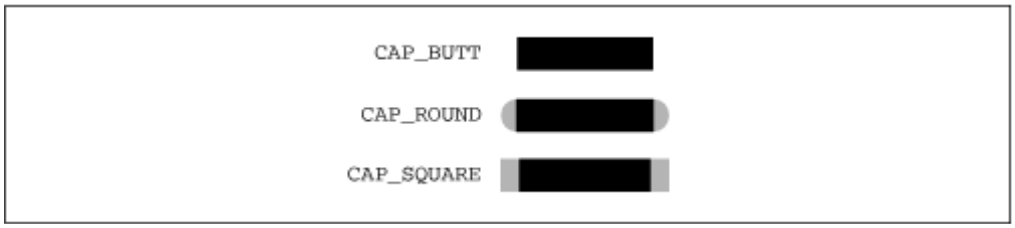


Рис. 2. Стили концов линий

Для соединения двух линий толщиной более одного пикселя можно указать один из следующих стилей.

- ***Косое соединение*** (bevel join) – соединяет линии по прямой, перпендикулярной биссектрисе угла между ними.
- ***Скруглённое соединение*** (round join) – соединяет линии с помощью овального перекрытия.
- ***Фасетное соединение*** (miter join) – продолжает линии с образованием клина. Фасетное соединение плохо подходит для линий, которые пересекаются под небольшими углами. Поэтому если две линии пересекаются под углом, меньшим *предельного фасетного угла* (miter limit), то вместо фасетного образуется косое соединение. По умолчанию предельный фасетный угол равен 10°.

Стили соединений приведены на рисунке 3.

Рис. 3. Стили соединений

Для пунктирных перьев можно выбрать *шаблон пунктиров* (dash pattern). Шаблон пунктира задаётся в виде массива float[], который содержит длины рисуемых и не рисуемых фрагментов штриха. Стили окончания линий применяются для всех элементов пунктира. При создании BasicStroke помимо шаблона пунктира указывается также фаза пунктира (dash phase), то есть место шаблона в данной линии, с которого начинается вывод (обычно фаза имеет значение 0).

```
float[] dashPattern = {12.0F,12.0F};
g2.setStroke(new BasicStroke(12.0F, BasicStroke.CAP_BUTT,
                BasicStroke.JOIN_MITER,
                10.0F,                  // Предельный фасетный угол
                dashPattern,            // Шаблон пунктира
                0));                    // Фаза пунктира
```

Шаблон пунктира и влияние фазы пунктира на его отображение, показаны на рис. 4.

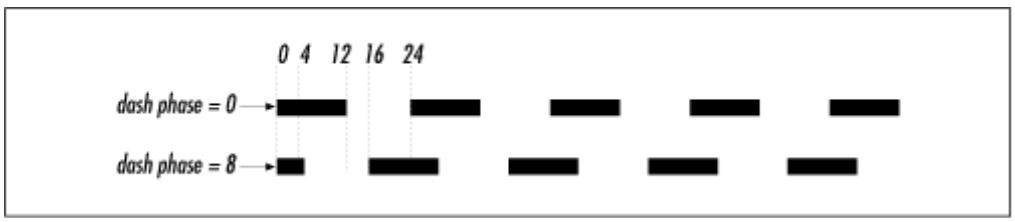


Рис. 4. Шаблон пунктира и фаза

Заливка

Внутренняя часть фигуры заполняется в соответствии с *заливкой* (paint). Для указания стиля заливки используется объект, класс которого реализует интерфейс `Paint`. В Java 2D API предусмотрено три таких класса.

- Класс `Color` предназначен для заполнения фигуры сплошным цветом. Нужно лишь вызвать метод `setPaint()` и указать объект `Color`: `g2.setPaint(new Color(255,0,0));`
- Класс `GradientPaint` предназначен для заполнения фигуры градиентной заливкой, то есть постепенно меняющимся цветом. Для создания объекта нужно указать две точки и цвета для них: `g2.setPaint(new GradientPaint(p1, Color.RED, p2, Color.YELLOW))`. Изменение цвета осуществляется вдоль прямой, соединяющей эти точки. Цвета остаются постоянными вдоль прямых, перпендикулярных к линии соединения. Для точек, которые находятся за концами соединительной линии, задаются цвета соответствующих конечных точек. Если вызвать конструктор объекта `GradientPaint` с параметром `cyclic`, имеющим значение true, то изменение цвета будет циклически происходить и за пределами конечных точек:

```
g2.setPaint(new GradientPaint(p1, Color.RED, p2,
                              Color.YELLOW, true)).
```

- Класс `TexturePaint()` предназначен для заполнения фигуры мозаичной текстурой, то есть несколькими копиями одного рисунка. Для создания объекта-текстуры `TexturePaint` нужно указать копируемый объект – рисунок `BufferImage` и *прямоугольник привязки* (anchor rectangle), например:

```
BufferImage bufImage=ImageIO.read(new File("abc.gif"));
Rectangle2D anchorRect =new(Rectangle2D.Double(2,2,10,10));
g2.setPaint(new TexturePaint(bufferedImage, anchorRect));
```

Заданный объект-рисунок масштабируется таким образом, чтобы соответствовать размерам прямоугольника привязки. Прямоугольник привязки повторяется необходимое количество раз в направлениях, параллельных осям *x* и *y* с образованием мозаики.

Классы геометрических примитивов (фигуры)

В Java 2D используется принципиально иной подход к изображению фигур по сравнению с подходом, используемым при работе с классом `Graphics`. Этот подход является полностью объектно-ориентированным и заключается в использовании фигур следующих классов: `Line2D`, `Rectangle2D`, `RoundRectangle2D`, `Ellipse2D`, `Arc2D`, `QuadCurve2D`, `CubicCurve2D`, `GeneralPath`. Все классы фигур реализуют интерфейс `Shape`. Отношения между фигурами показаны на рис. 5.

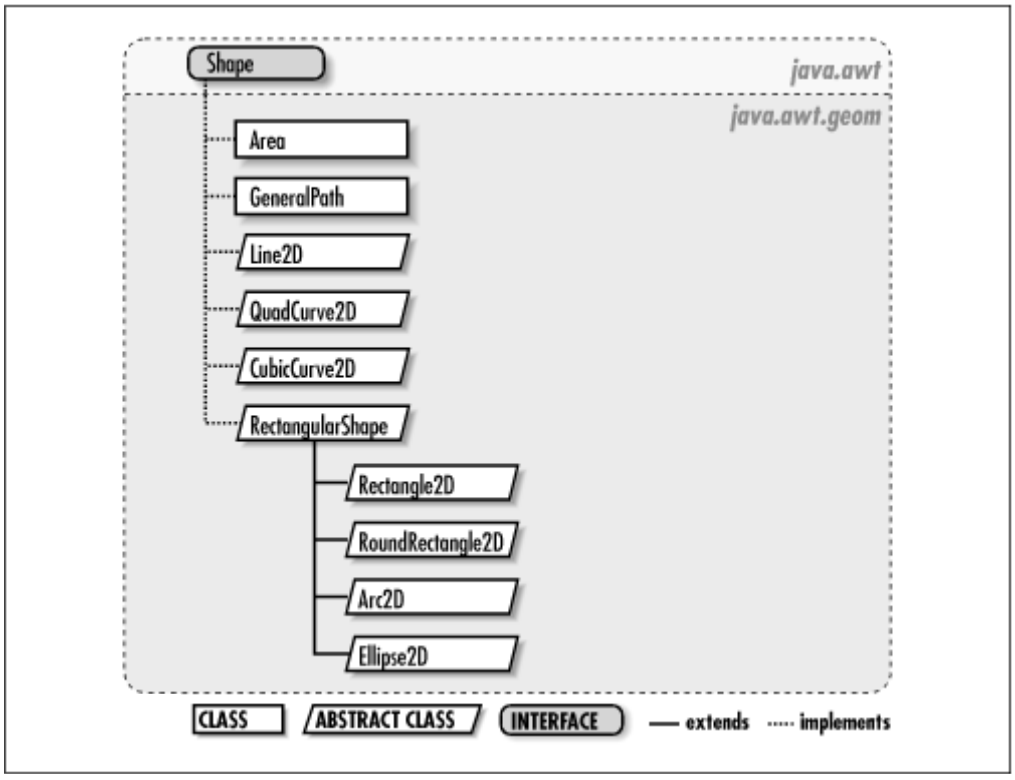


Рис. 5. Отношения между классами фигур Java 2D

Каждый абстрактный класс, название которого заканчивается на 2D, имеет два не абстрактных подкласса, отличающиеся типом задания координат (они задаются в виде чисел типа `float` или `double`): например, `Rectangle2D.Float` и `Rectangle2D.Double`.

В Java 2D имеется также класс точки `Point2D`, который описывает точку с координатами (x, y). Точки используются для определения фигур, но сами фигурами не являются.

Для того чтобы создать фигуру, нужно создать экземпляр соответствующего класса, а затем передать его в качестве аргумента метода `draw()` для класса `Grahpics2D`.

Классы `Line2D`, `Rectangle2D`, `RoundRectangle2D`, `Ellipse2D`, `Arc2D` соответствуют методам `drawLine()`, `drawRectangle()`, `drawRoundRect()`, `drawOval()` и `drawArc()` класса Graphics.

Для того чтобы описать линию, нужно задать начальную и конечную точки, заданные как объекты класса `Point2D`, либо в виде пары чисел:

```
Line2D line = new Line2D.Double(
    new Point2D.Double(5.0,5.0),
    new Point2D.Double(50.0,50.0)
);
Line2D line = new Line2D.Double(5.0,5.0,50.0,50.0);
```

Классы `Rectangle2D`, `RoundRectangle2D`, `Ellipse2D`, `Arc2D` являются абстрактными потомками абстрактного класса `RectangularShape` (фигуры, вписываемые в прямоугольник). Из этого следует, что для создания этих фигур необходимо определять прямоугольник, в который вписывается соответствующая фигура. Например:

```
Rectanle2D rect = new Rectangle2D.Double(1,1,50,75);
Ellipse2D ellipse = new Ellipse2D.Double(1,1,50,75);
Arc2D arc = new Arc2D.Double(1,1,50,75,0,90,
                                        closureType);
```

Во всех этих примерах два первые аргумента описывают координаты левого верхнего угла прямоугольника, в который вписана фигура, третий и четвёртый аргументы – соответственно длина и ширина этого прямоугольника.

Пятый и шестой аргументы в конструкторе класса `Arc2D.Double` определяют начальный и конечный углы (в градусах) дуги (дуга является частью эллипса, вписанного в прямоугольник, определяемый четырьмя первыми параметрами конструктора). Последний параметр конструкторов `Arc2D.Double` и `Arc2D.Double` – `closureType` (тип замыкания) определяет должна ли замыкаться определяемая дуга, а если должна, то каким образом. Значением `closureType` может являться одна из констант, определённых в классе `Arc2D`: `Arc2D.OPEN` (просто дуга), `Arc2D.PIE` (дуга, края которой соединены с центром), `Arc2D.CHORD` (края дуги соединены хордой). На рис. 6 показаны три дуги, являющиеся частями одного и того же эллипса, но имеющие разное значение параметра `closureType`.

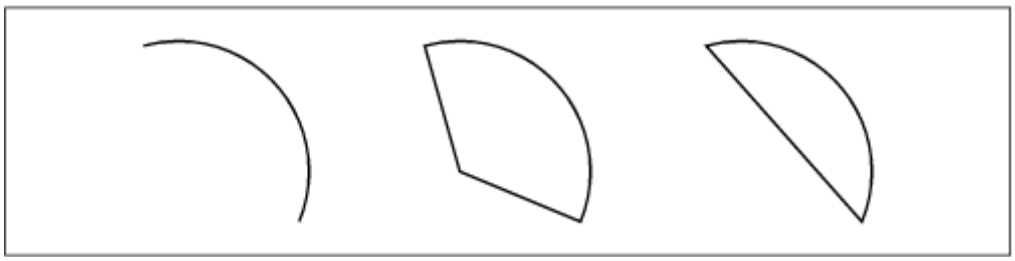


Рис. 6. Влияние значения параметра `closureType` на дугу

В Java 2D существует возможность построения кривых второго и третьего порядков. Кривые второго и третьего порядков определяются

двумя *конечными точками* и одной (для кривой второго порядка) или двумя (для кривой третьего порядка) *управляющими точками*. На рис. 7 показаны кривые третьего (вверху) и второго (внизу) порядков. На рисунке хорошо видны конечные и управляющие точки.

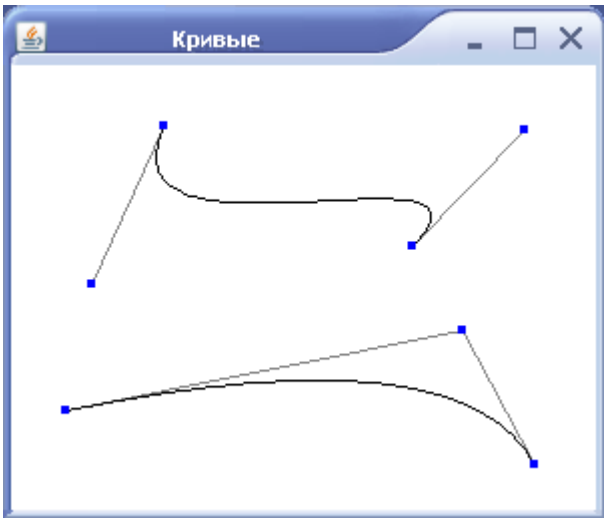


Рис. 7. Кривые второго и третьего порядков

Для создания квадратичных и кубических кривых задаются координаты конечных и контрольных точек:

```
QuadCurve2D q = new QuadCurve2D.Double(startX, startY, controlX, controlY, endX, endY);
CubicCurve2D c = CubicCurve2D.Double(startX, startY, control1X, control1Y, control2X, control2Y,endX, endY).
```

Для создания и сохранения произвольной кривой в виде последовательности линий, кривых второго и третьего порядка предусмотрен класс `GeneralPath`. Для создания *пути* (path) сначала можно создать пустой путь и с помощью метода `moveTo()` передать координату первой точки:

```
GeneralPath path = new GeneralPath();
path.moveTo(10,10);
```

Затем можно продолжить построение пути с помощью методов `lineTo()` (линия до точки), `quadTo()` (кривая второго порядка до точки), `curveTo()` (кривая третьего порядка до точки). Эти методы продолжают путь прямой линией, кривыми второго и третьего порядка соответственно. При вызове метода `lineTo()` в качестве его параметра надо передать конечную точку (начальная точка прямой уже задана в пути), а для двух других методов нужно сначала указать контрольную точку (или две контрольных точки), а затем конечную точку. Например:

```
// Добавляем отрезок прямой (10,10)- (50,50) к пути
path.lineTo(50,50);
// Отрезок добавлен к пути,
// теперь текущая конечная точка пути – (50,50)
path.curveTo(control1X, control1Y, control2X,
          control2Y,endX, endY);
// Кривая третьего порядка добавлена к пути,
// теперь текущая конечная точка пути – (endX,endY).
```

Путь не обязательно должен быть связанным, то есть метод `moveTo()` можно вызвать в любое время для создания нового сегмента пути. К пути можно добавить произвольный объект типа `Shape`, используя для этого метод `append()` и передав в качестве параметра этого метода объект, класс которого реализует интерфейс `Shape`. При этом контур фигуры добавляется в конец пути. Второй параметр метода `append()` имеет значение `true`, если новую фигуру необходимо связать с последней точкой траектории, и `false` в противном случае. Например:

```
Rectangle2D r =…;
path.append(r, false);
```

добавляет к пути контур прямоугольника, не соединяя его с концом маршрута, а вызов метода `path.append(r, true)` добавляет прямую

линию, которая соединяет конечную точку пути и начальную точку прямоугольника `r`, и затем добавляет контур прямоугольника к пути.

После того, как путь построен, вызывается метод `path.closePath()` (закрыть путь). Чтобы нарисовать созданный путь `path` достаточно воспользоваться методом `g2.draw(path)`.

Порядок выполнения работы

1. Изучите теоретический материал.

2. Выполните задание преподавателя, связанное с разработкой программы, строящей изображение с использованием Java 2D.

3. Ответьте на контрольные вопросы.

Контрольные вопросы

1. Опишите процесс создания изображения в Java 2D, воспользовавшись рисунком 1.

2. Опишите процесс установки параметров рисования.

3. Расскажите о параметрах пера и процессе их установки.

4. Расскажите об установке параметров заливки.

5. Проанализируйте рисунок 5.

6. Расскажите о классах `Rectangle2D` и `RoundRectangle2D`.

7. Расскажите о классах `QuadCurve2D` и `CubicCurve2D`.

8. Расскажите о классе `GeneralPath`.

Библиографический список

1. Хорстман К., Корнелл Г. Java 2. Библиотека профессионала, том. Тонкости программирования. – М.: ООО «И.Д. Вильямс», 2009.

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

Приложение 1

Пример программы Java 2D и результат её выполнения

```
// GraphDemo2.java
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.geom.Arc2D;
import java.awt.geom.CubicCurve2D;
import java.awt.geom.GeneralPath;
import javax.swing.JFrame;
public class GraphDemo2 extends JFrame {
// Переопределяем метод paint, который вызывается
// автоматически,когда необходимо перерисовать окно
public void paint(Graphics g) {
super.paint(g);
// Получаем объект класса Graphics2D
Graphics2D g2 = (Graphics2D) g;
// Устанавливаем параметры пера
g2.setStroke(new BasicStroke(12F, BasicStroke.CAP_ROUND,
                              BasicStroke.JOIN_BEVEL ));
// Рисуем две линии
g2.drawLine(50, 50, 100, 50);
g2.drawLine(50, 50, 100, 100);
// Устанавливаем сглаживание
 g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                     RenderingHints.VALUE_ANTIALIAS_ON);
// Меняем перо
g2.setStroke(new BasicStroke(3, BasicStroke.CAP_ROUND,
                             BasicStroke.JOIN_BEVEL ));
// Создаём и выводим дугу. Обратите внимание на Arc2D.PIE
```

```
     Arc2D arc_1=new
     Arc2D.Double(100,120,150,150,0,90,Arc2D.PIE);
     g2.draw(arc_1);
// Сбрасываем режим сглаживания, чтобы увидеть, что это даёт
     g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                         RenderingHints.VALUE_ANTIALIAS_OFF);
     Arc2D arc_2=new
     Arc2D.Double(120,100,150,150,0,90,Arc2D.CHORD);
     g2.draw(arc_2);
// Обязательно сравните качество изображения
// со сглаживанием и без него!!!
// Создаём и выводим кривую третьего порядка, предварительно
// включив сглаживание и изменив цвет и перо.
     g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                         RenderingHints.VALUE_ANTIALIAS_ON);
     g2.setColor(new Color(200,25,100));
     g2.setStroke(new BasicStroke(5, BasicStroke.CAP_ROUND,
                                  BasicStroke.JOIN_BEVEL ));
     CubicCurve2D c = new CubicCurve2D.Double(10,250,100,
                                       190,150,200, 250,250);
     g2.draw(c);
// Создаём и выводим путь
     GeneralPath gp = new GeneralPath();
     gp.moveTo(100, 240);
     gp.lineTo(200, 240);
     gp.lineTo(200, 290);
     gp.lineTo(100, 290);
     gp.lineTo(100, 240);
     gp.closePath();
     g2.draw(gp);
// Изменяем текущий цвет
     g2.setColor(new Color(0,50,100));
// Заливаем сплошным цветом выведенный путь
     g2.fill(gp);
```

```
    }
    public static void main(String[] args) {
    GraphDemo2 gd = new GraphDemo2();
// Определяем заголовок окна
    gd.setTitle("Java 2D");
// 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);
        }
    }
```

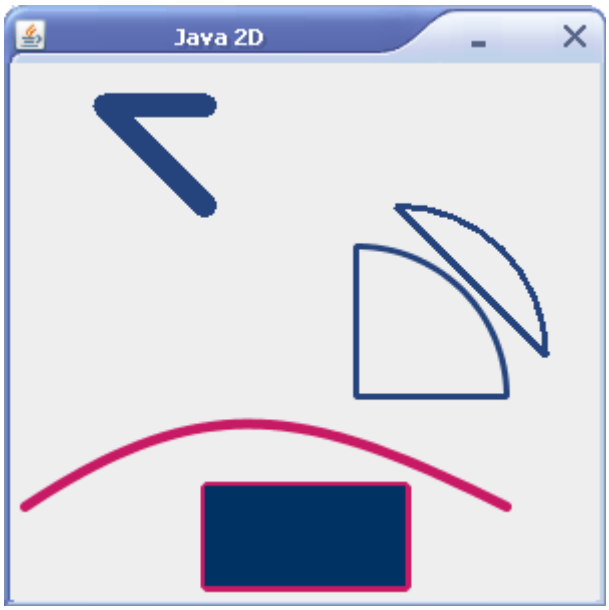


Рис. 7. Результат работы программы

Приложение 2
Текст программа, создающей изображение, показанное на рис. 7

```
// DragKing.java
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;

public class DragKing extends ApplicationFrame implements
                        MouseListener, MouseMotionListener{
public static void main(String[] args) {
     new DragKing();
}
protected Point2D[] mPoints;
protected Point2D mSelectedPoint;
public DragKing() {
super("Кривые");
setSize(300, 300);
mPoints = new Point2D[9];
// Кривая третьего порядка.
mPoints[0] = new Point2D.Double(50, 75);
mPoints[1] = new Point2D.Double(100, 100);
mPoints[2] = new Point2D.Double(200, 50);
mPoints[3] = new Point2D.Double(250, 75);
// Кривая второго порядка.
mPoints[4] = new Point2D.Double(50, 175);
mPoints[5] = new Point2D.Double(150, 150);
mPoints[6] = new Point2D.Double(250, 175);
// Линия.
mPoints[7] = new Point2D.Double(50, 275);
mPoints[8] = new Point2D.Double(250, 275);
mSelectedPoint = null;
// Прослушивание событий мыши.
addMouseListener(this);
addMouseMotionListener(this);
```

```
setVisible(true);
}
public void paint(Graphics g) {
Graphics2D g2 = (Graphics2D)g;
// Рисуем направляющие.
Line2D tangent1 = new Line2D.Double(mPoints[0], mPoints[1]);
Line2D tangent2 = new Line2D.Double(mPoints[2], mPoints[3]);
g2.setPaint(Color.gray);
g2.draw(tangent1);
g2.draw(tangent2);
// Рисуем кривую третьего порядка.
CubicCurve2D c = new CubicCurve2D.Float();
c.setCurve(mPoints, 0);
g2.setPaint(Color.black);
g2.draw(c);
// Рисуем направляющие.
tangent1 = new Line2D.Double(mPoints[4], mPoints[5]);
tangent2 = new Line2D.Double(mPoints[5], mPoints[6]);
g2.setPaint(Color.gray);
g2.draw(tangent1);
g2.draw(tangent2);
// Рисуем кривую второго порядка.
QuadCurve2D q = new QuadCurve2D.Float();
q.setCurve(mPoints, 4);
g2.setPaint(Color.black);
g2.draw(q);
// Рисуем линию.
Line2D l = new Line2D.Float();
l.setLine(mPoints[7], mPoints[8]);
g2.setPaint(Color.black);
g2.draw(l);
for (int i = 0; i < mPoints.length; i++) {
// Если точка выбрана, то используем цвет для выбранной
точки.
```

```
if (mPoints[i] == mSelectedPoint)
g2.setPaint(Color.red);
else
g2.setPaint(Color.blue);
// Рисуем точку.
g2.fill(getControlPoint(mPoints[i]));
}
}
protected Shape getControlPoint(Point2D p) {
// Создаём маленький квадрат вокруг точки.
int side = 4;
return new Rectangle2D.Double(p.getX() - side / 2, p.getY() -
                                        side / 2,side, side);
}
public void mouseClicked(MouseEvent me) {
     }
public void mousePressed(MouseEvent me) {
mSelectedPoint = null;
for (int i = 0; i < mPoints.length; i++) {
     Shape s = getControlPoint(mPoints[i]);
          if (s.contains(me.getPoint())) {
               mSelectedPoint = mPoints[i];
               break;
          }
}
repaint();
}
public void mouseReleased(MouseEvent me) {}
public void mouseMoved(MouseEvent me) {}
public void mouseDragged(MouseEvent me) {
if (mSelectedPoint != null) {
mSelectedPoint.setLocation(me.getPoint());
repaint();
}
```

```
}
public void mouseEntered(MouseEvent me) {}
public void mouseExited(MouseEvent me) {}
}
```

Приложение 3

Текст программа, демонстрирующей градиентную заливку и пример её работы

```
import java.awt.Color;
import java.awt.GradientPaint;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.RenderingHints;
import java.awt.geom.Ellipse2D;
import javax.swing.JFrame;

public class Zalivki extends JFrame {
private static final long serialVersionUID = 1L;
public void paint(Graphics g) {
super.paint(g);
Graphics2D g2 = (Graphics2D)g;
Ellipse2D ellipse = new Ellipse2D.Double(100,100,100,100);
g2.setPaint(new GradientPaint(100,150,
          new Color(255,0,0),200,150, new Color(0,0,200)));
g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                        RenderingHints.VALUE_ANTIALIAS_ON);
g2.fill(ellipse);
}
public static void main(String[] args) {
     Zalivki gd = new Zalivki();
     gd.setTitle("Градиентная заливка");
     gd.setSize(300, 300);
     gd.setResizable(false);
```

```
        gd.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        gd.setVisible(true);
        }
}
```

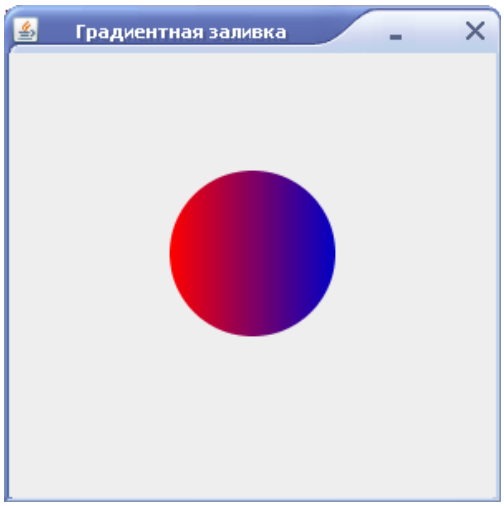


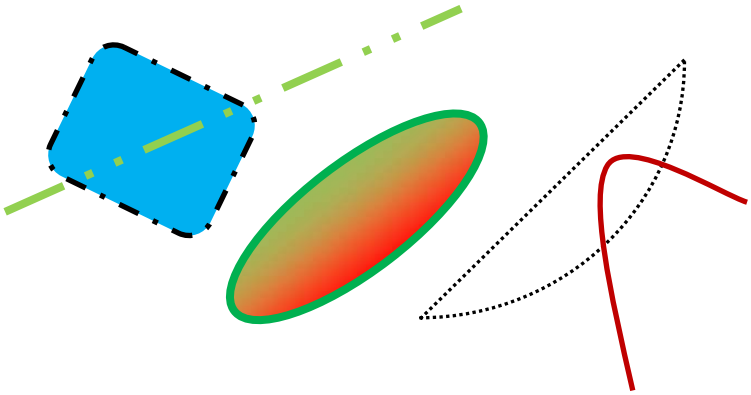
Рис. 8 Результаты работы программы

Приложение 4

Варианты заданий

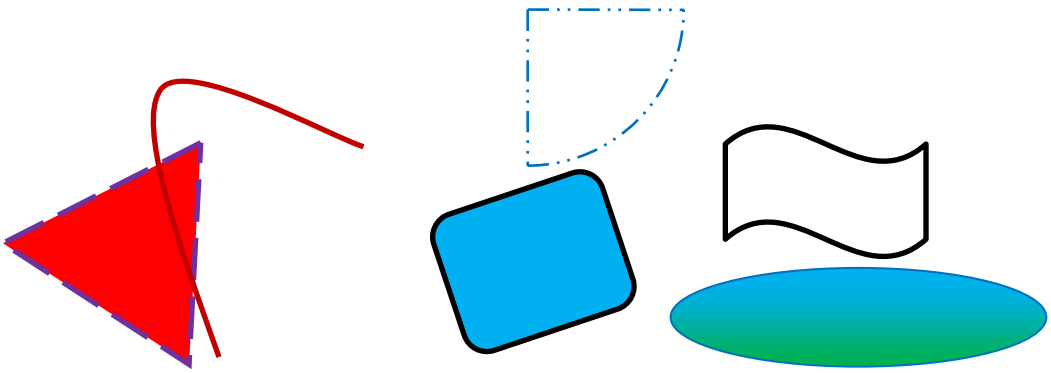
Напишите программу, строящую приведённое изображение. При написании программы обязательно используйте приведённые под изображением классы.

Вариант 1



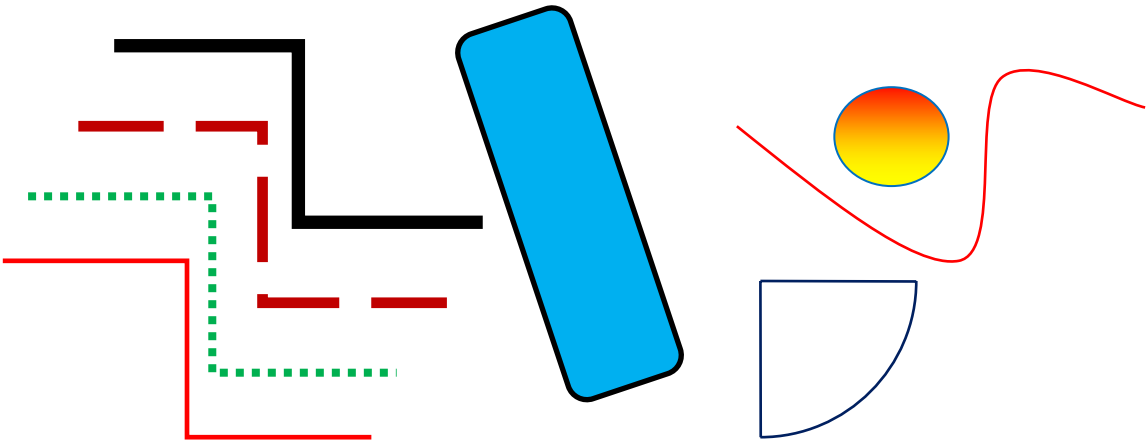
`CubicCurve2D, Arc2D, Ellipse2D, RoundRectangle2D`

Вариант 2



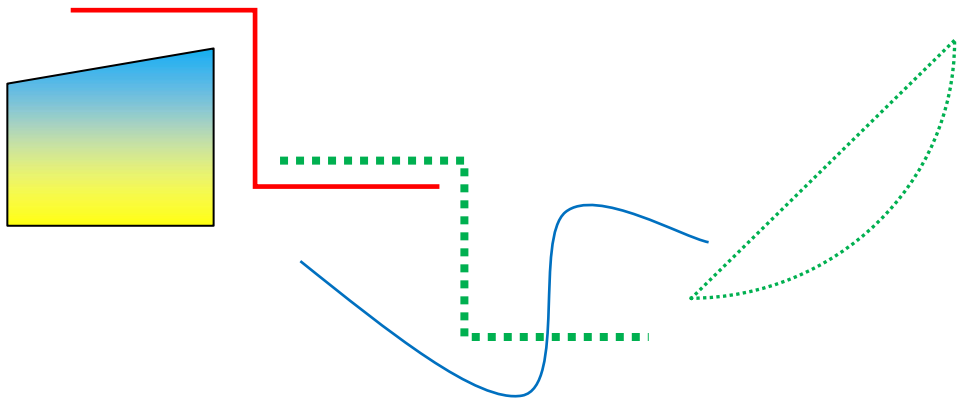
`CubicCurve2D, Arc2D, Ellipse2D, RoundRectangle2D, GeneralPath`

Вариант 3



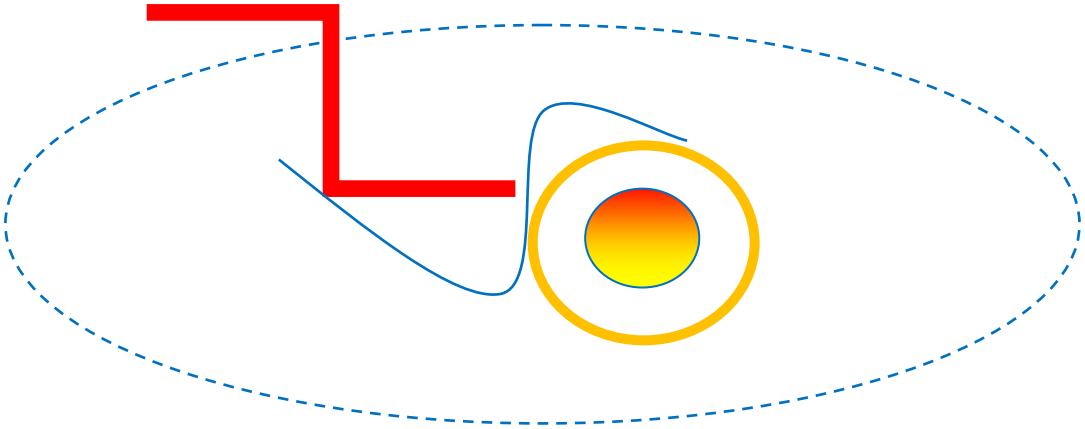
`CubicCurve2D, Arc2D, Ellipse2D, RoundRectangle2D, GeneralPath`

Вариант 4



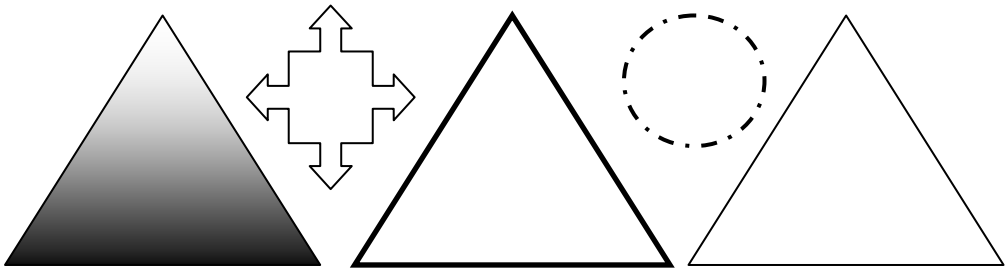
`CubicCurve2D, Arc2D, GeneralPath`

Вариант 5



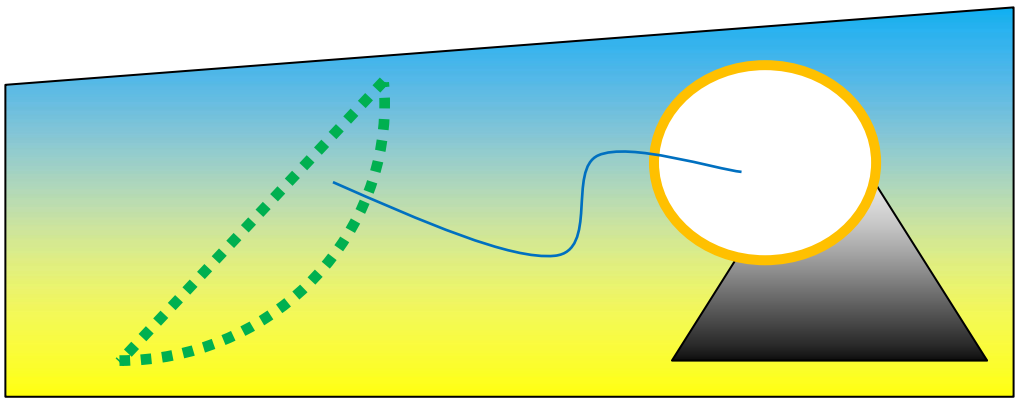
`CubicCurve2D, Arc2D, Ellipse2D, GeneralPath`

Вариант 6



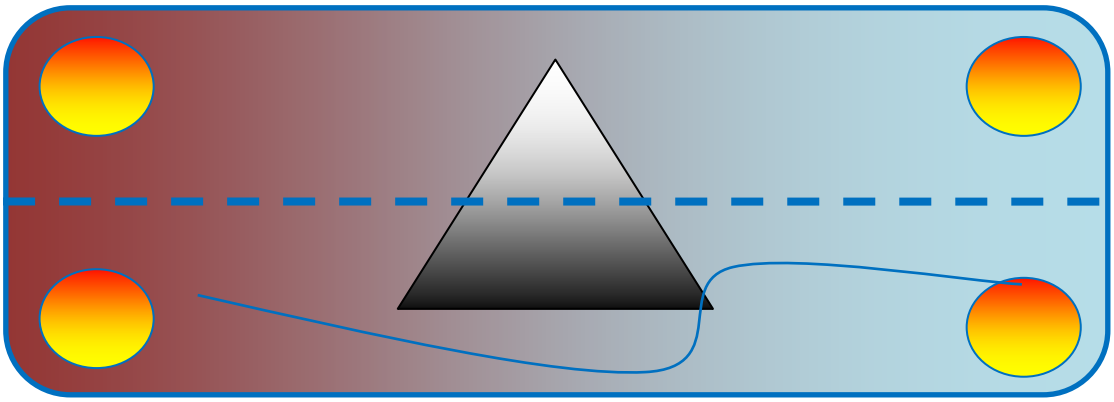
`Ellipse2D, GeneralPath`

Вариант 7



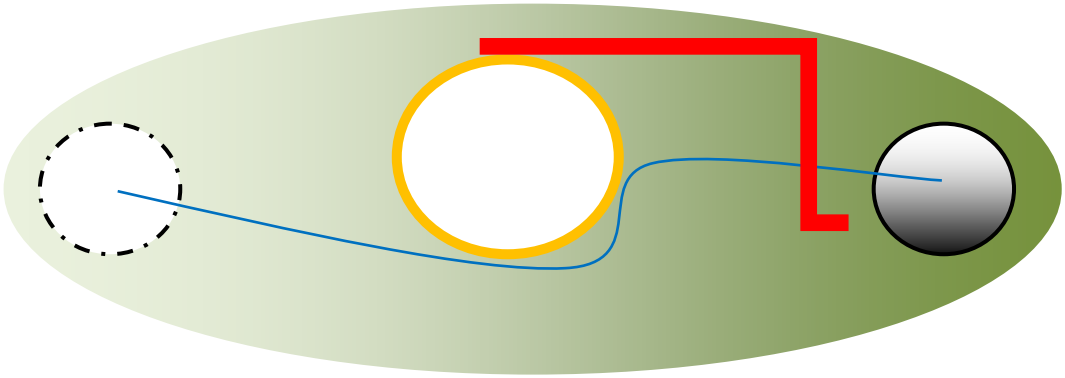
`CubicCurve2D, Arc2D, Ellipse2D, GeneralPath`

Вариант 8



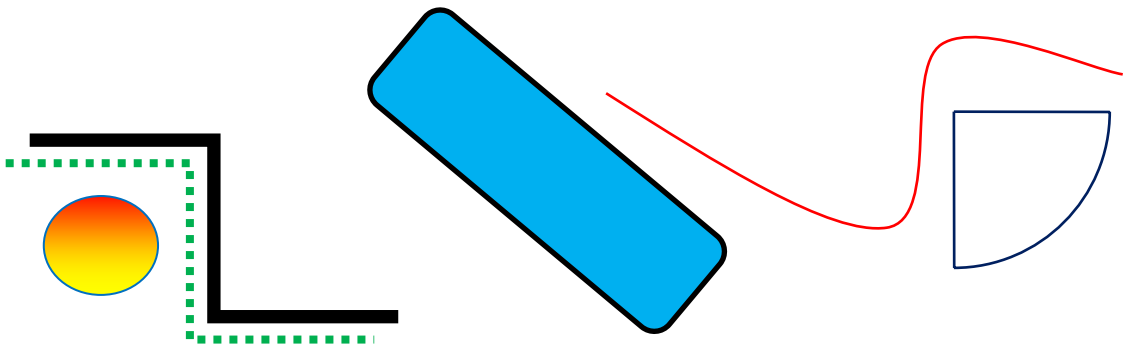
`CubicCurve2D, Arc2D, Ellipse2D, GeneralPath`

Вариант 9



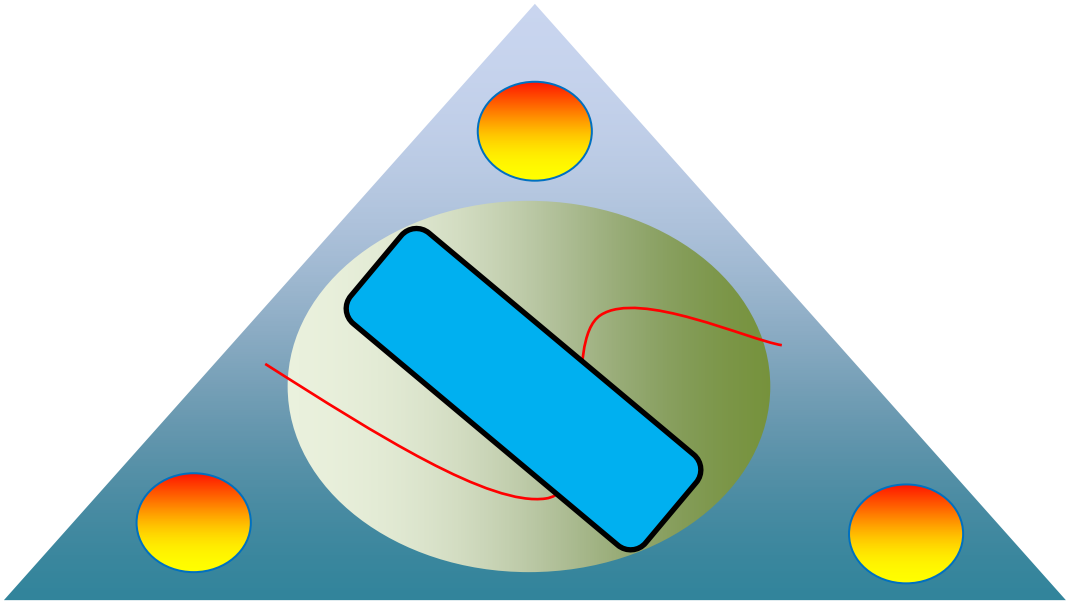
`CubicCurve2D, Ellipse2D, GeneralPath`

Вариант 10



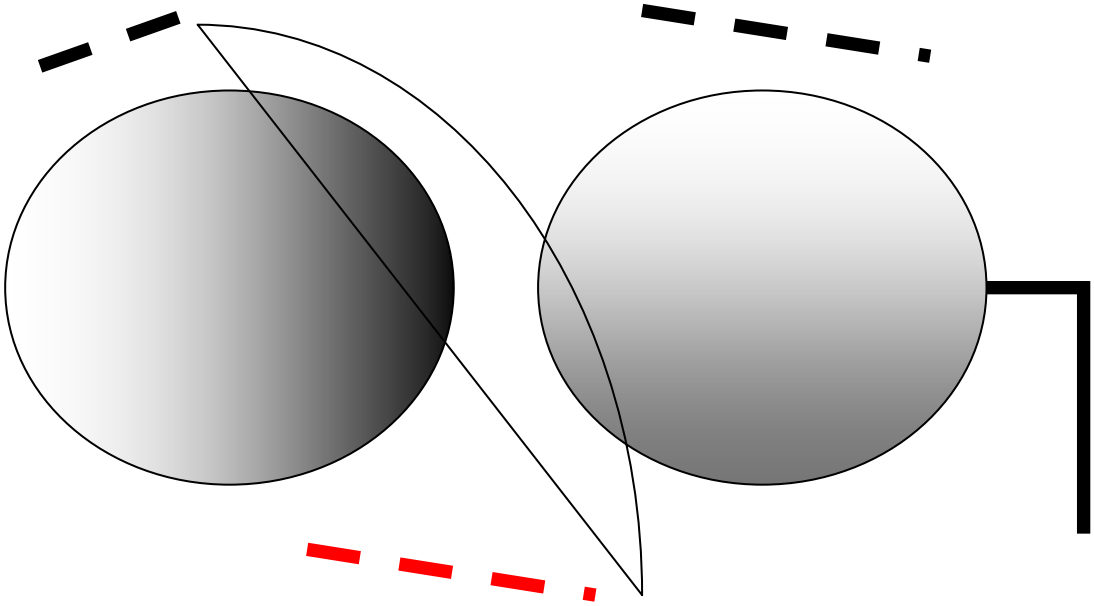
`CubicCurve2D, Arc2D, Ellipse2D, RoundRectangle2D, GeneralPath`

Вариант 11



`CubicCurve2D, Ellipse2D, RoundRectangle2D`

Вариант 12



`Arc2D, Ellipse2D, GeneralPath`