ТЕМА 4
Отображение графики

Цель лабораторной работы

Научиться составлять простейшие оконные приложения, отображающие графику (примитивы геометрических фигур, текст).

1 Особенности отображения графики в приложениях

Как мы уже знаем, в оконных приложениях Java за отображение любого компонента отвечает метод `paintComponent()`, получающий в качестве аргумента объект класса `Graphics`. Этот объект представляет некоторый виртуальный *холст*, на котором происходит рисование, хранит набор параметров для отображения изображений и текста, а также предоставляет для этого ряд методов. При каждой перерисовке окна (по любой причине) обработчик события перерисовки уведомляет компонент. Это приводит к вызову метода `paintComponent()` всех компонентов интерфейса.

По этой причине, стандартным способом вывода графики в приложениях является использование классов-потомков стандартных компонентов пользовательского интерфейса библиотеки Swing (например, `JPanel`, `JButton` и т.д.), с переопределением в них метода `paintComponent()`.

Так как реализация данного метода в базовых классах обеспечивает визуализацию стандартного вида соответствующих элементов интерфейса на экране, то, как правило, первой инструкцией переопределённой версии метода является обращение к методу предка (что обеспечивает показ на экране, например, кнопки, выпадающего списка и т.д.). Все последующие команды вывода графики будут отображаться «поверх» его стандартного вида, что позволяет изменять его требуемым образом. При необходимости полностью изменить внешний вид компонента, отобразив его по-другому, унаследованная версия может не вызываться.

Одним из наиболее предпочтительных кандидатов на использование в качестве базового класса при разработке собственного компонента, отображающего графику на экране, является класс `JPanel`. Причин для этого несколько:

- при компоновке панель масштабируется таким образом, чтобы заполнить всё доступное пространство;
- панель является контейнером, т.е. может содержать вложенные компоненты;
- метод `paintComponent()` панели только закрашивает всё её пространство цветом заднего фона и ничего не отображает сверху.

Сам процесс вывода графики состоит из последовательности операций с виртуальным холстом, соответствующим пространству, занимаемому отображаемым компонентом.

В Java версии 1.0 средства отображения графики были весьма простыми: выбирался цвет и режим «заливки» фона, после чего вызывались методы класса `Graphics`, такие как `drawRect()` и `fillOval()`. Начало координат холста (точка с координатами (0,0)) располагалась в левом верхнем углу отображаемого компонента, а измерения выполнялись в точках, поэтому все координаты были целочисленными. Некоторые из методов класса `Graphics`, осуществляющих непосредственный вывод графики, представлены в таблице 1.1:

Таблица 1.1 – Некоторые методы вывода графики класса `Graphics`

Название метода	Описание
`drawLine`	Рисует линию
`drawRect`	Рисует контур прямоугольника
`drawRoundRect`	Рисует контур прямоугольника с закруглёнными краями
`draw3DRect`	Рисует контур прямоугольника с эффектом объёма
`drawPolygon()`	Рисует замкнутый многоугольник
`drawPolyline()`	Рисует ломаную линию
`drawOval()`	Рисует овал
`drawArc()`	Рисует дугу

Начиная с Java версии 2.0, в состав J2SE включена библиотека Java 2D, реализующая широкий набор графических операций. Её особенностями являются:

- Управление *пером*, т.е. типом линий, которыми прорисовываются границы фигур.
- Создание *заливки* с однородными цветами, градиентами, повторяющимися шаблонами.
- Использование *трансформаций* для перемещения, масштабирования, поворота, растягивания фигур.
- Использование *кадрирования* для ограничения области перерисовки.
- Использование правил композиции для описания способов объединения точек новой фигуры с существующими точками.
- Настройка *ориентиров визуализации* для задания компромисса между скоростью и качеством прорисовки.
- Вывод графики на основе *объектных представлений геометрических фигур*, например `Line2D`, `Rectangle2D` (таблица 1.2). Все классы фигур реализуют общий интерфейс `Shape` (фигура).

Таблица 1.2 – Некоторые классы фигур библиотеки Java 2D

Название метода	Описание
`Point2D`	Представляет точку, с координатами (x,y). Используется для задания фигур, но сам фигурой не является.
`Line2D`	Фигура, представляющая линию
`Rectangle2D`	Фигура, представляющая прямоугольник
`RoundRectangle2D`	Фигура, представляющая прямоугольник с закруглёнными краями
`Ellipse2D`	Фигура, представляющая эллипс (круг)
`Arc2D`	Фигура, представляющая дугу
`QuadCurve2D`	Фигура, представляющая кривую второго порядка
`QubicCurve2D`	Фигура, представляющая кривую третьего порядка
`GeneralPath`	Фигура, состоящая из множества сегментов, каждый из которых может быть линией, кривой второго или третьего порядка.

- *Использование вещественных координат*, что в ряде случаев является большим удобством, так как позволяет задавать координаты фигур в осмысленных единицах (метрах, дюймах, километрах), затем автоматически преобразуя их в точки на основе заданных коэффициентов масштаба. В Java 2D для внутренних вычислений используются числа типа `float`, точности которых достаточно. Тем не менее, так как многие методы в Java работают с вещественными числами повышенной точности (`double`), а `float` и `double` напрямую несовместимы (требуется явное преобразование типов), для удобства программиста в Java 2D предусмотрены *две версии* каждого класса фигур: использующих координаты одинарной `float` и двойной точности `double` (рисунок 1.1).

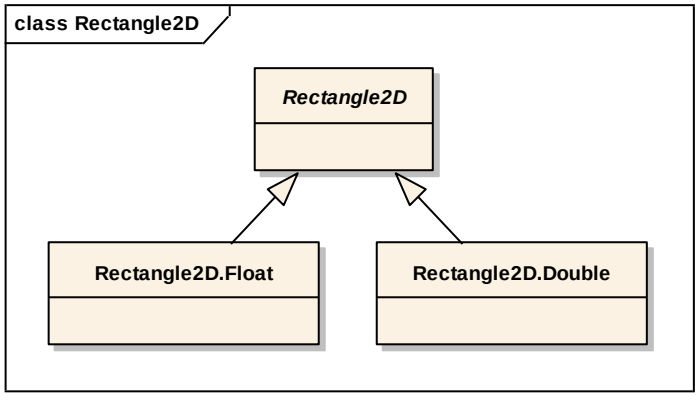

Рисунок 1.1 – Две версии класса для вещественных координат различной точности на примере Rectangle2D

Для доступа к расширенным возможностям библиотеки Java 2D необходимо в явном виде привести экземпляр класса `Graphics` (например, полученный как аргумент в методе `paintComponent()`) к классу `Graphics2D`.

При инициализации экземпляров фигур библиотеки Java 2D непосредственной их визуализации не происходит. Фигура отображается

только при обращении к методу `draw()` или `fill()` класса `Graphics2D`. В этот момент новая фигура обрабатывается в *конвейере визуализации* (рисунок 1.2).

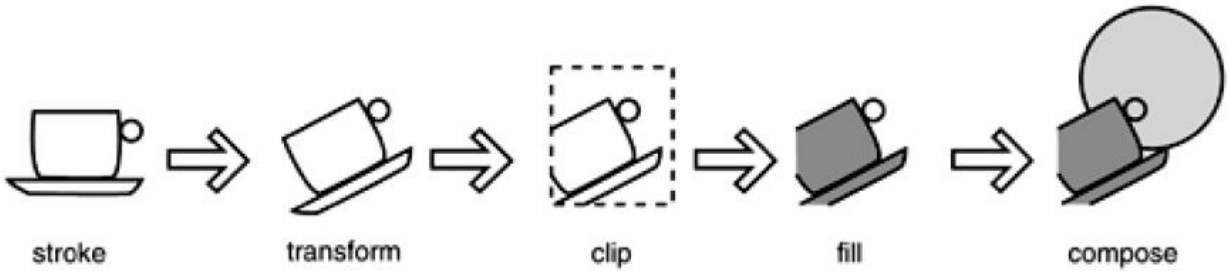

Рисунок 1.2 – Схема обработки фигуры в конвейере визуализации

Как видно на рисунке, последовательность этапов следующая:

- прорисовывается контур фигуры;
- к контуру применяются заданные преобразования (сдвига, поворота и т.п.);
- фигура кадрируется (если фигура и область кадрирования не пересекаются, то процесс визуализации прекращается);
- оставшаяся после кадрирования часть фигуры закрашивается;
- точки закрашенной фигуры совмещаются с существующими точками.

2 Краткая справка по необходимым программным компонентам

2.1 Представление контекста отображения – класс Graphics2D

Является представлением поверхности, на которой происходит рисование, и предоставляет ряд методов для настройки параметров отображения и непосредственного рисования (таблица 2.1).

Для получения экземпляра класса необходимо осуществить приведение к типу `Graphics2D` экземпляра `Graphics`, получаемого в метод `paintComponent()` в качестве аргумента:

```
public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D canvas = (Graphics2D) g;
        // Дальнейшие действия с экземпляром класса
        // ...

}
```

Таблица 2.1 – Некоторые часто используемые методы класса `Graphics2D`

Название метода	Описание
`draw(Shape)`	Рисует контур фигуры с учётом настроек пера, кадрирования,

	трансформации, заливки, композиции
`fill(Shape)`	Заполняет внутреннюю часть фигуры с использованием настроек трансформации, кадрирования, заливки и композиции
`Color getColor()`	Возвращает текущий цвет холста
`Font getFont()`	Возвращает текущий шрифт холста
`Paint getPaint()`	Возвращает текущую заливку холста
`Stroke getStroke()`	Возвращает текущее перо холста
`setColor(Color)`	Устанавливает для холста текущий цвет
`setFont(Font)`	Устанавливает для холста текущий шрифт
`setPaint(Paint)`	Устанавливает для холста текущую заливку
`setStroke(Stroke)`	Устанавливает для холста текущее перо

При изменении текущих настроек холста хорошим стилем считается сохранение предыдущих настроек до их использования и восстановление старых значений после использования. В следующем фрагменте сохраняются и восстанавливаются значения текущего цвета и заливки:

```
// Сохранение текущих настроек
Color oldColor = canvas.getColor();
Paint oldPaint = canvas.getPaint();
// Установка новых настроек
canvas.setColor(Color.RED);
canvas.setPaint(Color.BLUE);
// Рисование с новыми настройками
// ...
// Восстановление старых настроек
canvas.setColor(oldColor);
canvas.setPaint(oldPaint);
```

2.2 Классы геометрических фигур

2.2.1 Представление линии – класс Line2D

Для отображения линий в библиотеке Java 2D предназначен абстрактный класс `Line2D` с двумя потомками `Line2D.Float` (представляющий координаты с помощью `float`) и `Line2D.Double` (представляющий координаты с помощью `double`).

Существуют два альтернативных способа инициализации экземпляра линии (не считая конструктора без параметров).

Способ 1: задать точки линии как экземпляры класса `Point2D.Double`:

```
Point2D.Double from = new Point2D.Double(0, 0);
Point2D.Double to = new Point2D.Double(1, 1);
Line2D.Double line = new Line2D.Double(from, to);
```

Способ 2: задать точки линии как пары чисел типа `double`:

```
Line2D.Double line = new Line2D.Double(0.0, 0.0, 1.0, 1.0);
```

При любом способе инициализации линии, для её показа на экране необходимо обратиться к методу `draw()` класса `Graphics2D`, например:

```
// canvas – экземпляр класса Graphics2D
canvas.draw(line);
```

2.2.2 Представление эллипса – класс Ellipse2D

Для отображения эллипсов и окружностей в библиотеке Java 2D предназначен абстрактный класс `Ellipse2D` с двумя потомками `Ellipse2D.Float` (представляющий координаты с помощью `float`) и `Ellipse2D.Double` (представляющий координаты с помощью `double`).

Существуют два альтернативных способа инициализации экземпляра эллипса.

Способ 1: задать эллипс с помощью ограничивающего прямоугольника, в который вписан эллипс:

```
/* Эллипс вписывается в прямоугольник, верхний левый угол
которого размещён в точке (-2,-2) с длинами сторон 4x4. Так как такой
прямоугольник является квадратом, то получается  окружность с центром в
точке (0,0) и радиусом 2.
*/
Ellipse2D.Double circle1 = new Ellipse2D.Double(-2, -2, 4, 4);
```

Способ 2: создать эллипс с помощью конструктора без параметров, а затем задать его размеры с помощью методов `setFrameFromCenter()` и `setFrameFromDiagonal()`:

```
// Инициализировать экземпляр класса констуктором по умолчанию
Ellipse2D.Double circle2 = new Ellipse2D.Double();
/* Задать окружность как эллипс, вписанный в прямоугольник, координаты
точек диагонами которого (-2,-2) и (2,2)
*/
circle2.setFrameFromDiagonal(-2, -2, 2, 2);
// Аналогично, но через Point2D.Double
circle2.setFrameFromDiagonal(new Point2D.Double(-2, -2),
                             new Point2D.Double(2, 2));
/* Задать окружность как эллипс, вписанный в прямоугольник с центром в
точке (5,5), один из углов которого находится в точке (8, 8)
*/
circle2.setFrameFromCenter(5, 5, 8, 8);
// Аналогично, но через Point2D.Double
circle2.setFrameFromCenter(new Point2D.Double(5, 5),
                           new Point2D.Double(8, 8));
```

При любом способе инициализации эллипса, для его показа на экране необходимо обратиться к методу `draw()`, а для заливки – к методу `fill()` класса `Graphics2D`, например:

```
// canvas – экземпляр класса Graphics2D
canvas.draw(circle1);
canvas.draw(circle2);
canvas.fill(circle2);
```

2.2.3 Представление сложного пути – класс GeneralPath

Для задания сложных линий, состоящих из различных сегментов, в библиотеке Java 2D используется класс `GeneralPath` (общий *путь*). Одним из способом создлания его экземпляров является использование конструктора по умолчанию:

```
// Создать новый экземпляр сложной линии
GeneralPath path = new GeneralPath();
```

Сегментами пути могут являться как прямые линии (`Line2D`), так и кривые второго (`QuadCurve2D`) и третьего (`QubicCurve2D`) порядка. Существует два способа добавления сегментов в путь.

Способ 1: предполагает создание экземпляров линий и добавление их в путь с помощью метода `append()`:

```
// Линия из (0,0) в (2,2)
path.append(new Line2D.Double(0, 0, 2, 2), true);
// Линия из (2,2) в (2,6)
path.append(new Line2D.Double(2, 2, 2, 6), true);
```

Способ 2: предполагает задание текущей точки пути с помощью метода `moveTo()`, а затем добавление сегментов пути с помощью методов `lineTo()`, `curveTo()`, `quadTo()`:

```
// Установить текущую точку пути в (0,0)
path.moveTo(0, 0);
// Вести линию из точки (0,0) в точку (2,2).
// (2, 2) при этом становится текущей
path.lineTo(2, 2);
// Вести линию из точки (2,2) в точку (2,6)
// (2, 6) становится текущей
path.lineTo(2, 6);
```

При необходимости, из пути можно сделать замкнутый многоугольник, обратившись к методу `closePath()`, например в следующем фрагменте кода из пути, включающего две прямых, соединяющихся под углом в 90 градусов, создаётся фигура прямоугольного треугольника:

```
// Текущая точка (0,0)
path.moveTo(0, 0);
// Вести линию в точку (-2, 2)
path.lineTo(-2, 2);
// Вести линию в точку (2, 2)
path.lineTo(2, 2);
// Замкнуть треугольник - в данном случае это эквивалентно
// path.lineTo(0, 0);
path.closePath();
```

После создания пути, его, как и другие объекты Java 2D, необходимо отобразить с помощью вызова:

```
canvas.draw(circle1);
```

2.3 Задание типа линий – класс BasicStroke

Операция отображения `draw()` класса `Graphics2D` прочерчивает границу фигуры с использованием текущих настроек *пера*. По умолчанию, перо чертит сплошную линию толщиной в 1 точку. С помощью метода `setStroke()` класса `Graphics2D` можно изменять настройки пера, передавая методу в качестве аргумента класс, реализующий интерфейс `Stroke`. В библиотеке Java 2D единственным классом, реализующим данный интерфейс, является класс `BasicStroke`. Для инициализации объектов данного класса предложен ряд конструкторов, наиболее сложный из которых позволяет при создании нового пера задавать целый ряд параметров:

- толщину линий;
- оформление края линий;
- оформление смыкания линий;
- предельный угол смыкания линий;
- шаблон линий;
- начальное смещение в шаблоне.

Толщина линии задаётся вещественным числом типа `float`, и может быть как целым, так и дробным значением.

Для **оформления края линий** на выбор предложено три варианта (рисунок 2.1):

Рисунок 2.1 – Стили оформления края линий

Стиль `CAP_BUTT` делает край линии резко обрубленным, `CAP_ROUND` – завершает его полукругом, `CAP_SQUARE` – завершает его половиной квадрата (сторона которого равна толщине линии).

Задание **способа оформления смыкания линий** позволяет указать, каким из трёх вариантов будет оформлено смыкание двух линий (рисунок 2.2):

Рисунок 2.2 – Стили оформления смыкания линий

Стиль соединения `JOIN_BEVEL` соединяет смыкающиеся линии прямой, перпендикулярной биссектрисе угла между ними, `JOIN_ROUND` – дополняет конец каждой линии полукруглым завершением, `JOIN_MITER` – удлиняет обе линии, образуя острый «шип».

Тип `JOIN_MITER` не приспособлен для оформления соединений линий, смыкающихся под небольшими углами, так как это может выразиться в очень длинных «шипах». Для предотвращения этого эффекта, вводится такой параметр как **предельный угол смыкания линий** (по умолчанию – 10 градусов). Если две линии смыкаются под углом, не превышающим значения предельного, вместо стиля `JOIN_MITER` используется стиль `JOIN_BEVEL`.

Шаблон линий задаётся в как массив вещественных чисел типа `float`, каждое из которых определяет длину чередующихся отрезков «линия есть» - «линии нет» (рисунок 2.3).

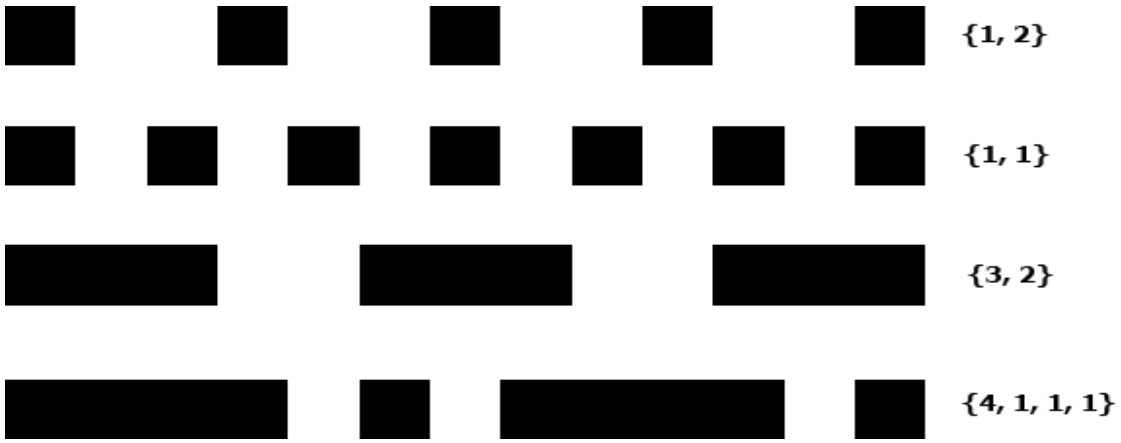

Рисунок 2.3 – Примеры некоторых шаблонов линий

Параметр **начального смещения в шаблоне** задаёт, в каком месте шаблона должно начинаться отображение каждой линии (по умолчанию – это значение равно 0).

В следующем фрагменте кода определены два типа линий:

```
/* Линия толщиной 2 точки, резко обрублена на краях,
 * смыкания линий оформлены полукругами,
 * предельный угол смыкания - 10 градусов,
 * шаблон - пунктирный (****    ****    ****),
 * смещение в шаблоне - 0 точек
 */
BasicStroke myStroke1 = new BasicStroke(2.0f, BasicStroke.CAP_BUTT,
      BasicStroke.JOIN_ROUND, 10.0f, new float[] {4, 4}, 0.0f);
/* Линия толщиной 0.5 точки, резко обрублена на краях,
 * смыкания линий оформлены острыми шипами,
 * предельный угол смыкания - 10 градусов,
 * шаблон - штрих-пунктирный (****  *  ****  *  ****),
 * смещение в шаблоне - 0 точек
 */
BasicStroke myStroke2 = new BasicStroke(0.5f, BasicStroke.CAP_BUTT,
      BasicStroke.JOIN_MITER, 10.0f, new float[] {4, 2, 1, 2}, 0.0f);
```

2.4 Задание способа заливки – интерфейс Paint

Операция заливки замкнутой области `fill()` класса `Graphics2D` использует текущие настройки заливки. Для задания альтернативного способа заливки применяется метод `setPaint()`, которому в качестве аргумента необходимо передать класс, реализующий интерфейс `Paint`.

Одним из классов, его реализующих, является класс `Color`, поэтому для заполнения фигур однородным цветом необходимо обратиться к методу `setPaint()`, передав в качестве аргумента объект класса Color, например:

```
canvas.setPaint(Color.RED);
```

Более сложные способы заливки обеспечивает класс `GradientPaint` (градиентная заливка), интерполируя цветовые оттенки между двумя заданными значениями. Объекты класса инициализируются посредством конструктора, которому передаются две точки и цвета, которые должны быть в этих точках, например:

```
// Задать исходную точку для градиента
Point2D from = new Point2D.Double(0, 0);
// Задать конечную точку для градиента
Point2D to = new Point2D.Double(10, 10);
// Установить новый способ заливки фигур
canvas.setPaint(new GradientPaint(from, Color.RED, to, Color.BLUE));
```

Цвета изменяются вдоль линии, соединяющей две заданные точки, и постоянны на линиях, перпендикулярных ей. Точки, находящиеся далее краевых точек имеют цвета краевых точек.

Дополнительным параметром, который может быть передан конструктору, является флаг периодичности заливки. Если он установлен в true, то цвета циклически продолжают изменяться и за пределами отрезка, ограниченного двумя точками:

```
// Такая градиентная заливка будет циклически повторяться
canvas.setPaint(new GradientPaint(from,Color.RED,to,Color.BLUE,true));
```

2.5 Задание шрифта отображения надписей – класс Font

Для вывода текста и надписей используется метод `drawString()` класса `Graphics2D`, принимающий в качестве аргументов строку *s* и отображающий её в точке с координатами (*x*, *y*). Отображение осуществляется с использованием текущего шрифта (вначале, текущим шрифтом является шрифт по умолчанию). Для отображения символов каким-либо особенным шрифтом, необходимо создать экземпляр класса `Font`, для чего указывается имя шрифта, его стиль и размер, например:

```
// Создать шрифт с именем “Serif”, жирный, размер 36 пунктов
Font myFont1 = new Font("Serif", Font.BOLD, 36);
```

Так как 100%-ного способа обеспечить наличие определённого шрифта на компьютере пользователя нет, то для задания некоторой определённости в библиотеке AWT определено пять *логических имён* шрифтов:

- SansSerif
- Serif
- Monospaced
- Dialog
- DialogInput

Эти шрифты всегда связываются со шрифтами, которые действительно присутствуют на компьютере пользователя. Например, в ОС Windows логическое имя *SansSerif* будет связано с шрифтом *Arial*.

Замечание: при инициализации экземпляра класса `Font` можно указывать как имена, так и логические имена шрифтов.

При задании стиля шрифта возможны варианты:

- обычный (`Font.PLAIN`);
- **жирный** (`Font.BOLD`);
- *курсив* (`Font.ITALIC`);
- ***жирный курсив*** (`Font.BOLD + Font.ITALIC`).

Начиная с J2SE версии 1.3, стало возможным использовать шрифты *TrueType*.Для использования шрифта его сначала нужно загрузить из потока ввода – обычно файла в файловой системе или удалённого файла. Получив объект открытого потока ввода, следует обратиться к статическому методу `Font.createFont()`. Объект, возвращённый в качестве результата выполнения метода, будет представлять шрифт размером 1 пункт. Для установки другого размера шрифта следует воспользоваться методом `deriveFont()`, например:

```
// Открыть поток чтения из файла mySuperFont.ttf
InputStream in = new FileInputStream(new File("mySuperFont.ttf"));
// Создать объект шрифта TrueType размером 1 пт
Font mySuperFont = Font.createFont(Font.TRUETYPE_FONT, in);
// На основе созданного шрифта, создать шрифт размером 14пт
Font mySuperFont14pt = mySuperFont.deriveFont(14.0f);
```

Достаточно часто возникает задача размещения текста или надписи определённым образом по отношению к ориентирам (например, центрирование на экране, выравнивание по верхнему краю рисунка и т.п.). Для её выполнения необходимо знать ширину и высоту (в точках) надписи. Эти размеры зависят от трёх факторов:

- используемого шрифта;

- выводимой строки;
- устройства, на котором отображается текст (экран, принтер и т.п.).

Для получения объекта, характеризующего свойства устройства, используется метод `getFontRenderContext()` класса `Graphics2D`. Возвращаемый им экземпляр класса `FontRenderContext` может использоваться для вычисления размеров надписи с помощью метода `getStringBounds()` класса `Font`:

```
// Создать объект контекста отображения текста - для получения
// характеристик устройства (экрана)
FontRenderContext context = canvas.getFontRenderContext();
// Получить размеры области для отображения надписи "Hello!"
Rectangle2D bounds = mySuperFont.getStringBounds("Hello!", context);
```

Для того, чтобы интерпретировать значения, возвращаемые методом `getStringBounds()`, следует знать некоторые термины, имеющие отношение к анатомии шрифта (рисунок 2.4):

Рисунок 2.4 – Анатомия шрифтовой надписи

Базовая линия – это воображаемая линия, на которой находятся нижние части большинства символов. *Аскент* – это расстояние от базовой линии до верхней части высоких символов, таких как *b*, *k* и т.д., или символов верхнего регистра. *Дескент* – это расстояние от базовой линии до нижней части низких символов, таких как *p*, *g* и т.д. *Интерлиньяж* – это расстояние между дескентом одной линии и аскентом другой. *Высота* шрифта – это расстояние между последовательными базовыми линиями, то есть аскент + дескент + интерлиньяж.

Ширина прямоугольника, возвращаемого методом `getStringBounds()`, представляет ширину отображаемой надписи. Высота прямоугольника равна сумме аскента и дескента. Начало координат прямоугольника находится на базовой линии. Верхняя y-координата прямоугольника отрицательна. Зная все эти особенности, можно вычислить все характеристики шрифта следующим образом:

```
double stringWidth = bounds.getWidth();
double stringHeight = bounds.getHeight();
double ascent = -bounds.getY();
double descent = bounds.getHeight() + bounds.getY();
```

Для вычисления интерлиньяжа и полной высоты шрифта следует воспользоваться методом `getLineMetrics()` класса `Font`, который возвращает объект класса `LineMetrics`. Методы-селекторы `getHeight()` и `getLeading()` полученного объекта позволяют узнать интерлиньяж и высоту шрифта:

```
LineMetrics metrics = mySuperFont.getLineMetrics(message, context);
float fontHeight = metrics.getHeight();
float leading = metrics.getLeading();
```

2.6 Управление состоянием элементов меню

В лабораторной работе №3 управление состоянием элементов меню (задание состояния их доступности или недоступности) осуществлялось непосредственно в обработчиках событий, т.е. если нажатие кнопки приводило к изменению состояния элемента меню, то соответствующие инструкции были указаны внутри тела обработчика:

```
// Нажатие на кнопку «Вычислить» делает элементы меню доступными
buttonCalc.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent ev) {
		// ...
		// Пометить элементы меню как доступные
		saveToTextMenuItem.setEnabled(true);
		saveToGraphicsMenuItem.setEnabled(true);
		searchValueMenuItem.setEnabled(true);
	}
});
// Нажатие на кнопу «Очистить поля» делает элементы недоступными
buttonReset.addActionListener(new ActionListener() {
	public void actionPerformed(ActionEvent ev) {
		// ...
		// Пометить элементы меню как недоступные
		saveToTextMenuItem.setEnabled(false);
		saveToGraphicsMenuItem.setEnabled(false);
		searchValueMenuItem.setEnabled(false);
	}
});
```

В данной работе мы реализуем альтернативный способ управления состоянием элементов меню, основанный на создании класса-слушателя событий, связанных с показом меню. Интерфейс `MenuListener` определяет три метода, вызываемых в различные моменты показа меню:

- `menuSelected()` вызывается после события активации меню (пользователь щёлкнул мышью на пункте меню), но до отображения раскрытого вида меню на экране;
- `menuDeselected()` вызывается после того, как меню исчезло с экрана;

- `menuCanceled()` вызывается в случае отмены выбора пункта меню, но это событие является очень редким.

В большинстве случаев интерес для нас будет представлять только метод `menuSelected()`, но так как реализация интерфейса `MenuListener` требует реализации всех определённых в нём методов, то методы `menuDeselected()` и `menuCanceled()` также необходимо реализовать, но можно сделать пустыми.

В большинстве случаев состояние элементов меню определяется переменными, не являющимися переменными экземпляра класса, реализующего `MenuListener`. Существует два способа доступа к подобным переменным.

Способ 1: добавление в класс, реализующий интерфейс `MenuListener`, внутреннего поля, содержащего ссылку на объект с необходимыми переменными.

Способ 2: объявление класса, реализующего интерфейс `MenuListener`, вложенным классом по отношению к классу с необходимыми переменными. В этом случае, будучи вложенным классом, он имеет доступ ко всем (включая `private`) полям наружного класса.

После реализации класса-обработчика событий, связанных с показом меню, его необходимо связать с экземпляром меню приложения с помощью метода `addMenuListener()` класса `JMenu`.

Наиболее простым и удобным, в большинстве случаев, является второй способ. Его реализация приведена в следующем фрагменте кода:

```
public class MainFrame extends JFrame {

	// Флаг, указывающий на доступность данных
	private boolean dataAvailable = false;
	// Пункты меню, состоянием которых управляем
	private JMenuItem menuItem1 = new JMenuItem();
	private JMenuItem menuItem2 = new JMenuItem();

	// ... часть кода пропущена ...

	// Конструктор класса
	public MainFrame() {
		// ... часть кода пропущена ...
		// Создать полосу меню
		JMenuBar menuBar = new JMenuBar();
		// Установить полосу меню вверху окна приложения
		setJMenuBar(menuBar);
		// Создать меню "Файл"
		JMenu fileMenu = new JMenu("Файл");
		// Добавить меню "Файл" в полосу меню
		menuBar.add(fileMenu);
		// Добавить два элемента в меню "Файл"
		fileMenu.add(menuItem1);
		fileMenu.add(menuItem2);
		// Зарегистрировать слушатель событий для меню "Файл"
		fileMenu.addMenuListener(new FileMenuListener());
```

```
        }

        // Класс-слушатель событий, связанных с отображением меню «Файл»
        private class FileMenuListener implements MenuListener {

              // Обработчик, вызываемый перед показом меню
              public void menuSelected(MenuEvent e) {
                    // Доступность или недоступность элементов
                    // определяется доступностью данных
                    menuItem1.setEnabled(dataAvailable);
                    menuItem2.setEnabled(dataAvailable);
              }

              // Обработчик, вызываемый после того, как меню исчезло с экрана
              public void menuDeselected(MenuEvent e) {
              }

              // Обработчик, вызываемый в случае отмены выбора пункта меню
              public void menuCanceled(MenuEvent e) {
              }
        }
}
```

3 Пример приложения

Задание: составить программу отображения графика функции по точкам, записанным в файле. Координаты каждой точки описываются парой чисел типа `Double`. Программа должна позволять управлять настройками отображения графика:

- оси координат включены/выключены;
- маркеры точек графика включены/выключены.

Изменение названных настроек должно осуществляться выставлением соответствующих флажков в элементах меню «*График*». До загрузки данных через элемент меню «*Файл* → *Открыть файл с графиком*» элементы меню «*График*» должны быть недоступны.

Внешний вид окна приложения представлен на рисунке 3.1.

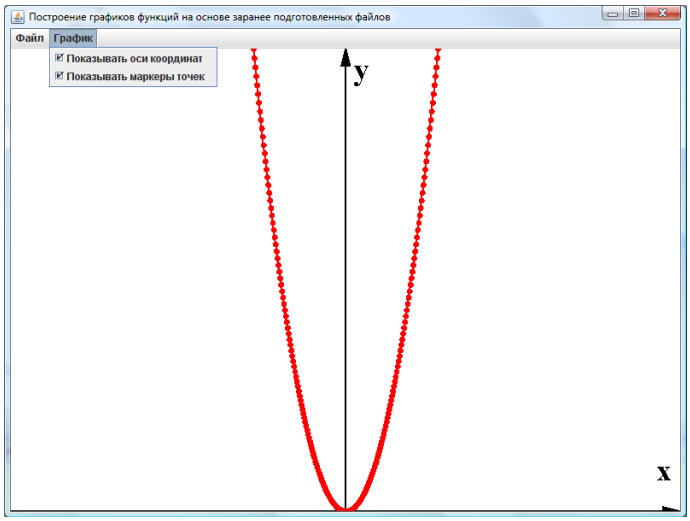

Рисунок 3.1 – Внешний вид главного окна приложения

3.1 Структура приложения

Главное окно приложения `MainFrame`, как и в лабораторной работе №3, будет организовывать интерфейс пользователя посредством меню, реагировать на действия пользователя посредством обработчиков событий от элементов меню, а также выполнять загрузку данных из файла.

Управление состоянием элементов меню будет выполняться посредством класса-слушателя событий от меню `GraphicsMenuListener`.

Для отображения графика функции на основе массива точек мы разработаем отдельный класс `GraphicsDisplay`, являющийся потомком `JPanel`.

Диаграмма классов приложения представлена на рисунке 3.2.

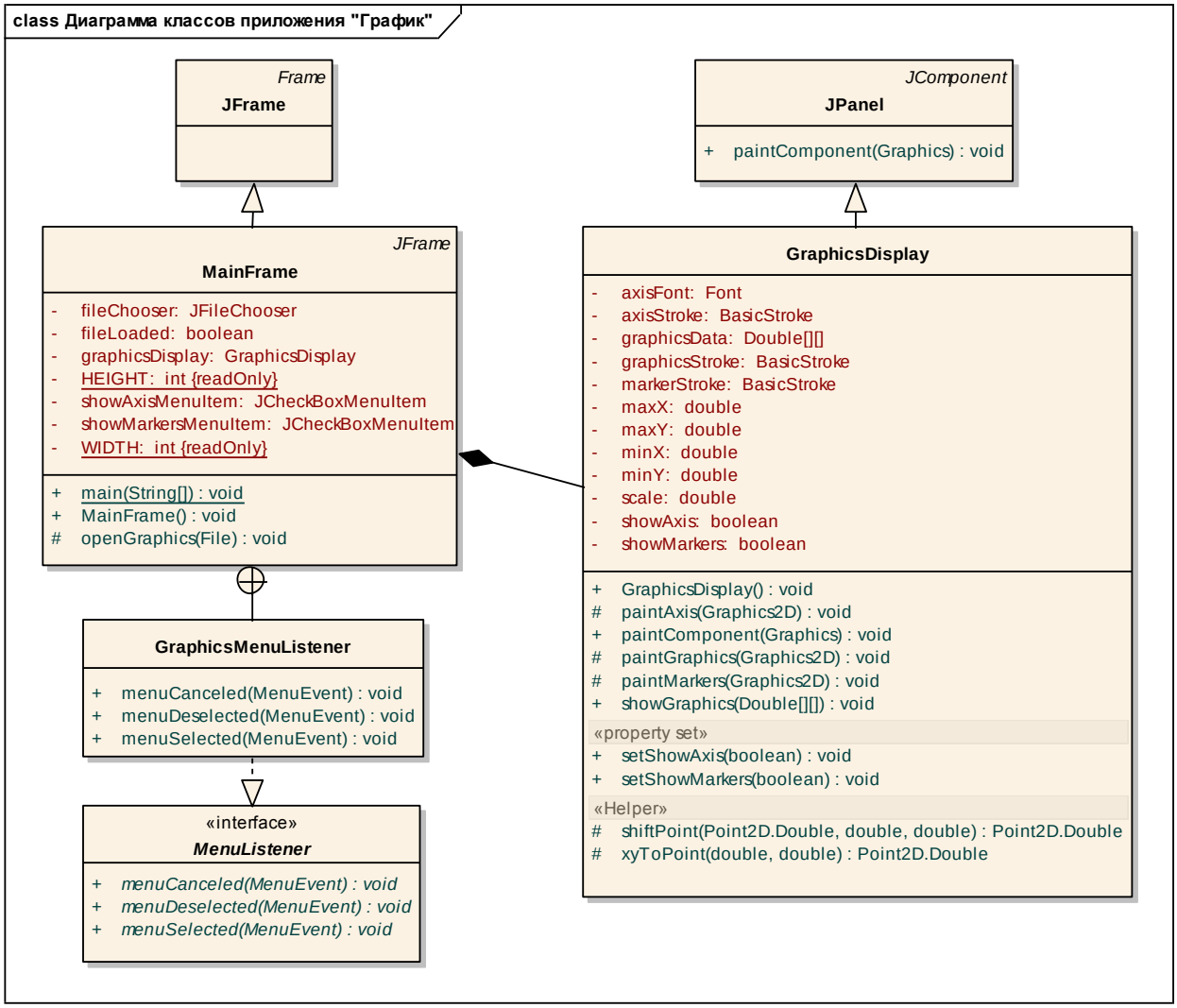

Рисунок 3.2 – Диаграмма классов приложения

3.2 Подготовительный этап

Этап предполагает создание каркаса приложения (см. шаги 3.1 – 3.4 лабораторной работы №1).

3.3 Реализация класса GraphicsDisplay

Класс `GraphicsDisplay` является потомком класса `JPanel` и обеспечивает отображение графика функции на основе массива координат его точек.

Следует обратить внимание на то, что необходимо выполнять преобразование координат между значениями точек графика и координатами точек на экране. Причин для этого две:

- различная ориентация систем координат – точка (0,0) холста находится в левом верхнем углу окна, координата *y* увеличивается вниз; в Декартовой системе координат *y* увеличивается вверх;
- различный масштаб отображения, зависящий от размеров окна приложения.

В библиотеке Java 2D предусмотрены средства преобразования и масштабирования систем координат, но их использование является нетривиальным. Вместо этого, в данной работе мы будем использовать собственные механизмы пересчёта координат, для чего предназначены два метода-помощника – `xyToPoint()` и `shiftPoint()`. Основные принципы преобразования координат показаны на рисунке 3.3.

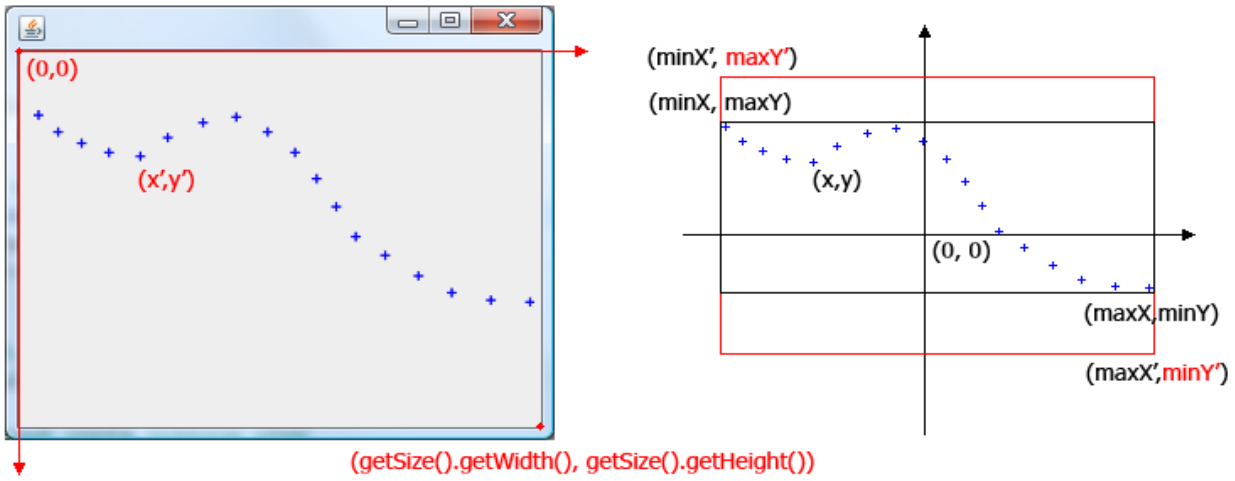

Рисунок 3.3 – Принцип преобразования координат при отображении точек на экране

Будем полагать, что график функции находится в области, ограниченной прямоугольником с вершинами в точках (*minX*, *maxY*) и (*maxX*, *minY*). Значения *minX*, *maxX*, *minY*, *maxY* находятся с помощью поиска минимального и максимального значений в массиве.

Из рисунка видно, что преобразование координат состоит из двух составляющих: масштабирования и параллельного переноса.

Для масштабирования следует вычислить коэффициенты масштаба по обеим координатам:

$$K_x = \frac{\text{Ширина окна}}{\text{Размер области по } x} = \frac{\text{getSize().getWidth()}}{maxX - minX}$$

$$K_y = \frac{\text{Высота окна}}{\text{Размер области по } y} = \frac{\text{getSize().getHeight()}}{maxY - minY}$$

Для того, чтобы график не искажался, коэффициент масштаба по обеим осям должен быть одинаков. Если выбрать максимальный из них, что часть изображения по другой координате окажется за пределами окна, поэтому в качестве общего масштаба выбирается минимальный из коэффициентов:

$$K = Min(K_x, K_y)$$

Это, в свою очередь, выразится в том, что по оси координат, обладающей большим масштабом, в окно поместится большая область пространства, чем исходная. Например, на рисунке видно, что коэффициент масштаба по оси *x* меньше, чем коэффициент по оси *y*. Он выбирается в качестве общего. Но с таким коэффициентом, области окна по вертикали будет достаточно для показа области не только от *minY* до *maxY*, а от *minY'* до *maxY'*, где *minY'<minY, maxY'>maxY*.

Скорректированные координаты границы области для случая, когда общим коэффициентом масштаба выбран коэффициент по оси *x*, вычисляются следующим образом:

$$maxY' = maxY' + \frac{\frac{\text{getSize().getHeight()}}{K} - (maxY - minY)}{2}$$

$$minY' = minY' - \frac{\frac{\text{getSize().getHeight()}}{K} - (maxY - minY)}{2}$$

Для случая, когда общим коэффициентом масштаба выбран коэффициент по оси *y*, *maxY* и *minY* остаются неизменными, а по схожим формулам изменяются *minX* и *maxX*:

$$maxX' = maxX' + \frac{\frac{\text{getSize().getWidth()}}{K} - (maxX - minX)}{2}$$

$$minX' = minX' - \frac{\frac{\text{getSize().getWidth()}}{K} - (maxX - minX)}{2}$$

Зная координаты отображаемой области пространства (*minX'*, *maxY'*) – (*maxX'*, *minY'*) преобразование значений координат в Декартовом пространстве в координаты точки холста выполняется следующим образом:

$$x' = (x - minX') \cdot K$$

$$y' = (maxY' - y) \cdot K$$

Отображение графика на экране реализуется в методе `paintComponent()`, унаследованном от предка `JPanel()`. Так как вывод дополнительных элементов изображения (осей и маркеров точек) является независимым, и (в зависимости от значения булевских переменных) может не выполняться, то спроектируем метод `paintComponent()` модульным. Отображение осей координат будет выполняется методом `paintAxis()`, маркеров точек – методом `paintMarkers()`, графика – методом `paintGraphics()`. Сам метод `paintComponent()` будет определять размер отображаемой области пространства, вычислять коэффициент масштаба и при необходимости вызывать вспомогательные методы отображения.

Так как переменные `showAxis` и `showMarkers` являются закрытыми полями данных класса, то для изменения их значения предназначены два метода-модификатора – `setShowAxis()` и `setShowMarkers()`.

3.3.1 Определение полей экземпляров класса

Проведя анализ перечня объектов, необходимых классу `GraphicsDisplay`, по известным нам критериям, имеем следующий набор полей:

Таблица 3.1 – Поля экземпляров класса GraphicsDisplay

Имя объекта	Сценарий использования
Константы	
Отсутствуют	
Совместно используемые объекты	
`Double[][] graphicsData`	Двумерный массив координат точек, используется многими методами
`double minX`	Координаты границ отображаемой области
`double maxX`	

`double minY` `double maxY`	пространства, используются многими методами
`double scale`	Единый коэффициент масштаба, используется многими методами
`boolean showAxis` `boolean showMarkers`	Флаги, задающие параметры отображения графика, используются многими методами
Повторно используемые объекты	
`Font axisFont`	Шрифт для вывода подписей осей координат
`BasicStroke axisStroke`	Тип пера для черчения осей координат
`BasicStroke graphicsStroke`	Тип пера для черчения линии графика
`BasicStroke markerStroke`	Тип пера для черчения контуров маркеров

Таким образом, внутренние поля данных класса главного окна описываются следующим фрагментом кода:

```
// Список координат точек для построения графика
private Double[][] graphicsData;

// Флаговые переменные, задающие правила отображения графика
private boolean showAxis = true;
private boolean showMarkers = true;

// Границы диапазона пространства, подлежащего отображению
private double minX;
private double maxX;
private double minY;
private double maxY;

// Используемый масштаб отображения
private double scale;

// Различные стили черчения линий
private BasicStroke graphicsStroke;
private BasicStroke axisStroke;
private BasicStroke markerStroke;

// Шрифт отображения подписей к осям координат
private Font axisFont;
```

3.3.2 Реализация конструктора класса GraphicsDisplay()

В конструкторе класса выполняется инициализация повторно используемых объектов, представляющих стили черчения линий и шрифт написания:

```
public GraphicsDisplay() {
	// Цвет заднего фона области отображения - белый
	setBackground(Color.WHITE);
	// Сконструировать необходимые объекты, используемые в рисовании
	// Перо для рисования графика
	graphicsStroke = new BasicStroke(2.0f, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_ROUND, 10.0f, null, 0.0f);
```

```
            // Перо для рисования осей координат
            axisStroke = new BasicStroke(2.0f, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_MITER, 10.0f, null, 0.0f);
            // Перо для рисования контуров маркеров
            markerStroke = new BasicStroke(1.0f, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_MITER, 10.0f, null, 0.0f);
            // Шрифт для подписей осей координат
            axisFont = new Font("Serif", Font.BOLD, 36);
      }
```

3.3.3 Реализация метода показа нового графика showGraphics()

Так как отображение графика на поверхности компонента реализовано методом `paintComponent()` и происходит автоматически при любой его перерисовке, то для отображения точек графика, только что загруженного из файла, нет необходимости выполнять какие-либо дополнительные действия, достаточно сохранить массив координат во внутреннем поле экземпляра класса и запросить перерисовку компонента. Это и делает метод `showGraphics()`, вызываемый из обработчика события активации элемента меню «*Открыть файл с графиком*» главного окна приложения в случае успешной загрузки данных:

```
// Метод вызывается из обработчика элемента меню  "Открыть файл с графиком"
// главного окна приложения в случае успешной загрузки данных
public void showGraphics(Double[][] graphicsData) {
      // Сохранить массив точек во внутреннем поле класса
      this.graphicsData = graphicsData;
      // Запросить перерисовку компонента (неявно вызвать paintComponent())
      repaint();
}
```

3.3.4 Реализация методов-модификаторов параметров отображения графики setShowAxis() и setShowMarkers()

Булевские переменные `showAxis` и `showMarkers` являются закрытыми полями данных экземпляров класса. Для их установки из обработчиков меню главного окна приложения необходимо предусмотреть соответствующие методы-модификаторы `setShowAxis()`:

```
      // Методы-модификаторы для изменения параметров отображения графика
      // Изменение любого параметра приводит к перерисовке области
      public void setShowAxis(boolean showAxis) {
            this.showAxis = showAxis;
            repaint();
      }
```

и `setShowMarkers()`:

```
      public void setShowMarkers(boolean showMarkers) {
            this.showMarkers = showMarkers;
            repaint();
```

```
}
```

Так как изменение параметров отображения графика обусловит другой внешний вид графика, после изменения их значений необходимо запросить обновление внешнего вида компонента с помощью метода `repaint()`.

3.3.5 Реализация методов-помощников для преобразования координат xyToPoint() и shiftPoint()

Метод-помощник `xyToPoint()` реализует описанный выше алгоритм пересчёта координат из Декартовой системы в систему холста отображения, для чего использует вычисленные значения границ области отображения пространства *minX* и *maxY*:

```
protected Point2D.Double xyToPoint(double x, double y) {
      // Вычисляем смещение X от самой левой точки (minX)
      double deltaX = x - minX;
      // Вычисляем смещение Y от точки верхней точки (maxY)
      double deltaY = maxY - y;
      return new Point2D.Double(deltaX*scale, deltaY*scale);
}
```

В ряде случаев, например при рисовании осей координат, возникает необходимость получения точек экрана, представляемых экземплярами класса `Point2D`, отстоящих от заданной точки на некоторое количество пикселов по горизонтали и вертикали. В библиотеке Java 2D не существует стандартного метода, выполняющего подобную операцию, поэтому для её выполнения предназначен метод-помощник `shiftPoint()`:

```
protected Point2D.Double shiftPoint(Point2D.Double src,
                                    double deltaX, double deltaY) {
      // Инициализировать новый экземпляр точки
      Point2D.Double dest = new Point2D.Double();
      // Задать её координаты как координаты существующей точки +
      // заданные смещения
      dest.setLocation(src.getX() + deltaX, src.getY() + deltaY);
      return dest;
}
```

3.3.6 Реализация метода отображения линии графика paintGraphics()

Для отображения линии графика воспользуемся классом `GeneralPath` библиотеки Java 2D, позволяющим сконструировать сложную линию, состоящую из ряда сегментов. Для получения координат точек линии воспользуемся реализованным методом-помощником `xyToPoint()`:

```
protected void paintGraphics(Graphics2D canvas) {
      // Выбрать линию для рисования графика
      canvas.setStroke(graphicsStroke);
```

```
        // Выбрать цвет линии
        canvas.setColor(Color.RED);
        /* Будем рисовать линию графика как путь, состоящий из множества
        сегментов (GeneralPath). Начало пути устанавливается в первую точку
        графика, после чего прямой соединяется со следующими точками */
        GeneralPath graphics = new GeneralPath();
        for (int i=0; i<graphicsData.length; i++) {
              // Преобразовать значения (x,y) в точку на экране point
              Point2D.Double point = xyToPoint(graphicsData[i][0],
                                               graphicsData[i][1]);
              if (i>0) {
                    // Не первая итерация – вести линию в точку point
                    graphics.lineTo(point.getX(), point.getY());
              } else {
                    // Первая итерация - установить начало пути в точку point
                    graphics.moveTo(point.getX(), point.getY());
              }
        }
        // Отобразить график
        canvas.draw(graphics);
}
```

Замечание: обратите внимание, что данный метод получает для рисования уже экземпляр класса `Graphics2D`, т.е. приведение экземпляра класса `Graphics` (передаваемого в `paintComponent()` как аргумент) к классу `Graphics2D` происходит за его пределами.

3.3.7 Реализация метода отображения осей координат paintAxis()

Метод отображения осей координат `paintAxis()`, как и метод отображения линии графика `paintGraphics()`, в качестве аргумента получает готовый к использованию экземпляр класса `Graphics2D`, и может сразу приступать к рисованию:

```
protected void paintAxis(Graphics2D canvas) {
        // Шаг 1 – установить необходимые настройки рисования
        // Установить особое начертание для осей
        canvas.setStroke(axisStroke);
        // Оси рисуются чёрным цветом
        canvas.setColor(Color.BLACK);
        // Стрелки заливаются чёрным цветом
        canvas.setPaint(Color.BLACK);
        // Подписи к координатным осям делаются специальным шрифтом
        canvas.setFont(axisFont);
        // Создать объект контекста отображения текста - для получения
        // характеристик устройства (экрана)
        FontRenderContext context = canvas.getFontRenderContext();
        // Шаг 2 - Определить, должна ли быть видна ось Y на графике
        if (minX<=0.0 && maxX>=0.0) {
              // Она видна, если левая граница показываемой области minX<=0.0,
              // а правая (maxX) >= 0.0
              // Шаг 2а - ось Y - это линия между точками (0, maxY) и (0, minY)
              canvas.draw(new Line2D.Double(xyToPoint(0, maxY), xyToPoint(0,
minY)));
              // Шаг 2б - Стрелка оси Y
              GeneralPath arrow = new GeneralPath();
              // Установить начальную точку ломаной точно на верхний конец оси Y
```

```
            Point2D.Double lineEnd = xyToPoint(0, maxY);
            arrow.moveTo(lineEnd.getX(), lineEnd.getY());
            // Вести левый "скат" стрелки в точку с относительными
            // координатами (5,20)
            arrow.lineTo(arrow.getCurrentPoint().getX()+5,
                         arrow.getCurrentPoint().getY()+20);
            // Вести нижнюю часть стрелки в точку с относительными
            // координатами (-10, 0)
            arrow.lineTo(arrow.getCurrentPoint().getX()-10,
                         arrow.getCurrentPoint().getY());
            // Замкнуть треугольник стрелки
            arrow.closePath();
            canvas.draw(arrow); // Нарисовать стрелку
            canvas.fill(arrow);     // Закрасить стрелку
            // Шаг 2в - Нарисовать подпись к оси Y
            // Определить, сколько места понадобится для надписи “y”
            Rectangle2D bounds = axisFont.getStringBounds("y", context);
            Point2D.Double labelPos = xyToPoint(0, maxY);
            // Вывести надпись в точке с вычисленными координатами
            canvas.drawString("y", (float)labelPos.getX() + 10,
                                   (float)(labelPos.getY() - bounds.getY()));
      }
      // Шаг 3 - Определить, должна ли быть видна ось X на графике
      if (minY<=0.0 && maxY>=0.0) {
            // Она видна, если верхняя граница показываемой области max)>=0.0,
            // а нижняя (minY) <= 0.0
            // Шаг 3а - ось X - это линия между точками (minX, 0) и (maxX, 0)
            canvas.draw(new Line2D.Double(xyToPoint(minX, 0),
                                          xyToPoint(maxX, 0)));
            // Шаг 3б - Стрелка оси X
            GeneralPath arrow = new GeneralPath();
            // Установить начальную точку ломаной точно на правый конец оси X
            Point2D.Double lineEnd = xyToPoint(maxX, 0);
            arrow.moveTo(lineEnd.getX(), lineEnd.getY());
            // Вести верхний "скат" стрелки в точку с относительными
            // координатами (-20,-5)
            arrow.lineTo(arrow.getCurrentPoint().getX()-20,
                         arrow.getCurrentPoint().getY()-5);
            // Вести левую часть стрелки в точку
            // с относительными координатами (0, 10)
            arrow.lineTo(arrow.getCurrentPoint().getX(),
                         arrow.getCurrentPoint().getY()+10);
            // Замкнуть треугольник стрелки
            arrow.closePath();
            canvas.draw(arrow); // Нарисовать стрелку
            canvas.fill(arrow);     // Закрасить стрелку
            // Шаг 3в - Нарисовать подпись к оси X
            // Определить, сколько места понадобится для надписи “x”
            Rectangle2D bounds = axisFont.getStringBounds("x", context);
            Point2D.Double labelPos = xyToPoint(maxX, 0);
            // Вывести надпись в точке с вычисленными координатами
            canvas.drawString("x",
                  (float)(labelPos.getX()-bounds.getWidth()-10),
                  (float)(labelPos.getY() + bounds.getY()));
      }
}
```

На шаге 1 устанавливаются необходимые настройки рисования – тип пера и цвет линий, используемый шрифт. На шаге 2 проверяется, попадает ли ось *y* в область отображения, если да – то на шаге 2а рисуется сама ось, на

шаге 2б – как сложная фигура рисуется треугольная стрелка к ней, на шаге 3в – подпись к оси. На шаге 3 проверяется, попадает ли ось *x* в область отображения, если да – то по аналогии с отображением оси *y* выводятся сама ось, стрелка и подпись. Как и в других методах, для преобразования координат точек в систему координат холста используется метод-помощник `xyToPoint()`.

3.3.8 Реализация метода отображения маркеров точек графика paintMarkers()

Метод отображения маркеров точек `paintMarkers()`, как и метод отображения линии графика `paintGraphics()`, в качестве аргумента получает готовый к использованию экземпляр класса `Graphics2D`, и может сразу приступать к рисованию:

```
protected void paintMarkers(Graphics2D canvas) {
      // Шаг 1 - Установить специальное перо для черчения контуров маркеров
      canvas.setStroke(markerStroke);
      // Выбрать красный цвета для контуров маркеров
      canvas.setColor(Color.RED);
      // Выбрать красный цвет для закрашивания маркеров внутри
      canvas.setPaint(Color.RED);
      // Шаг 2 - Организовать цикл по всем точкам графика
      for (Double[] point: graphicsData) {
            // Инициализировать эллипс как объект для представления маркера
            Ellipse2D.Double marker = new Ellipse2D.Double();
            /* Эллипс будет задаваться посредством указания координат его
                  центра и угла прямоугольника, в который он вписан */
            // Центр - в точке (x,y)
            Point2D.Double center = xyToPoint(point[0], point[1]);
            // Угол прямоугольника - отстоит на расстоянии (3,3)
            Point2D.Double corner = shiftPoint(center, 3, 3);
            // Задать эллипс по центру и диагонали
            marker.setFrameFromCenter(center, corner);
            canvas.draw(marker);    // Начертить контур маркера
            canvas.fill(marker);    // Залить внутреннюю область маркера
      }
}
```

На шаге 1 устанавливаются необходимые настройки рисования – цвет и тип линий, способ заливки. На шаге 2 организуется цикл по всем точкам графика, и с центром в точке и радиусом 3 пиксела рисуется маркер в виде закрашенного круга. Как и в других методах, для преобразования координат точек в систему координат холста используется метод-помощник `xyToPoint()`.

3.3.9 Реализация метода перерисовки компонента paintComponent()

Метод `paintComponent()` является переопределённой версией метода, унаследованного от предка `JPanel`, вызываемой при каждой перерисовке внешнего вида компонента. В качестве аргумента метод получает экземпляр

класса `Graphics`, который почти сразу приводится к классу `Graphics2D` для получения доступа к расширенным возможностям библиотеки Java 2D. В задачи метода входит определение коэффициента масштаба и границ отображаемой области пространства, сохранение и восстановление настроек рисования, а также вызов (по необходимости) методов, осуществляющих отображение элементов графика:

```
public void paintComponent(Graphics g) {
	/* Шаг 1 - Вызвать метод предка для заливки области цветом заднего фона
	* Эта функциональность - единственное, что осталось в наследство от
	* paintComponent класса JPanel
	*/
	super.paintComponent(g);
	// Шаг 2 - Если данные графика не загружены (при показе компонента при
	// запуске программы) - ничего не делать
	if (graphicsData==null || graphicsData.length==0) return;
	// Шаг 3 - Определить начальные границы области отображения
	// Её верхний левый угол - (minX, maxY), правый нижний - (maxX, minY)
	minX = graphicsData[0][0];
	maxX = graphicsData[graphicsData.length-1][0];
	minY = graphicsData[0][1];
	maxY = minY;
	// Найти минимальное и максимальное значение функции
	for (int i = 1; i<graphicsData.length; i++) {
		if (graphicsData[i][1]<minY) {
			minY = graphicsData[i][1];
		}
		if (graphicsData[i][1]>maxY) {
			maxY = graphicsData[i][1];
		}
	}
	/* Шаг 4 - Определить (исходя из размеров окна) масштабы по осям X и Y -
		сколько пикселов  приходится на единицу длины по X и по Y */
	double scaleX = getSize().getWidth() / (maxX - minX);
	double scaleY = getSize().getHeight() / (maxY - minY);
	// Выбрать единый масштаб как минимальный из двух
	scale = Math.min(scaleX, scaleY);
	// Шаг 5 - корректировка границ области согласно выбранному масштабу
	if (scale==scaleX) {
		/* Если за основу был взят масштаб по оси X, значит по оси Y
	делений меньше, т.е. подлежащий отображению диапазон по Y будет меньше
	высоты окна. Значит необходимо добавить делений, сделаем это так:
	1) Вычислим, сколько делений влезет по Y при выбранном масштабе -
		getSize().getHeight()/scale;
	2) Вычтем из этого значения  сколько делений требовалось изначально;
	3) Набросим по половине недостающего расстояния на maxY и minY */
		double yIncrement = (getSize().getHeight()/scale-(maxY-minY))/2;
		maxY += yIncrement;
		minY -= yIncrement;
	}
	if (scale==scaleY) {
	// Если за основу был взят масштаб по оси Y, действовать по аналогии
		double xIncrement = (getSize().getWidth()/scale-(maxX-minX))/2;
		maxX += xIncrement;
		minX -= xIncrement;
	}
	// Шаг 5 - Преобразовать экземпляр Graphics к Graphics2D
	Graphics2D canvas = (Graphics2D) g;
	// Шаг 6 - Сохранить текущие настройки холста
	Stroke oldStroke = canvas.getStroke();
```

```
        Color oldColor = canvas.getColor();
        Paint oldPaint = canvas.getPaint();
        Font oldFont = canvas.getFont();
        // Шаг 8 - В нужном порядке вызвать методы отображения элементов графика
        // Порядок вызова методов имеет значение, т.к. предыдущий рисунок будет
        // затираться последующим
        // Первым (если нужно) отрисовываются оси координат.
        if (showAxis) paintAxis(canvas);
        // Затем отображается сам график
        paintGraphics(canvas);
        // Затем (если нужно) отображаются маркеры точек графика.
        if (showMarkers) paintMarkers(canvas);
        // Шаг 9 - Восстановить старые настройки холста
        canvas.setFont(oldFont);
        canvas.setPaint(oldPaint);
        canvas.setColor(oldColor);
        canvas.setStroke(oldStroke);
}
```

На шаге 1 вызывается унаследованная версия метода для заливки всего пространства выбранным цветом фона.

На шаге 2 проверяется наличие данных графика. Если данные не загружены, то дальнейшее выполнение метода прекращается (нечего рисовать).

На шаге 3 посредством поиска минимальных и максимальных значений в массиве определяются исходные границы области отображения пространства.

На шаге 4, исходя из размеров окна и исходных границ области отображения, вычисляются коэффициенты масштаба по осям координат и выбирается единый коэффициент масштаба.

На шаге 5 осуществляется корректировка границ области отображения с учётом выбранного единого коэффициента масштаба.

На шаге 6 происходит приведение экземпляра класса `Graphics`, переданного в метод как аргумент, к типу `Graphics2D`, используемому в методах отображения элементов графика.

На шаге 7 выполняется сохранение текущих настроек рисования.

На шаге 8, с помощью вызова необходимых методов, выполняется непосредственное отображение всех требуемых элементов графика.

На шаге 9 восстанавливаются сохранённые настройки рисования.

3.4 Реализация класса GraphicsMenuListener

Класс `GraphicsMenuListener` является внутренним классом по отношению к классу главного окна `MainFrame` и реализует интерфейс `MenuListener` и отвечает за изменение состояния элементов меню «*График*» в зависимости от загруженности данных графика: если поле `fileLoaded` – истина, то элементы меню доступны, в противном случае – недоступны:

```
        // Класс-слушатель событий, связанных с отображением меню
        private class GraphicsMenuListener implements MenuListener {

                // Обработчик, вызываемый перед показом меню
                public void menuSelected(MenuEvent e) {
                        // Доступность или недоступность элементов меню "График"
определяется загруженностью данных
                        showAxisMenuItem.setEnabled(fileLoaded);
                        showMarkersMenuItem.setEnabled(fileLoaded);
                }

                // Обработчик, вызываемый после того, как меню исчезло с экрана
                public void menuDeselected(MenuEvent e) {
                }

                // Обработчик, вызываемый в случае отмены выбора пункта меню
(очень редкая ситуация)
                public void menuCanceled(MenuEvent e) {
                }

        }
```

3.5 Реализация главного класса приложения MainFrame

Главный класс приложения должен обеспечить показ окна, содержащего элементы интерфейса пользователя – полосу меню, включающую два пункта «*Файл*» и «*График*», и компонент `GraphicsDisplay`, занимающий всё пространство окна и обеспечивающий отображение графика. Кроме того, он должен обеспечить реакцию на действия пользователя и по запросу выполнять загрузку данных из файла, указанного пользователем с помощью диалогового окна «*Открыть файл*».

3.5.1 Определение полей экземпляров класса

Проанализировав по известным нам критериям сценарии использования объектов разрабатываемого приложения, имеем следующий набор полей:

Таблица 3.2 – Поля экземпляров класса MainFrame

Имя объекта	Сценарий использования
Константы	
`WIDTH`	Исходный размер окна по горизонтали
`HEIGHT`	Исходный размер окна по вертикали
Совместно используемые объекты	
`boolean fileLoaded`	Флаг, указывающий на загруженность данных. Совместно используется в обработчиках событий.
`GraphicsDisplay display`	Компонент, обеспечивающий отображение

	графика. Компонуется в конструкторе, используется в обработчиках событий.
`JCheckBoxMenuItem showAxisMenuItem`	Элементы меню, создаются в конструкторе, используются в обработчике событий меню.
`JCheckBoxMenuItem showMarkersMenuItem`	
Повторно используемые объекты	
`JFileChooser`	Компонент диалогового окна выбора файла

Таким образом, внутренние поля данных класса главного окна описываются следующим фрагментом кода:

```
// Начальные размеры окна приложения
private static final int WIDTH = 800;
private static final int HEIGHT = 600;

// Объект диалогового окна для выбора файлов
private JFileChooser fileChooser = null;

// Пункты меню
private JCheckBoxMenuItem showAxisMenuItem;
private JCheckBoxMenuItem showMarkersMenuItem;

// Компонент-отображатель графика
private GraphicsDisplay display = new GraphicsDisplay();

// Флаг, указывающий на загруженность данных графика
private boolean fileLoaded = false;
```

3.5.2 Реализация конструктора окна MainFrame()

Конструктор окна обеспечивает задание интерфейса пользователя, а также задание реакций на его действия. Первым этапом является вызов конструктора предка (`Frame`), масштабирование и позиционирование окна:

```
// Вызов конструктора предка Frame
super("Построение графиков функций на основе подготовленных файлов");
// Установка размеров окна
setSize(WIDTH, HEIGHT);
Toolkit kit = Toolkit.getDefaultToolkit();
// Отцентрировать окно приложения на экране
setLocation((kit.getScreenSize().width - WIDTH)/2,
            (kit.getScreenSize().height - HEIGHT)/2);
// Развёртывание окна на весь экран
setExtendedState(MAXIMIZED_BOTH);
```

Следующим этапом является конструирование полосы меню, включающего меню «*Файл*» и «*График*» и добавление элемента «*Открыть файл с графиком*» в меню «*Файл*»:

```
// Создать и установить полосу меню
JMenuBar menuBar = new JMenuBar();
setJMenuBar(menuBar);
```

```
        // Добавить пункт меню "Файл"
        JMenu fileMenu = new JMenu("Файл");
        menuBar.add(fileMenu);
        // Создать действие по открытию файла
        Action openGraphicsAction = new AbstractAction("Открыть файл") {
              public void actionPerformed(ActionEvent event) {
                    if (fileChooser==null) {
                          fileChooser = new JFileChooser();
                          fileChooser.setCurrentDirectory(new File("."));
                    }
                    if (fileChooser.showOpenDialog(MainFrame.this) ==
                                                JFileChooser.APPROVE_OPTION)
                          openGraphics(fileChooser.getSelectedFile());
              }
        };
        //  Добавить соответствующий элемент меню
        fileMenu.add(openGraphicsAction);
        // Создать пункт меню "График"
        JMenu graphicsMenu = new JMenu("График");
        menuBar.add(graphicsMenu);
```

Далее создаются элементы для меню «*График*» и задаются обработчики событий их активации. Логика действий всех обработчиков одинакова: при включении (выключении) элемента меню, в экземпляре класса `GraphicsDisplay` в аналогичное состояние устанавливается соответствующая булевская переменная, управляющий параметрами отображения графика:

```
        // Создать действие для реакции на активацию элемента
        // "Показывать оси координат"
        Action showAxisAction = new AbstractAction("Показывать оси координат") {
              public void actionPerformed(ActionEvent event) {
                    // свойство showAxis класса GraphicsDisplay истина,
                    // если элемент меню showAxisMenuItem отмечен флажком,
                    // ложь - в противном случае
                    display.setShowAxis(showAxisMenuItem.isSelected());
              }
        };
        showAxisMenuItem = new JCheckBoxMenuItem(showAxisAction);
        // Добавить соответствующий элемент в меню
        graphicsMenu.add(showAxisMenuItem);
        // Элемент по умолчанию включен (отмечен флажком)
        showAxisMenuItem.setSelected(true);
        // Повторить действия для элемента "Показывать маркеры точек"
        Action showMarkersAction = new AbstractAction("Показывать маркеры
точек") {
              public void actionPerformed(ActionEvent event) {
                    // по аналогии с showAxisMenuItem
                    display.setShowMarkers(showMarkersMenuItem.isSelected());
              }
        };
        showMarkersMenuItem = new JCheckBoxMenuItem(showMarkersAction);
        graphicsMenu.add(showMarkersMenuItem);
        // Элемент по умолчанию выключен
        showMarkersMenuItem.setSelected(false);
        // Зарегистрировать обработчик событий, связанных с меню "График"
        graphicsMenu.addMenuListener(new GraphicsMenuListener());
```

Последним шагом, выполняемым конструктором, является установка компонента `GraphicsDisplay` в центральную область компоновки окна приложения:

```
// Установить GraphicsDisplay в цент граничной компоновки
getContentPane().add(display, BorderLayout.CENTER);
```

3.5.3 Реализация чтения данных из файла – метод openGraphics()

При активации пользователем элемента меню «*Открыть файл с графиком*» обработчик этого события отображает диалоговое окно выбора файла, а затем вызывает метод `openGraphics()`, осуществляющий чтение данных из файла:

```
// Считывание данных графика из существующего файла
protected void openGraphics(File selectedFile) {
	try {
		// Шаг 1 - Открыть поток чтения данных, связанный с файлом
		DataInputStream in = new DataInputStream(
					new FileInputStream(selectedFile));
		/* Шаг 2- Зная объём данных в потоке ввода можно вычислить,
		 * сколько памяти нужно зарезервировать в массиве:
		 * Всего байт в потоке - in.available() байт;
		 * Размер числа Double - Double.SIZE бит, или Double.SIZE/8 байт;
		 * Так как числа записываются парами, то число пар меньше в 2 раза
		 */
		Double[][] graphicsData = new
				Double[in.available()/(Double.SIZE/8)/2][];
		// Шаг 3 – Цикл чтения данных (пока в потоке есть данные)
		int i = 0;
		while (in.available()>0) {
			// Первой из потока читается координата точки X
			Double x = in.readDouble();
			// Затем - значение графика Y в точке X
			Double y = in.readDouble();
			// Прочитанная пара координат добавляется в массив
			graphicsData[i++] = new Double[] {x, y};
		}
		// Шаг 4 - Проверка, имеется ли в списке в результате чтения
		// хотя бы одна пара координат
		if (graphicsData!=null && graphicsData.length>0) {
			// Да - установить флаг загруженности данных
			fileLoaded = true;
			// Вызывать метод отображения графика
			display.showGraphics(graphicsData);
		}
		// Шаг 5 - Закрыть входной поток
		in.close();
	} catch (FileNotFoundException ex) {
		// В случае исключительной ситуации типа "Файл не найден"
		// показать сообщение об ошибке
		JOptionPane.showMessageDialog(MainFrame.this,
			"Указанный файл не найден", "Ошибка загрузки данных",
			JOptionPane.WARNING_MESSAGE);
			return;
	} catch (IOException ex) {
		// В случае ошибки ввода из файлового потока
		// показать сообщение об ошибке
```

```
                JOptionPane.showMessageDialog(MainFrame.this,
                        "Ошибка чтения координат точек из файла",
                        "Ошибка загрузки данных", JOptionPane.WARNING_MESSAGE);
                        return;
        }
}
```

Первым шагом метода является конструирование потока чтения данных.

На шаге 2, зная количество байт, доступных для чтения из потока, инициализируется массив достаточного размера.

На шаге 3 в цикле выполняется чтение данных из потока, пока не будет достигнут его конец.

На шаге 4 проверяется, была ли в результате чтения из потока успешно считана хотя бы одна пара координат? Если да, то загрузка данных считается успешной, и прочитанный массив координат передаётся как аргумент методу `showGraphics()` класса `GraphicsDisplay`.

На шаге 5 закрывается входной поток.

Данный метод должен уметь обрабатывать исключительные ситуации типа `FileNotFoundException` и `IOException`. Первая из них может произойти по причине того, что за время, прошедшее между выбором файла в диалоговом окне, и моментом его открытия, другое приложение могло удалить открываемый файл. Причиной второй ситуации может стать нарушение формата файла, например, если в файл было записано нечётное число вещественных чисел (пар координат). В любом случае, на исключительную ситуацию следует отреагировать показом пользователю сообщения об ошибке, для чего всё тело метода помещено в блок `try-catch`.

3.5.4 Реализация главного метода main()

В главном методе класса, как и в лабораторной работе №2, выполняется инициализация и показ окна приложения:

```
public static void main(String[] args) {
        // Создать и показать экземпляр главного окна приложения
        MainFrame frame = new MainFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
}
```

4 Задания

4.1 Вариант А

а) Модифицировать (по вариантам) способ отображения маркеров точек и линии графика. Маркер должен представлять собой фигуру размером 11x11 точек, с центром в точке графика функции.

б) Модифицировать приложение таким образом, чтобы цветом выделялись точки, в которых выполняется условие (по вариантам).

№ п/п	Маркер	Линия	Условие
1	✢		Целая часть значения функции в точке является квадратом целого числа
2	△		Значение функции в точке находится в диапазоне 0.1 от целого числа
3	✥		Целая часть значения функции в точке - чётная
4	◇		Целая часть значения функции в точке - нечётная
5	▽		Сумма цифр в записи целой части значения функции в точке меньше десяти
6	⊕		В записи целой части значения функции в точке используются только чётные цифры
7	⊠		В записи значения функции в точке цифры идут последовательно по возрастанию
8	✳		Значение функции в точке превышает в более чем два раза среднее значение функции на всём интервале

4.2 Вариант В

а) Модифицировать (по вариантам) способ отображения маркеров точек и линии графика. Маркер должен представлять собой фигуру размером 11x11 точек, с центром в точке графика функции.

б) Модифицировать приложение таким образом, чтобы цветом выделялись точки, в которых выполняется условие (по вариантам).

в) Реализовать (по вариантам) дополнительный элемент отображения графика, включить соответствующий элемент в меню «*График*».

№ п/п	Маркер	Линия	Условие	Элемент отображения
1	✳		В записи значения функции в точке цифры идут последовательно по возрастанию	Показать координатную сетку с шагом, равным 1/10 расстояния (*maxX-minX*) по горизонтали и (*maxY-minY*) по вертикали
2	⊠		Значение функции в точке находится в диапазоне 0.1 от целого числа	Показать деления на осях координат с шагом, равным 1/100 расстояния (*maxX-minX*) по горизонтали и (*maxY-minY*) по вертикали. Отметку каждого 5-го деления делать длиннее.
3	⊕		Целая часть значения функции в точке - чётная	Обозначить на графике точку (0,0), а на осях координат обозначить точки и подписать значения *minX*, *maxX*, *minY*, *maxY*.

4	▽		Значение функции в точке превышает в более чем два раза среднее значение функции на всём интервале	Горизонтальными линиями отметить на графике уровни: 90% от (*maxY-minY*); 50% от (*maxY-minY*); 10% от (*maxY-minY*).
5	◇		В записи целой части значения функции в точке используются только чётные цифры	Обозначить на графике все точки локальных экстремумов функции, подписать значения в этих точках
6	✧		Сумма цифр в записи целой части значения функции в точке меньше десяти	Отобразить поверх графика исходной функции *f* график функции «целая часть *f*». Использовать другой цвет и стиль линии.
7	△		Целая часть значения функции в точке является квадратом целого числа	Отобразить поверх графика исходной функции *f* график функции «модуль *f*». Использовать другой цвет и стиль линии.
8	✢		Целая часть значения функции в точке - нечётная	Обозначить на графике все точки, в которых функция обращается нуль. Если в точке №1 с координатой $x1$ значение функции – $y1$, а в точке №2 с координатой $x2$ значение функции – $y2$, и знаки $y1$ и $y2$ различны, то координата x точки, где функция обращается в нуль, может быть приблизительно вычислена как: $$x = \frac{\lvert y_1\rvert \cdot x_2 + \lvert y_2\rvert \cdot x_1}{\lvert y_1\rvert + \lvert y_2\rvert}$$

4.3 Вариант С

а) Модифицировать (по вариантам) способ отображения маркеров точек и линии графика. Маркер должен представлять собой фигуру размером 11x11 точек, с центром в точке графика функции.

б) Модифицировать приложение таким образом, чтобы цветом выделялись точки, в которых выполняется условие (по вариантам).

в) Реализовать (по вариантам) дополнительный элемент отображения графика, включить соответствующий элемент в меню «*График*».

г) Реализовать поворот графика в пространстве окна на 90 градусов влево (ось x направлена вверх, ось y – влево) с использованием механизмов преобразования системы координат Java 2D. График должен занимать всё

пространство окна. Данный режим должен включаться/выключаться через элемент меню «*График*».

№ п/п	Маркер	Линия	Условие	Элемент отображения
1	✳		В записи значения функции в точке цифры идут последовательно по возрастанию	Найти все замкнутые области, образуемые линией графика и осью *x*. Для каждой области вычислить её площадь (площадь под кривой) методом трапеций, закрасить область однородным цветом, в центре области показать значение площади.
2	✢		В записи целой части значения функции в точке используются только чётные цифры	Показывать координатную сетку с подписями координат ячеек и делениями по осям – внутри каждой ячейки сетки должно быть 10 малых отрезков (9 делений), середина ячейки (деление №5 должно быть длиннее). Шаг сетки должен выбираться автоматически таким образом, чтобы: он был точным числом (например, 1, 20, 50, 100, 0.1, 0.2, 0.5, а не 0.1244, 0.3495, 12,34955); по осям помещалось не более 20 делений (шаг был не очень мелким, и не очень крупным).
3	⊕		Сумма цифр в записи целой части значения функции в точке меньше десяти	Обеспечить возможность одновременного отображения (различным цветом и стилем линии) графиков сразу двух функций. При этом отображаемая область пространства должна расширяться таким образом, чтобы вместить оба графика.

Приложение 1. Исходный код приложения

Класс отображения графика GraphicsDisplay

```
package bsu.rfe.java.group7.lab4.Ivanov.varB4;

import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Paint;
import java.awt.Stroke;
import java.awt.font.FontRenderContext;
import java.awt.geom.Ellipse2D;
import java.awt.geom.GeneralPath;
import java.awt.geom.Line2D;
import java.awt.geom.Point2D;
import java.awt.geom.Rectangle2D;
import javax.swing.JPanel;

@SuppressWarnings("serial")
public class GraphicsDisplay extends JPanel {

	// Список координат точек для построения графика
	private Double[][] graphicsData;

	// Флаговые переменные, задающие правила отображения графика
	private boolean showAxis = true;
	private boolean showMarkers = true;

	// Границы диапазона пространства, подлежащего отображению
	private double minX;
	private double maxX;
	private double minY;
	private double maxY;

	// Используемый масштаб отображения
	private double scale;

	// Различные стили черчения линий
	private BasicStroke graphicsStroke;
	private BasicStroke axisStroke;
	private BasicStroke markerStroke;

	// Различные шрифты отображения надписей
	private Font axisFont;

	public GraphicsDisplay() {
		// Цвет заднего фона области отображения - белый
		setBackground(Color.WHITE);
		// Сконструировать необходимые объекты, используемые в рисовании
		// Перо для рисования графика
		graphicsStroke = new BasicStroke(2.0f, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_ROUND, 10.0f, null, 0.0f);
		// Перо для рисования осей координат
		axisStroke = new BasicStroke(2.0f, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_MITER, 10.0f, null, 0.0f);
		// Перо для рисования контуров маркеров
		markerStroke = new BasicStroke(1.0f, BasicStroke.CAP_BUTT,
BasicStroke.JOIN_MITER, 10.0f, null, 0.0f);
		// Шрифт для подписей осей координат
```

```
            axisFont = new Font("Serif", Font.BOLD, 36);
      }

      // Данный метод вызывается из обработчика элемента меню "Открыть файл с
графиком"
      // главного окна приложения в случае успешной загрузки данных
      public void showGraphics(Double[][] graphicsData) {
            // Сохранить массив точек во внутреннем поле класса
            this.graphicsData = graphicsData;
            // Запросить перерисовку компонента, т.е. неявно вызвать
paintComponent()
            repaint();
      }

      // Методы-модификаторы для изменения параметров отображения графика
      // Изменение любого параметра приводит к перерисовке области
      public void setShowAxis(boolean showAxis) {
            this.showAxis = showAxis;
            repaint();
      }

      public void setShowMarkers(boolean showMarkers) {
            this.showMarkers = showMarkers;
            repaint();
      }

      // Метод отображения всего компонента, содержащего график
      public void paintComponent(Graphics g) {
      /* Шаг 1 - Вызвать метод предка для заливки области цветом заднего фона
       * Эта функциональность - единственное, что осталось в наследство от
       * paintComponent класса JPanel
       */
            super.paintComponent(g);
            // Шаг 2 - Если данные графика не загружены (при показе компонента
при запуске программы) - ничего не делать
            if (graphicsData==null || graphicsData.length==0) return;
            // Шаг 3 - Определить минимальное и максимальное значения для
координат X и Y
            // Это необходимо для определения области пространства, подлежащей
отображению
            // Её верхний левый угол это (minX, maxY) - правый нижний это
(maxX, minY)
            minX = graphicsData[0][0];
            maxX = graphicsData[graphicsData.length-1][0];
            minY = graphicsData[0][1];
            maxY = minY;
            // Найти минимальное и максимальное значение функции
            for (int i = 1; i<graphicsData.length; i++) {
                  if (graphicsData[i][1]<minY) {
                        minY = graphicsData[i][1];
                  }
                  if (graphicsData[i][1]>maxY) {
                        maxY = graphicsData[i][1];
                  }
            }
            /* Шаг 4 - Определить (исходя из размеров окна) масштабы по осям X
и Y - сколько пикселов
             * приходится на единицу длины по X и по Y
             */
            double scaleX = getSize().getWidth() / (maxX - minX);
            double scaleY = getSize().getHeight() / (maxY - minY);
            // Шаг 5 - Чтобы изображение было неискажённым - масштаб должен
быть одинаков
            // Выбираем за основу минимальный
```

```
            scale = Math.min(scaleX, scaleY);
            // Шаг 6 - корректировка границ отображаемой области согласно
выбранному масштабу
            if (scale==scaleX) {
                  /* Если за основу был взят масштаб по оси X, значит по оси Y
делений меньше,
                   * т.е. подлежащий визуализации диапазон по Y будет меньше
высоты окна.
                   * Значит необходимо добавить делений, сделаем это так:
                   * 1) Вычислим, сколько делений влезет по Y при выбранном
масштабе - getSize().getHeight()/scale
                   * 2) Вычтем из этого сколько делений требовалось изначально
                   * 3) Набросим по половине недостающего расстояния на maxY и
minY
                   */
                  double yIncrement = (getSize().getHeight()/scale - (maxY -
minY))/2;
                  maxY += yIncrement;
                  minY -= yIncrement;
            }
            if (scale==scaleY) {
                  // Если за основу был взят масштаб по оси Y, действовать по
аналогии
                  double xIncrement = (getSize().getWidth()/scale - (maxX -
minX))/2;
                  maxX += xIncrement;
                  minX -= xIncrement;
            }
            // Шаг 7 - Сохранить текущие настройки холста
            Graphics2D canvas = (Graphics2D) g;
            Stroke oldStroke = canvas.getStroke();
            Color oldColor = canvas.getColor();
            Paint oldPaint = canvas.getPaint();
            Font oldFont = canvas.getFont();
            // Шаг 8 - В нужном порядке вызвать методы отображения элементов
графика
            // Порядок вызова методов имеет значение, т.к. предыдущий рисунок
будет затираться последующим
            // Первыми (если нужно) отрисовываются оси координат.
            if (showAxis) paintAxis(canvas);
            // Затем отображается сам график
            paintGraphics(canvas);
            // Затем (если нужно) отображаются маркеры точек, по которым
строился график.
            if (showMarkers) paintMarkers(canvas);
            // Шаг 9 - Восстановить старые настройки холста
            canvas.setFont(oldFont);
            canvas.setPaint(oldPaint);
            canvas.setColor(oldColor);
            canvas.setStroke(oldStroke);
      }

      // Отрисовка графика по прочитанным координатам
      protected void paintGraphics(Graphics2D canvas) {
            // Выбрать линию для рисования графика
            canvas.setStroke(graphicsStroke);
            // Выбрать цвет линии
            canvas.setColor(Color.RED);
            /* Будем рисовать линию графика как путь, состоящий из множества
сегментов (GeneralPath)
             * Начало пути устанавливается в первую точку графика, после чего
прямой соединяется со
             * следующими точками
             */
```

```
            GeneralPath graphics = new GeneralPath();
            for (int i=0; i<graphicsData.length; i++) {
                  // Преобразовать значения (x,y) в точку на экране point
                  Point2D.Double point = xyToPoint(graphicsData[i][0],
graphicsData[i][1]);
                  if (i>0) {
                        // Не первая итерация цикла - вести линию в точку
point
                        graphics.lineTo(point.getX(), point.getY());
                  } else {
                        // Первая итерация цикла - установить начало пути в
точку point
                        graphics.moveTo(point.getX(), point.getY());
                  }
            }
            // Отобразить график
            canvas.draw(graphics);
      }

      // Отображение маркеров точек, по которым рисовался график
      protected void paintMarkers(Graphics2D canvas) {
            // Шаг 1 - Установить специальное перо для черчения контуров
маркеров
            canvas.setStroke(markerStroke);
            // Выбрать красный цвета для контуров маркеров
            canvas.setColor(Color.RED);
            // Выбрать красный цвет для закрашивания маркеров внутри
            canvas.setPaint(Color.RED);
            // Шаг 2 - Организовать цикл по всем точкам графика
            for (Double[] point: graphicsData) {
                  // Инициализировать эллипс как объект для представления
маркера
                  Ellipse2D.Double marker = new Ellipse2D.Double();
                  /* Эллипс будет задаваться посредством указания координат
его центра
                   и угла прямоугольника, в который он вписан */
                  // Центр - в точке (x,y)
                  Point2D.Double center = xyToPoint(point[0], point[1]);
                  // Угол прямоугольника - отстоит на расстоянии (3,3)
                  Point2D.Double corner = shiftPoint(center, 3, 3);
                  // Задать эллипс по центру и диагонали
                  marker.setFrameFromCenter(center, corner);
                  canvas.draw(marker);    // Начертить контур маркера
                  canvas.fill(marker);    // Залить внутреннюю область маркера
            }
      }

      // Метод, обеспечивающий отображение осей координат
      protected void paintAxis(Graphics2D canvas) {
            // Установить особое начертание для осей
            canvas.setStroke(axisStroke);
            // Оси рисуются чёрным цветом
            canvas.setColor(Color.BLACK);
            // Стрелки заливаются чёрным цветом
            canvas.setPaint(Color.BLACK);
            // Подписи к координатным осям делаются специальным шрифтом
            canvas.setFont(axisFont);
            // Создать объект контекста отображения текста - для получения
характеристик устройства (экрана)
            FontRenderContext context = canvas.getFontRenderContext();
            // Определить, должна ли быть видна ось Y на графике
            if (minX<=0.0 && maxX>=0.0) {
                  // Она должна быть видна, если левая граница показываемой
области (minX) <= 0.0,
```

```
                    // а правая (maxX) >= 0.0
                    // Сама ось - это линия между точками (0, maxY) и (0, minY)
                    canvas.draw(new Line2D.Double(xyToPoint(0, maxY),
xyToPoint(0, minY)));
                    // Стрелка оси Y
                    GeneralPath arrow = new GeneralPath();
                    // Установить начальную точку ломаной точно на верхний конец
оси Y
                    Point2D.Double lineEnd = xyToPoint(0, maxY);
                    arrow.moveTo(lineEnd.getX(), lineEnd.getY());
                    // Вести левый "скат" стрелки в точку с относительными
координатами (5,20)
                    arrow.lineTo(arrow.getCurrentPoint().getX()+5,
arrow.getCurrentPoint().getY()+20);
                    // Вести нижнюю часть стрелки в точку с относительными
координатами (-10, 0)
                    arrow.lineTo(arrow.getCurrentPoint().getX()-10,
arrow.getCurrentPoint().getY());
                    // Замкнуть треугольник стрелки
                    arrow.closePath();
                    canvas.draw(arrow); // Нарисовать стрелку
                    canvas.fill(arrow);     // Закрасить стрелку
                    // Нарисовать подпись к оси Y
                    // Определить, сколько места понадобится для надписи "y"
                    Rectangle2D bounds = axisFont.getStringBounds("y", context);
                    Point2D.Double labelPos = xyToPoint(0, maxY);
                    // Вывести надпись в точке с вычисленными координатами
                    canvas.drawString("y", (float)labelPos.getX() + 10,
(float)(labelPos.getY() - bounds.getY()));
                }
                // Определить, должна ли быть видна ось X на графике
                if (minY<=0.0 && maxY>=0.0) {
                    // Она должна быть видна, если верхняя граница показываемой
области (maxX) >= 0.0,
                    // а нижняя (minY) <= 0.0
                    canvas.draw(new Line2D.Double(xyToPoint(minX, 0),
xyToPoint(maxX, 0)));
                    // Стрелка оси X
                    GeneralPath arrow = new GeneralPath();
                    // Установить начальную точку ломаной точно на правый конец
оси X
                    Point2D.Double lineEnd = xyToPoint(maxX, 0);
                    arrow.moveTo(lineEnd.getX(), lineEnd.getY());
                    // Вести верхний "скат" стрелки в точку с относительными
координатами (-20,-5)
                    arrow.lineTo(arrow.getCurrentPoint().getX()-20,
arrow.getCurrentPoint().getY()-5);
                    // Вести левую часть стрелки в точку с относительными
координатами (0, 10)
                    arrow.lineTo(arrow.getCurrentPoint().getX(),
arrow.getCurrentPoint().getY()+10);
                    // Замкнуть треугольник стрелки
                    arrow.closePath();
                    canvas.draw(arrow); // Нарисовать стрелку
                    canvas.fill(arrow);     // Закрасить стрелку
                    // Нарисовать подпись к оси X
                    // Определить, сколько места понадобится для надписи "x"
                    Rectangle2D bounds = axisFont.getStringBounds("x", context);
                    Point2D.Double labelPos = xyToPoint(maxX, 0);
                    // Вывести надпись в точке с вычисленными координатами

                    canvas.drawString("x", (float)(labelPos.getX() -
bounds.getWidth() - 10), (float)(labelPos.getY() + bounds.getY()));
```

```
            }
        }

        /* Метод-помощник, осуществляющий преобразование координат.
         * Оно необходимо, т.к. верхнему левому углу холста с координатами
         * (0.0, 0.0) соответствует точка графика с координатами (minX, maxY),
где
         * minX - это самое "левое" значение X, а
         * maxY - самое "верхнее" значение Y.
         */
        protected Point2D.Double xyToPoint(double x, double y) {
            // Вычисляем смещение X от самой левой точки (minX)
            double deltaX = x - minX;
            // Вычисляем смещение Y от точки верхней точки (maxY)
            double deltaY = maxY - y;
            return new Point2D.Double(deltaX*scale, deltaY*scale);
        }

        /* Метод-помощник, возвращающий экземпляр класса Point2D.Double
         * смещённый по отношению к исходному на deltaX, deltaY
         * К сожалению, стандартного метода, выполняющего такую задачу, нет.
         */
        protected Point2D.Double shiftPoint(Point2D.Double src, double deltaX,
double deltaY) {
            // Инициализировать новый экземпляр точки
            Point2D.Double dest = new Point2D.Double();
            // Задать её координаты как координаты существующей точки +
заданные смещения
            dest.setLocation(src.getX() + deltaX, src.getY() + deltaY);
            return dest;
        }
```

Класс главного окна приложения MainFrame

```
package bsu.rfe.java.teacher;

import java.awt.BorderLayout;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.io.DataInputStream;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.IOException;
import javax.swing.AbstractAction;
import javax.swing.Action;
import javax.swing.JCheckBoxMenuItem;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JOptionPane;
import javax.swing.event.MenuEvent;
import javax.swing.event.MenuListener;

@SuppressWarnings("serial")
public class MainFrame extends JFrame {

        // Начальные размеры окна приложения
        private static final int WIDTH = 800;
        private static final int HEIGHT = 600;

        // Объект диалогового окна для выбора файлов
```

```
	private JFileChooser fileChooser = null;

	// Пункты меню
	private JCheckBoxMenuItem showAxisMenuItem;
	private JCheckBoxMenuItem showMarkersMenuItem;

	// Компонент-отображатель графика
	private GraphicsDisplay display = new GraphicsDisplay();

	// Флаг, указывающий на загруженность данных графика
	private boolean fileLoaded = false;

	public MainFrame() {
		// Вызов конструктора предка Frame
		super("Построение графиков функций на основе заранее
подготовленных файлов");
		// Установка размеров окна
		setSize(WIDTH, HEIGHT);
		Toolkit kit = Toolkit.getDefaultToolkit();
		// Отцентрировать окно приложения на экране
		setLocation((kit.getScreenSize().width - WIDTH)/2,
(kit.getScreenSize().height - HEIGHT)/2);
		// Развёртывание окна на весь экран
		setExtendedState(MAXIMIZED_BOTH);

		// Создать и установить полосу меню
		JMenuBar menuBar = new JMenuBar();
		setJMenuBar(menuBar);
		// Добавить пункт меню "Файл"
		JMenu fileMenu = new JMenu("Файл");
		menuBar.add(fileMenu);
		// Создать действие по открытию файла
		Action openGraphicsAction = new AbstractAction("Открыть файл с
графиком") {
			public void actionPerformed(ActionEvent event) {
				if (fileChooser==null) {
					fileChooser = new JFileChooser();
					fileChooser.setCurrentDirectory(new File("."));
				}
				if (fileChooser.showOpenDialog(MainFrame.this) ==
JFileChooser.APPROVE_OPTION)
					openGraphics(fileChooser.getSelectedFile());
			}
		};
		//  Добавить соответствующий элемент меню
		fileMenu.add(openGraphicsAction);
		// Создать пункт меню "График"
		JMenu graphicsMenu = new JMenu("График");
		menuBar.add(graphicsMenu);
		// Создать действие для реакции на активацию элемента "Показывать
оси координат"
		Action showAxisAction = new AbstractAction("Показывать оси
координат") {
			public void actionPerformed(ActionEvent event) {
				// свойство showAxis класса GraphicsDisplay истина,
если элемент меню
				// showAxisMenuItem отмечен флажком, и ложь - в
противном случае
				display.setShowAxis(showAxisMenuItem.isSelected());
			}
		};
		showAxisMenuItem = new JCheckBoxMenuItem(showAxisAction);
		// Добавить соответствующий элемент в меню
		graphicsMenu.add(showAxisMenuItem);
```

```
            // Элемент по умолчанию включен (отмечен флажком)
            showAxisMenuItem.setSelected(true);
            // Повторить действия для элемента "Показывать маркеры точек"
            Action showMarkersAction = new AbstractAction("Показывать маркеры
точек") {
                  public void actionPerformed(ActionEvent event) {
                        // по аналогии с showAxisMenuItem

      display.setShowMarkers(showMarkersMenuItem.isSelected());
                  }
            };
            showMarkersMenuItem = new JCheckBoxMenuItem(showMarkersAction);
            graphicsMenu.add(showMarkersMenuItem);
            // Элемент по умолчанию включен (отмечен флажком)
            showMarkersMenuItem.setSelected(true);
            // Зарегистрировать обработчик событий, связанных с меню "График"
            graphicsMenu.addMenuListener(new GraphicsMenuListener());

            // Установить GraphicsDisplay в цент граничной компоновки
            getContentPane().add(display, BorderLayout.CENTER);
      }

      // Считывание данных графика из существующего файла
      protected void openGraphics(File selectedFile) {
            try {
                  // Шаг 1 - Открыть поток чтения данных, связанный с входным
файловым потоком
                  DataInputStream in = new DataInputStream(new
FileInputStream(selectedFile));
                  /* Шаг 2 - Зная объём данных в потоке ввода можно вычислить,
                   * сколько памяти нужно зарезервировать в массиве:
                   * Всего байт в потоке - in.available() байт;
                   * Размер одного числа Double - Double.SIZE бит, или
Double.SIZE/8 байт;
                   * Так как числа записываются парами, то число пар меньше в
2 раза
                   */
                  Double[][] graphicsData = new
Double[in.available()/(Double.SIZE/8)/2][];
                  // Шаг 3 - Цикл чтения данных (пока в потоке есть данные)
                  int i = 0;
                  while (in.available()>0) {
                        // Первой из потока читается координата точки X
                        Double x = in.readDouble();
                        // Затем - значение графика Y в точке X
                        Double y = in.readDouble();
                        // Прочитанная пара координат добавляется в массив
                        graphicsData[i++] = new Double[] {x, y};
                  }
                  // Шаг 4 - Проверка, имеется ли в списке в результате чтения
хотя бы одна пара координат
                  if (graphicsData!=null && graphicsData.length>0) {
                        // Да - установить флаг загруженности данных
                        fileLoaded = true;
                        // Вызывать метод отображения графика
                        display.showGraphics(graphicsData);
                  }
                  // Шаг 5 - Закрыть входной поток
                  in.close();
            } catch (FileNotFoundException ex) {
                  // В случае исключительной ситуации типа "Файл не найден"
показать сообщение об ошибке
                  JOptionPane.showMessageDialog(MainFrame.this, "Указанный
файл не найден", "Ошибка загрузки данных", JOptionPane.WARNING_MESSAGE);
```

```
                return;
            } catch (IOException ex) {
                // В случае ошибки ввода из файлового потока показать
сообщение об ошибке
                JOptionPane.showMessageDialog(MainFrame.this, "Ошибка чтения
координат точек из файла", "Ошибка загрузки данных",
JOptionPane.WARNING_MESSAGE);
                return;
            }
      }

      public static void main(String[] args) {
            // Создать и показать экземпляр главного окна приложения
            MainFrame frame = new MainFrame();
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
      }

      // Класс-слушатель событий, связанных с отображением меню
      private class GraphicsMenuListener implements MenuListener {

            // Обработчик, вызываемый перед показом меню
            public void menuSelected(MenuEvent e) {
                // Доступность или недоступность элементов меню "График"
определяется загруженностью данных
                showAxisMenuItem.setEnabled(fileLoaded);
                showMarkersMenuItem.setEnabled(fileLoaded);
            }

            // Обработчик, вызываемый после того, как меню исчезло с экрана
            public void menuDeselected(MenuEvent e) {
            }

            // Обработчик, вызываемый в случае отмены выбора пункта меню
(очень редкая ситуация)
            public void menuCanceled(MenuEvent e) {
            }
      }

}
```