ТЕМА 3
Расширение возможностей графического интерфейса пользователя

Цель лабораторной работы

Освоить основные принципы написания оконных приложений, обладающих меню, представляющих данные в табличном виде и осуществляющих взаимодействие с символьными и байтовыми потоками ввода-вывода на языке Java в среде Eclipse.

1 Особенности построения приложений с расширенным графическим интерфейсом

Большинство приложений с графическим интерфейсом обладают дополнительным средством взаимодействия с пользователем – полосу меню. Полоса меню вверху окна приложения содержит названия пунктов выпадающих меню. Щелчок на имени пункта открывает меню, содержащее элементы меню и подменю. Когда пользователь щёлкает на элементе меню, все меню закрываются и программе посылается сообщение. В используемой нами библиотеке Swing для построения меню используются шесть компонентов – `JMenuBar`, `JMenu`, `JPopupMenu`, `JMenuItem`, `JCheckBoxMenuItem`, `JRadioButtonMenuItem`.

Во многих приложениях возникает необходимость отображения табличных структур данных. Для этого в библиотеке Swing используется класс `JTable`, позволяющий не только отображать, но и изменять табличные данные.

Дополнительные средства по расширению возможностей графического интерфейса предлагают присутствующие в библиотеке Swing классы `JFileChooser` и `JColorChooser`, представляющие диалоговые окна выбора файла (рисунок 1.1) и выбора цвета из палитры (рисунок 1.2).

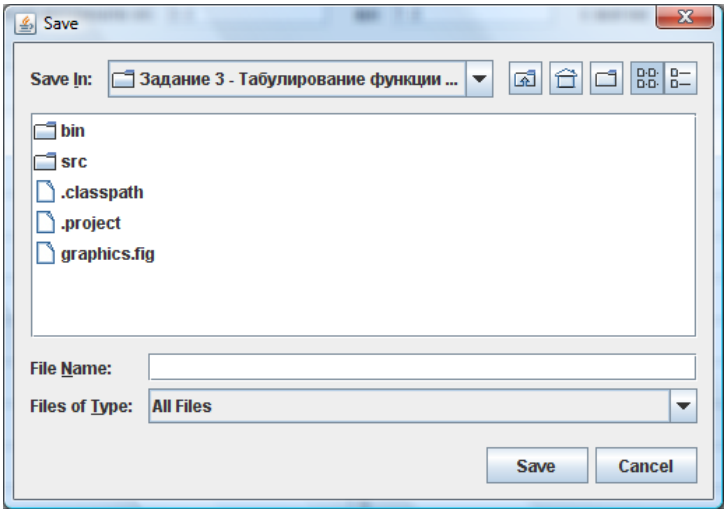


Рисунок 1.1 – Диалоговое окно выбора файла

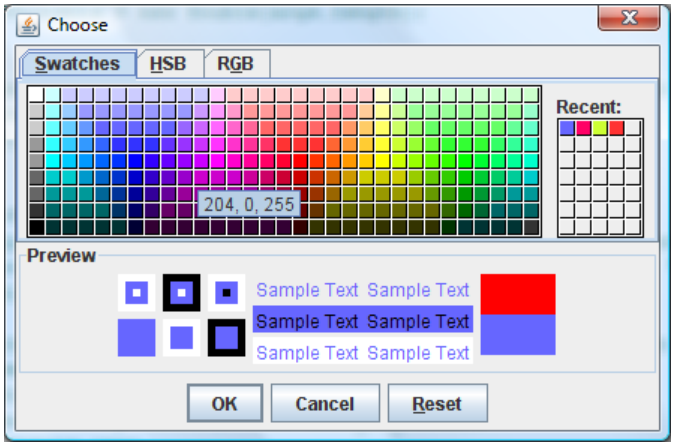


Рисунок 1.2 – Диалоговое окно выбора цвета

2 Краткая справка по необходимым программным компонентам

2.1 Представление меню приложения – класс JMenuBar

Чтобы создать меню приложения необходимо создать полосу меню. Полоса меню создается с помощью обращения к конструктору:

```
JMenuBar menuBar = new JMenuBar();
```

Полоса меню является таким же компонентом, как и другие объекты библиотеки Swing, и может быть размещена внутри любого компонента. В большинстве случаев мы будем размещать её в верхней части главного окна. Это делается обращением к методу `setJMenuBar()` класса `Frame`:

```
setJMenuBar(menuBar);
```

Независимое выполнение одного из блоков следующего фрагмента кода обеспечит размещение полосы меню в левой, в правой или в нижней частях главного окна приложения. При этом последовательное выполнение всех трёх блоков приведёт к отображению меню лишь в нижней части окна.

```
// Блок 1 – размещение меню в левой части окна
getContentPane().add(menuBar, BorderLayout.WEST);
// Блок 2 – размещение меню в правой части окна
getContentPane().add(menuBar, BorderLayout.EAST);
// Блок 3 – размещение меню в нижней части окна
getContentPane().add(menuBar, BorderLayout.SOUTH);
```

2.2 Представление пункта меню приложения – класс JMenu

Любое меню (верхнего уровня, пункты которого размещены вверху окна, или вложенного уровня) представляется экземпляром класса `JMenu`,

который создаётся обращением к конструктору `JMenu(menuName)`, принимающему в качестве аргумента имя создаваемого меню:

```
JMenu fileMenu = new JMenu("Файл");
```

Для добавления меню верхнего уровня в полосу меню, а также для добавления вложенных меню в меню-контейнеры применяется метод `add()`:

```
menuBar.add(fileMenu);
JMenu nestedMenu = new JMenu("Экспортировать данные");
fileMenu.add(nestedMenu);
```

2.3 Представление пункта подменю приложения – класс JMenuItem

Когда пользователь выбирает какой-либо элемент меню, приложению посылается сообщение типа `ActionEvent`, для получения которого необходимо зарегистрировать для соответствующего элемента слушателя сообщений данного типа. Существует два способа добавления элементов меню – через слушателей событий и классы действий.

2.3.1 Добавление пунктов меню через слушателей событий

Наиболее простым способом создания элемента меню является конструирование экземпляра `JMenuItem` с помощью метода `add()` родительского меню. Например, если у нас есть объект `fileMenu` типа `JMenu`, то элемент `openFileMenuItem` типа `JMenuItem` создаётся следующим образом:

```
JMenuItem openFileMenuItem = fileMenu.add("Открыть");
```

Используя возвращённую ссылку на экземпляр `JMenuItem` становится возможным прикрепить к нему слушателя событий типа `ActionEvent`. Удобным способом для этого является применение анонимных классов:

```
openFileMenuItem.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent ev) {
              // Код обработки события
        }
});
```

2.3.2 Добавление пунктов меню через классы действий

В ряде случаев команды меню могут быть активированы и через другие элементы интерфейса пользователя, например панели инструментов. В этом случае, реакцию на произошедшее событие целесообразно вынести в общий класс действия, экземпляр которого будет использоваться совместно целым рядом элементов интерфейса. И в этом случае удобным способом является

применение анонимных классов. На основе абстрактного класса `AbstractAction` создаётся экземпляр класса, реализующего реакцию на наступившее событие, после чего для него создаётся соответствующий пункт меню:

```
Action openFileAction = new AbstractAction("Открыть файл") {
        public void actionPerformed(ActionEvent ev) {
              // Обработка события
        }
};
JMenuItem openFileMenuItem = fileMenu.add(openFileAction);
```

Внимание! Запись `Action openFileAction = new AbstractAction() {…}` означает, что на основе базового абстрактного класса `AbstractAction` создаётся класс-потомок, реализующий унаследованный метод `actionPerformed(ActionEvent)`, и ссылка на экземпляр созданного безымянного (поэтому и название, анонимный) класса записывается в переменную `openFileAction`, которая может ссылаться на любой из объектов, реализующих интерфейс `Action`. Эта возможность обусловлена механизмом полиморфизма.

2.4 Диалоговое окно выбора файла – класс JFileChooser

Создание диалогового окна для выбора файлов является достаточно сложной задачей, что обусловлено необходимостью отображения содержимого файловой системы различными способами (с сортировкой по различным критериям и с различными способами фильтрации файлов). При этом следует иметь в виду, что различные платформы (Windows, Unix) имеют различные принципы организации файловых систем, что также необходимо учитывать. Библиотека Swing предлагает для этих целей уже готовый программный компонент – класс `JFileChooser`. Так как создание экземпляра данного класса является ресурсоёмкой операцией, занимающей достаточно времени, рекомендуется по возможности использовать созданные экземпляры повторно (сохранить ссылку на объект во внутреннем поле главного окна, и каждый раз отображать не новый экземпляр диалогового окна, а исходный).

```
JFileChooser fileChooser = new JFileChooser();
```

Далее необходимо установить текущую директорию, а также (при необходимости) файл, выделенный по умолчанию:

```
fileChooser.setCurrentDirectory(new File("."));
```

Диалоговое окно выбора файла может быть показано как при открытии файла, так и при его сохранении. В классе `JFileChooser` для этого предусмотрены два различных метода – `showOpenDialog()` (в этом случае кнопка будет иметь подпись «*Open*») и `showSaveDialog()` (кнопка будет иметь подпись «*Save*»). В качестве аргумента любому из этих методов передаётся указатель на компонент, являющийся «родителем» диалогового окна (в большинстве случаев им будет являться ссылка на экземпляр главного окна приложения):

```
int result = fileChooser.showSaveDialog(MainFrame.this);
```

Показ диалогового окна является модальной операций (т.е. блокирующей другие действия пользователя до того, пока он не закроет окно). Результат показа окна возвращается в целочисленную переменную: если пользователь закрыл окно успешно выбрав какой-либо файл, значение `result` будет равно `JFileChooser.APPROVE_OPTION`. Если пользователь закрыл диалоговое окно, нажав «*Cancel*», результатом будет `JFileChooser.CANCEL_OPTION`.

Узнать (в случае успешного закрытия окна), какой файл был выбран пользователем, можно обратившись к методу `getSelectedFile()`, возвращающему объект класса `File`.

```
File selectedFile = fileChooser.getSelectedFile();
```

2.5 Компонент представления таблиц – класс JTable

Компонент `JTable` отображает двумерную матрицу объектов. Он не хранит данные внутри себя, а получает их из *модели таблицы*. При создании экземпляра таблицы модель передаётся конструктору в качестве аргумента. Полученный в итоге компонент таблицы можно добавлять в окно приложения по аналогии с другими компонентами (кнопками, полями ввода и т.п.). Следует отметить, что таблица обладает достаточно широкими возможностями поведения – позволяет выделять ячейки и строки, масштабировать столбцы и изменять их порядок следования и т.п.

```
// data - модель таблицы, содержащая данные для отображения
// Создаётся экземпляр таблицы
JTable table = new JTable(data);
// Таблица "оборачивается" в панель, обладающую возможностями прокрутки
// и устанавливается в качестве содержания окна
getContentPane.add(new JScrollPane(table), BorderLayout.CENTER);
```

Данные для отображения извлекаются таблицей из модели, предоставляющей доступ к двумерному множеству объектов различных типов (подробнее модели таблиц описаны в п. 1.6). По умолчанию, любая

ячейка таблицы отображается следующим образом: над объектом, являющимся данными ячейки, выполняется операция `toString()`, результирующее значение показывается в ячейке таблицы. Это значит, что данными в модели могут являться любые (как стандартные, так и определённые пользователем) типы данных, при этом визуально в ячейках таблицы будет отображено строковое представление объектов этих типов. При необходимости задания собственного способа отображения данных модели (например, вместо показа кода цвета в шестнадцатеричной системе – заливка ячейки этим цветом) следует определить собственный класс-визуализатор ячеек таблицы (об этом далее в п. 1.7).

2.6 Представление данных в табличном виде – класс AbstractTableModel

Модель таблицы выступает по отношению к компоненту `JTable` источником данных. Так как класс `AbstractTableModel` является абстрактным, то создание его экземпляров невозможно, и он используется только в качестве базового класса для определения собственной модели данных.

Основные задачи модели таблицы: определение размерности таблицы (задание количества строк и столбцов модели); задание имён столбцов (для отображения заголовков столбцов); задание типов данных в столбцах (для использования совместно с визуализаторами ячеек); получение доступа к элементу таблицы (значение *i*-ой строки *j*-го столбца). Им соответствуют следующие методы, унаследованные от `AbstractTableModel` и переопределяемые в классе-потомке:

Таблица 2.1 – Основные методы модели данных таблицы

Название метода	Описание
`int getColumnCount()`	Возвращает число столбцов таблицы
`int getRowCount()`	Возвращает число строк таблицы
`String getColumnName(int col)`	Возвращает имя столбца с номером `col` (нумерация начинается с нуля)
`Class getColumnClass(int col)`	Возвращает тип данных столбца с номером `col` (нумерация начинается с нуля)
`Object getValueAt(int row, int col)`	Возвращает значение в ячейке, находящейся в строке с номером `row` и столбце с номером `col` (нумерация в обоих начинается с нуля)

2.7 Отображение ячеек таблиц – интерфейс TableCellRenderer

В библиотеке Swing имеются три встроенных визуализатора, автоматически выбираемых в зависимости от типа данных в столбцах таблицы:

Таблица 2.2 – Стандартные визуализаторы ячеек таблицы

Тип данных	Способ отображения
`ImageIcon`	Как картинка
`Boolean`	Как флажок
`Object`	Как строка

При переопределении способа отображения данных в ячейке таблицы, необходимо выполнить:

1. В модели таблицы определить тип данных в столбцах – для этого переопределить метод `getColumnClass()`. Например, следующая реализация метода сообщает, что в первом столбце модели хранятся данные типа `String`, во втором – данные типа `Double`, а в третьем – данные типа `Color`:

```
public Class<?> getColumnClass(int col) {
      switch (col) {
      case 1:
            return Double.class;
      case 2:
            return Color.class;
      default:
            return String.class;
      }
}
```

2. Создать класс, реализующий интерфейс визуализатора ячеек (`TableCellRenderer`), переопределив его метод `getTableCellRendererComponent()` и передавая ему в качестве аргументов следующие сведения:
   - ссылку на экземпляр таблицы, ячейка которой отображается;
   - значение в отображаемой ячейке;
   - флаг выделения ячейки;
   - флаг наличия фокуса ввода в ячейке;
   - позицию элемента таблицы (номер строки и столбца).

   Метод возвращает экземпляр (например, `renderer`) класса `Component`. Тогда для непосредственной прорисовки ячейки будет автоматически вызван метод `renderer.paint()`. Например, следующая реализация метода отображает панель, закрашенную

цветом, объектное представление которого находится в отображаемой ячейке таблицы:

```
public Component getTableCellRendererComponent(JTable table,
Object value, boolean isSelected, boolean hasFocus, int row, int col) {
      JPanel panel = new JPanel();
      panel.setBackground((Color) value);
      return panel;
}
```

3. Связать тип данных ячеек таблицы с соответствующим визуализатором. Например, в следующем фрагменте для данных типа `Color` задаётся специализированный визуализатор (самостоятельно реализованный пользователем):

```
table.setDefaultRenderer(Color.class, new ColoredCellRenderer());
```

3 Пример приложения

Задание: составить программу вычисления значений многочлена по схеме Горнера на отрезке с представлением результатов в табличной форме. Коэффициенты многочлена должны передаваться программе как аргументы командной строки, границы отрезка и шаг табулирования – задаваться с помощью графического интерфейса. Приложение должно выполнять: сохранение результатов вычислений в текстовый и двоичный файлы; поиск в таблице заданного значения многочлена. Вид главного окна приложения представлен на рисунке 3.1 (цветом выделена ячейка, содержащая искомое значение 0.576):

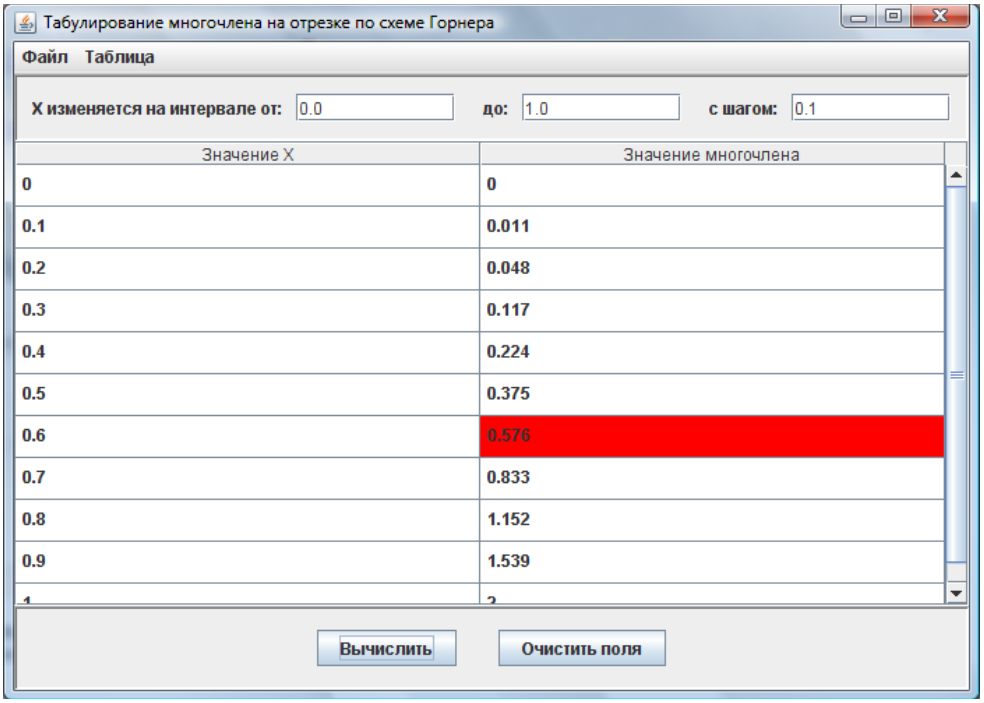


Рисунок 3.1 – Внешний вид главного окна приложения

3.1 Структура приложения и размещение элементов интерфейса

В структуре приложения можно выделить следующие блоки:

Модель таблицы, вычисляющую значения многочлена в точках отрезка и сохраняющую их в памяти для представления компоненту, отвечающему за визуализацию ячеек таблицы.

Визуализатор ячеек таблицы, обеспечивающий выделение ячеек, значения которых совпадают с искомым.

Главное окно приложения, организующее рабочее пространство окна, показ главного меню, реакцию на действия пользователя, запись данных в потоки вывода.

Диаграмма классов приложения, на которой видны выделенные блоки, приведена на рисунке 3.2:

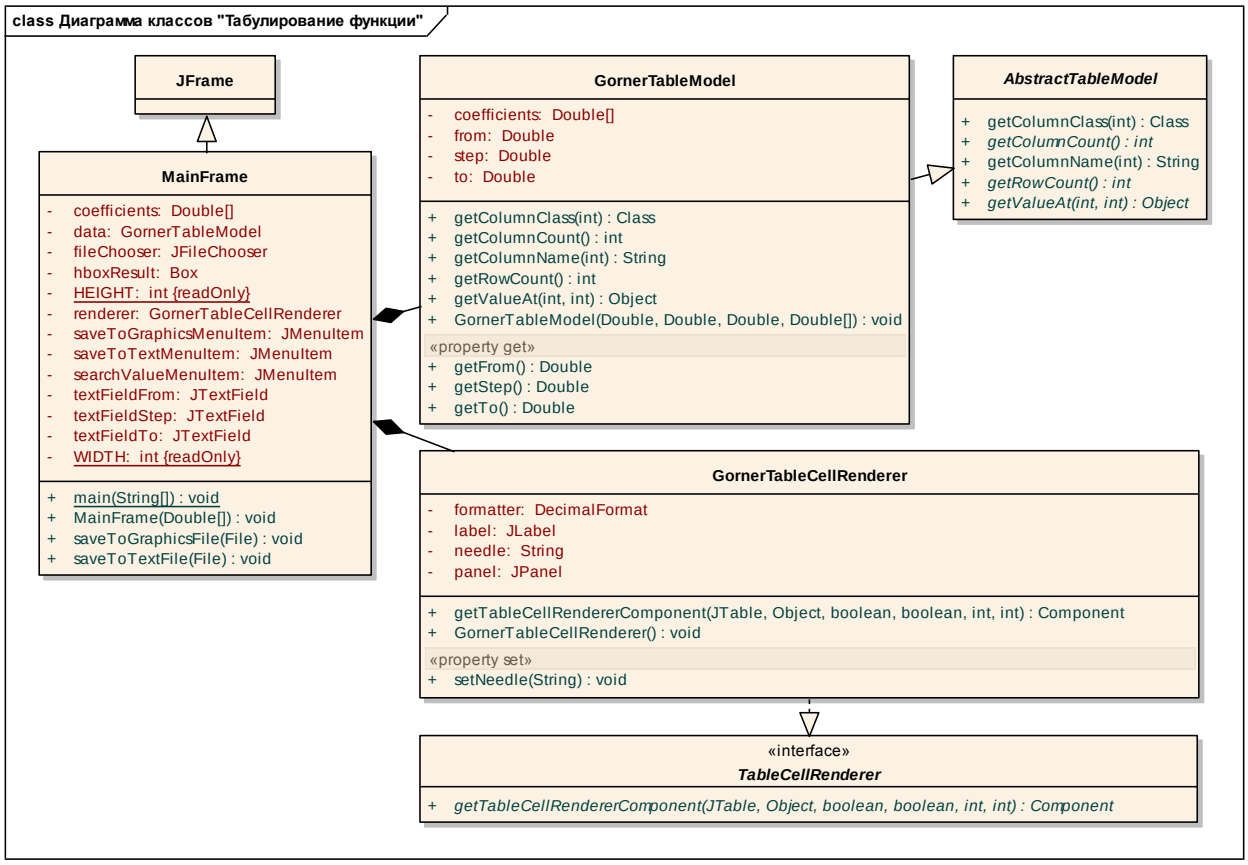


Рисунок 3.2 – Диаграмма классов приложения

Рассмотрим более подробно способ размещения элементов интерфейса в пространстве фрейма. Будем использовать для этого стандартный для фрейма менеджер «граничной» компоновки, совмещая его с контейнерами «коробка с горизонтальной укладкой». Граничная компоновка в данном случае предпочтительна по причине того, что после начального размещения элементов в пространстве фрейма, размеры всех краевых регионов (кроме центральной области) остаются неизменными. Общую компоновку

элементов интерфейса лучше анализировать на начальном состоянии окна, до вычисления значений многочлена в точках (рисунок 3.3):

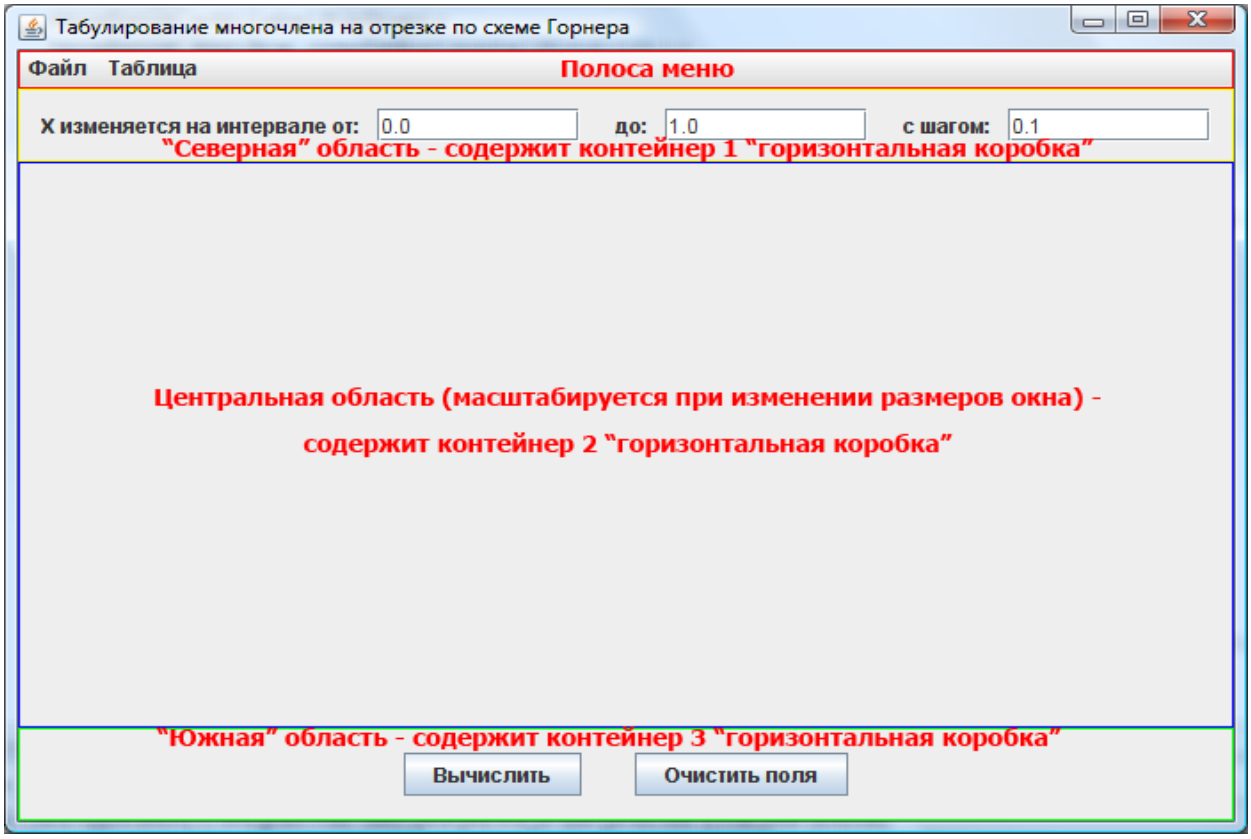
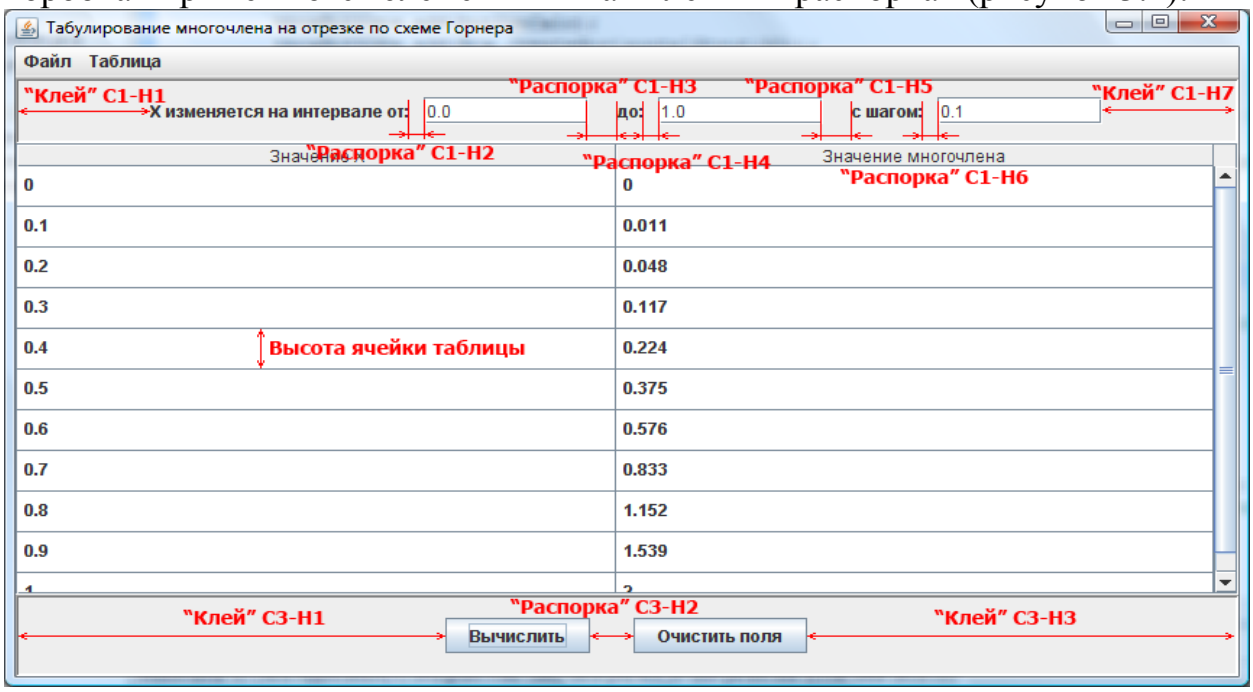


Рисунок 3.3 – Выделение контейнеров в главном окне приложения

На рисунке видно, что:

- полоса меню не является частью граничной компоновки, располагаясь в верхней части фрейма;
- в верхней области граничной компоновки находится контейнер 1 типа «горизонтальная коробка», в котором содержатся текстовые поля ввода для задания границ отрезка и шага табулирования;
- левая и правая области компоновки не используются и имеют нулевой размер;
- центральная область (масштабируемая при изменении размеров окна таким образом, чтобы занять всё свободное пространство) содержит контейнер 2 типа «горизонтальная коробка». Контейнер, в свою очередь, в исходном состоянии (до нажатия кнопки «*Вычислить*» или после нажатия кнопки «*Очистить поля*») содержит панель `JPanel` и визуально представляется пустым;
- нижняя область компоновки содержит контейнер 3 типа «горизонтальная коробка», в котором размещены кнопки «*Вычислить*» и «*Очистить поля*».

Для компоновки элементов внутри контейнеров типа «горизонтальная коробка» применяются элементы типа «клей» и «распорка» (рисунок 3.4):



0	0
0.1	0.011
0.2	0.048
0.3	0.117
0.4	0.224
0.5	0.375
0.6	0.576
0.7	0.833
0.8	1.152
0.9	1.539

Рисунок 3.4 – Компоновка элементов внутри горизонтальных контейнеров

3.2 Подготовительный этап

Этап предполагает создание каркаса приложения (см. шаги 3.1 – 3.4 лабораторной работы №1). Назовём главный класс приложения `MainFrame`. В роли класса-предка для него выступает класс `JFrame`.

3.3 Реализация модели таблицы

Для реализации модели таблицы необходимо добавить в пакет класс, являющийся потомком базового класса `AbstractTableModel`. Сгенерированный посредством мастера добавления нового класса код аналогичен представленному ниже:

```
package bsu.rfe.java.group7.lab3.Ivanov.varB4;

import javax.swing.table.AbstractTableModel;

public class GornerTableModel extends AbstractTableModel {

	@Override
	public int getColumnCount() {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public int getRowCount() {
		// TODO Auto-generated method stub
		return 0;
```

```
	}

	@Override
	public Object getValueAt(int arg0, int arg1) {
		// TODO Auto-generated method stub
		return null;
	}

}
```

С помощью аннотации `@Override` мастер отмечает, что данные методы являются переопределяемыми, а комментарии `// TODO Auto-generated …` сообщают, что данный метод был создан автоматически, и его содержание пользователю необходимо реализовать самостоятельно. После того, как методы будут реализованы, комментарии `// TODO` следует убрать.

Перечень автоматически сгенерированных методов нас не удовлетворяет по ряду причин: 1) необходимо предоставить сведения о названиях столбцов (метод `getColumnName()`); 2) необходимо предоставить сведения о типе данных в столбцах, в противном случае они будут отображены как строки (метод `getColumnClass()`). Поэтому с помощью пункта меню «*Source* → *Override/Implement Methods…*» добавим в код класса заглушки названных методов:

```
	@Override
	public Class<?> getColumnClass(int arg0) {
		// TODO Auto-generated method stub
		return super.getColumnClass(arg0);
	}

	@Override
	public String getColumnName(int arg0) {
		// TODO Auto-generated method stub
		return super.getColumnName(arg0);
	}
```

Для вычисления данных в модели таблицы необходимо наличие в ней массива коэффициентов многочлена, начальной и конечной точек отрезка табулирования, а также величины шага табулирования. Это приводит к необходимости добавления в модель:

- следующих полей класса:

```
		private Double[] coefficients;
		private Double from;
		private Double to;
		private Double step;
```

- конструктора для их начальной инициализации:

```
	public GornerTableModel(Double from, Double to, Double step, Double[]
coefficients) {
		this.from = from;
		this.to = to;
		this.step = step;
		this.coefficients = coefficients;
	}
```

- селекторов (геттеров) для чтения их текущих значений из модели (потребуется, в частности, при сохранении значений вычислений в файл):

```
public Double getFrom() {
	return from;
}

public Double getTo() {
	return to;
}

public Double getStep() {
	return step;
}
```

Напоминаем, что быстрым способом создания селекторов (при необходимости – модификаторов) является использование возможностей среды *Eclipse* по быстрой генерации кода, доступных через меню «*Source* → *Generate Getters and Setters*…».

С учётом всех названных добавок, исходный код класса модели таблицы примет вид:

```
package bsu.rfe.java.group7.lab3.Ivanov.varB4;

import javax.swing.table.AbstractTableModel;

public class GornerTableModel extends AbstractTableModel {

	private Double[] coefficients;
	private Double from;
	private Double to;
	private Double step;

	public GornerTableModel(Double from, Double to, Double step, Double[]
coefficients) {
		this.from = from;
		this.to = to;
		this.step = step;
		this.coefficients = coefficients;
	}

	public Double getFrom() {
		return from;
	}

	public Double getTo() {
		return to;
	}

	public Double getStep() {
		return step;
	}
```

```
	@Override
	public int getColumnCount() {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public int getRowCount() {
		// TODO Auto-generated method stub
		return 0;
	}

	@Override
	public Object getValueAt(int arg0, int arg1) {
		// TODO Auto-generated method stub
		return null;
	}

	@Override
	public Class<?> getColumnClass(int arg0) {
		// TODO Auto-generated method stub
		return super.getColumnClass(arg0);
	}

	@Override
	public String getColumnName(int arg0) {
		// TODO Auto-generated method stub
		return super.getColumnName(arg0);
	}
}
```

Заключительным шагом в создании модели таблицы является реализация тела пяти методов, помеченных комментариями `// TODO`.

Тело метода `getColumnCount()` является наиболее простым, так как число столбцов в нашей модели неизменно и равняется двум:

```
	public int getColumnCount() {
		return 2;
	}
```

Количество строк в таблице зависит от длины интервала табулирования и размера шага, поэтому его необходимо вычислять:

```
	public int getRowCount() {
		return new Double(Math.ceil((to-from)/step)).intValue()+1;
	}
```

Так как в нашей модели первый столбец содержит значение *X* в точке, где вычисляется значение многочлена, а второй столбец – непосредственно вычисленное значение, метод `getColumnName()` реализуем следующим образом:

```
	public String getColumnName(int col) {
```

```
        switch (col) {
        case 0:
              return "Значение X";
        default:
              return "Значение многочлена";
        }
  }
```

Тип данных для обоих столбцов в нашем случае одинаков, им является число с плавающей точкой – `Double`. Исходный код метода `getColumnClass()` представлен ниже:

```
public Class<?> getColumnClass(int col) {
        return Double.class;

}
```

По сравнению с описанными выше методами, более сложным является реализация метода `getValueAt()`, возвращающего содержимое ячеек таблицы. Если необходимо получить значение первого столбца *i*-ой строки, то следует вернуть значение *X* в точке. Если необходимо получить значение второго столбца *i*-ой строки, то следует вернуть значение многочлена в точке *X*.

```
  public Object getValueAt(int row, int col) {
        // Вычислить значение X как НАЧАЛО_ОТРЕЗКА + ШАГ*НОМЕР_СТРОКИ
        double x = from + step*row;
        if (col==0) {
              return x;
        } else {
              Double result;
              // Вычисление значения в точке по схеме Горнера.
              // Вспомнить 1-ый курс и реализовать
              // ...
              return result;
        }
  }
```

3.4 Реализация визуализатора ячеек таблицы

Как отмечалось в пункте 2.7, визуализатор должен реализовывать интерфейс `TableCellRenderer`, и, в частности, метод `getTableCellRendererComponent()`. Сгенерированный каркас класса визуализатора представлен ниже:

```
package bsu.rfe.java.group7.lab3.Ivanov.varB4;

import java.awt.Component;
import javax.swing.JTable;
import javax.swing.table.TableCellRenderer;

public class GornerTableCellRenderer implements TableCellRenderer {
```

```
	@Override
	public Component getTableCellRendererComponent(JTable arg0, Object arg1,
			boolean arg2, boolean arg3, int arg4, int arg5) {
		// TODO Auto-generated method stub
		return null;
	}

}
```

Так как способ отображения ячейки таблицы зависит от совпадения её значения с искомым, необходимо добавить в класс визуализатора поле данных, хранящее искомое значение и модификатор для его задания:

```
	// Ищем ячейки, строковое представление которых равно needle (иголке)
	// Применяется аналогия поиска иголки в сене, в роли сена - таблица
	private String needle = null;

	public void setNeedle(String needle) {
		this.needle = needle;
	}
```

«Рабочей лошадкой» визуализатора являются компоненты панель (`JPanel`) и надпись (`JLabel`). С этими компонентами мы знакомились в лабораторной работе №2. Первая версия метода `getTableCellRendererComponent()` представлена ниже:

```
    public Component getTableCellRendererComponent(JTable table, Object
value, boolean isSelected, boolean hasFocus, int row, int col) {
		// Создать компонент панели
		JPanel panel = new JPanel();
		// Создать компонент надписи
		JLabel label = new JLabel(value.toString());
		// Разместить надпись внутри панели
		panel.add(label);
		// Установить выравнивание надписи по левому краю панели
		panel.setLayout(new FlowLayout(FlowLayout.LEFT));
		if (col==1 && needle!=null && needle.equals(value)) {
			// Номер столбца = 1 (т.е. второй столбец)
			// + иголка не null (т.е. мы что-то ищем)
			// + значение иголки совпадает со значением ячейки таблицы -
			// окрасить задний фон панели в красный цвет
			panel.setBackground(Color.RED);
		} else {
			// Иначе - в обычный белый
			panel.setBackground(Color.WHITE);
		}
		return panel;

	}
```

Анализ приведенного фрагмента кода выявляет ряд его недостатков.

Во-первых, для отображения каждой ячейки таблицы создаются отдельные экземпляры компонентов панели и надписи, которые после показа более не используются и утилизируются сборщиком мусора. Это приводит к

избыточному расходу памяти и замедлению работы приложения. Если создать внутри класса визуализатора внутренние поля данных для компонентов панели и надписи, инициализировать их в конструкторе и повторно использовать при показе – расход памяти станет существенно ниже:

```
public class GornerTableCellRenderer implements TableCellRenderer {

        ...
        private JPanel panel = new JPanel();
        private JLabel label = new JLabel();

        public GornerTableCellRenderer() {
              // Разместить надпись внутри панели
              panel.add(label);
              // Установить выравнивание надписи по левому краю панели
              panel.setLayout(new FlowLayout(FlowLayout.LEFT));
        }
}
```

Во-вторых, в данном фрагменте мы полагались на стандартный способ преобразования числа типа `Double` в строку. Это приводит к тому, что после запятой показывается большое количество цифр, а в ряде случаев – и к видимым ошибкам округления (рисунок 3.5).

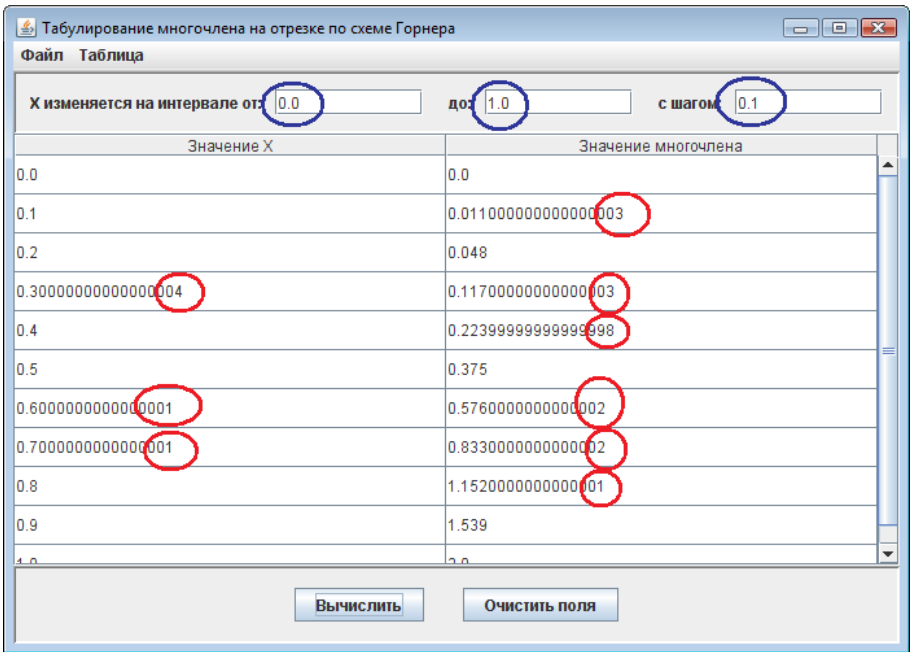


Рисунок 3.5 – Особенности, обусловленные стандартным преобразованием `Double` в `String`

Как видно на рисунке, начало и конец отрезка, а также шаг табулирования являются числами с плавающей точкой, дробная часть которых ограничена первым разрядом после запятой. В то же время

достаточно часто при вычислениях появляются цифры в шестнадцатом разряде после запятой. Для исправления этого воспользуемся предоставленным платформой *Java* классом `DecimalFormat` для форматирования строкового представления чисел (сделав его внутренним полем данных класса):

```
public class GornerTableCellRenderer implements TableCellRenderer {
 ...
 private DecimalFormat formatter = (DecimalFormat)NumberFormat.getInstance();
 ...
}
```

Перед использованием его необходимо настроить под наши требования, что удобно сделать в конструкторе класса-визуализатора:

```
public GornerTableCellRenderer() {
      ...
      // Показывать только 5 знаков после запятой
      formatter.setMaximumFractionDigits(5);
      // Не использовать группировку (не отделять тысячи)
      // Т.е. показывать число как "1000", а не "1 000" или "1,000"
      formatter.setGroupingUsed(false);
      // Установить в качестве разделителя дробной части точку, а не запятую
      // По умолчанию, в региональных настройках Россия/Беларусь дробная часть
      // отделяется запятой
      DecimalFormatSymbols dottedDouble = formatter.getDecimalFormatSymbols();
      dottedDouble.setDecimalSeparator('.');
      formatter.setDecimalFormatSymbols(dottedDouble);
}
```

Изменённая с учётом сделанных замечаний окончательная версия метода `getTableCellRendererComponent()` представлена ниже:

```
      public Component getTableCellRendererComponent(JTable table, Object
value, boolean isSelected, boolean hasFocus, int row, int col) {
            // Преобразовать число в строку с помощью форматировщика
            String formattedDouble = formatter.format(value);
            // Установить текст надписи равным строковому представлению числа
            label.setText(formattedDouble);
            if (col==1 && needle!=null && needle.equals(formattedDouble)) {
                  // Номер столбца = 1 (т.е. второй столбец)
                  // + иголка не null (т.е. мы что-то ищем)
                  // + значение иголки совпадает со значением ячейки таблицы -
                  // окрасить задний фон панели в красный цвет
                  panel.setBackground(Color.RED);
            } else {
                  // Иначе - в обычный белый
                  panel.setBackground(Color.WHITE);
            }
            return panel;
      }
```

3.5 Реализация главного окна приложения

Реализация главного окна приложения состоит из нескольких этапов:

1. Определение необходимых внутренних полей и их добавление.
2. Реализация конструктора окна, обеспечивающего компоновку элементов пользовательского интерфейса и задание реакций на действия пользователя.
3. Реализация методов сохранения данных в потоки вывода.
4. Реализация главного метода `main()`.

3.5.1 Определение внутренних полей

Как мы помним, внутренними полями класса целесообразно делать:

- Константы (определяя их с модификаторами static final);
- Объекты, совместно используемые различными методами;
- Объекты, которые используются только одним методом, но создание которых нецелесообразно при каждом обращении к методу.

Тогда анализируя сценарии использования объектов разрабатываемого приложения, имеем:

Таблица 3.1 – Классификация объектов приложения

Имя объекта	Сценарий использования
Константы	
`WIDTH`	Исходный размер окна по горизонтали
`HEIGHT`	Исходный размер окна по вертикали
Совместно используемые объекты	
`Double[] coefficients`	Массив коэффициентов многочлена, инициализируется в конструкторе, используется в обработчиках событий
`JMenuItem saveToTextMenuItem`	Элементы меню, создаются в конструкторе, совместно используются в обработчиках событий
`JMenuItem saveToGraphicsMenuItem`	
`JMenuItem searchValueMenuItem`	
`JTextField textFieldFrom`	Текстовые поля ввода, создаются в конструкторе, используются в обработчике событий
`JTextField textFieldTo`	
`JTextField textFieldStep`	
`Box hBoxResult`	Контейнер для результатов, создаётся в конструкторе, используется в обработчиках событий
`GornerTableModel`	Модель данных для таблицы, совместно используется в обработчиках событий
Повторно используемые объекты	
`JFileChooser`	Компонент диалогового окна выбора файла

GonerTableCellRenderer	Визуализатор ячеек таблицы

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

```
// Константы с исходным размером окна приложения
private static final int WIDTH = 700;
private static final int HEIGHT = 500;
// Массив коэффициентов многочлена
private Double[] coefficients;
// Объект диалогового окна для выбора файлов
// Компонент не создаётся изначально, т.к. может и не понадобиться
// пользователю если тот не собирается сохранять данные в файл
private JFileChooser fileChooser = null;
// Элементы меню
private JMenuItem saveToTextMenuItem;
private JMenuItem saveToGraphicsMenuItem;
private JMenuItem searchValueMenuItem;
// Поля ввода для считывания значений переменных
private JTextField textFieldFrom;
private JTextField textFieldTo;
private JTextField textFieldStep;
private Box hBoxResult;
// Визуализатор ячеек таблицы
private GornerTableCellRenderer renderer = new
      GornerTableCellRenderer();
// Модель данных с результатами вычислений
private GornerTableModel data;
```

3.5.2 Реализация конструктора окна

Конструктор окна обеспечивает компоновку интерфейса пользователя, а также задание реакций на действия пользователя. Входным аргументом для него является массив коэффициентов многочлена, передаваемый из главного метода `main()`. Первым этапом является вызов конструктора предка (`Frame`), сохранение массива коэффициентов во внутреннем поле данных, масштабирование и позиционирование окна:

```
// Обязательный вызов конструктора предка
super("Табулирование многочлена на отрезке по схеме Горнера");
// Запомнить во внутреннем поле переданные коэффициенты
this.coefficients = coefficients;
// Установить размеры окна
setSize(WIDTH, HEIGHT);
Toolkit kit = Toolkit.getDefaultToolkit();
// Отцентрировать окно приложения на экране
setLocation((kit.getScreenSize().width - WIDTH)/2,
      (kit.getScreenSize().height - HEIGHT)/2);
```

Следующим этапом является конструирование главного меню, включающее создание полосы меню и создание пунктов главного меню:

```
// Создать меню
JMenuBar menuBar = new JMenuBar();
```

```
        // Установить меню в качестве главного меню приложения
        setJMenuBar(menuBar);
        // Добавить в меню пункт меню "Файл"
        JMenu fileMenu = new JMenu("Файл");
        // Добавить его в главное меню
        menuBar.add(fileMenu);
        // Создать пункт меню "Таблица"
        JMenu tableMenu = new JMenu("Таблица");
        // Добавить его в главное меню
        menuBar.add(tableMenu);
```

После добавления пунктов главного меню становится возможным создать элементы этих пунктов и задать реакцию на их активацию пользователем. Для пункта меню «*Сохранить в текстовый файл*»:

```
// Создать новое "действие" по сохранению в текстовый файл
Action saveToTextAction = new AbstractAction("Сохранить в текстовый файл") {
      public void actionPerformed(ActionEvent event) {
            if (fileChooser==null) {
                  // Если диалоговое окно "Открыть файл" ещё не создано,
                  // то создать его
                  fileChooser = new JFileChooser();
                  // и инициализировать текущей директорией
                  fileChooser.setCurrentDirectory(new File("."));
            }
            // Показать диалоговое окно
            if (fileChooser.showSaveDialog(MainFrame.this) ==
                  JFileChooser.APPROVE_OPTION)
                  // Если результат его показа успешный, сохранить данные в
                  // текстовый файл
                  saveToTextFile(fileChooser.getSelectedFile());
      }
};
// Добавить соответствующий пункт подменю в меню "Файл"
saveToTextMenuItem = fileMenu.add(saveToTextAction);
// По умолчанию пункт меню является недоступным (данных ещё нет)
saveToTextMenuItem.setEnabled(false);
```

Как видно из приведенного фрагмента, непосредственно в теле обработчика логика обработки его выбора не реализуется. Вместо этого происходит обращение к методу `saveToTextFile()`, принимающему в качестве аргумента объект типа `File`.

По аналогии осуществляется добавление элемента меню «Сохранить данные для построения графика», с тем лишь отличием, что вместо метода `saveToTextFile()` вызывается метод `saveToGraphicsFile()`:

```
// Создать новое "действие" по сохранению в текстовый файл
Action saveToGraphicsAction = new AbstractAction("Сохранить данные для
построения графика") {
      public void actionPerformed(ActionEvent event) {
            if (fileChooser==null) {
                  // Если диалоговое окно "Открыть файл" ещё не создано,
                  // то создать его
                  fileChooser = new JFileChooser();
                  // и инициализировать текущей директорией
                  fileChooser.setCurrentDirectory(new File("."));
```

```
            }
            // Показать диалоговое окно
            if (fileChooser.showSaveDialog(MainFrame.this) ==
                  JFileChooser.APPROVE_OPTION);
                  // Если результат показа успешный,
                  // сохранить данные в двоичный файл
                  saveToGraphicsFile(fileChooser.getSelectedFile());
      }
};
// Добавить соответствующий пункт подменю в меню "Файл"
saveToGraphicsMenuItem = fileMenu.add(saveToGraphicsAction);
// По умолчанию пункт меню является недоступным(данных ещё нет)
saveToGraphicsMenuItem.setEnabled(false);
```

Реакция на выбор пользователем элемента меню «*Найти значение многочлена*» сводится к запросу у пользователя искомой строки, установка введенного значения в качестве искомой *иголки* визуализатора и запросу обновления области содержания главного окна:

```
// Создать новое действие по поиску значений многочлена
Action searchValueAction = new AbstractAction("Найти значение многочлена") {
      public void actionPerformed(ActionEvent event) {
            // Запросить пользователя ввести искомую строку
            String value = JOptionPane.showInputDialog(MainFrame.this,
                  "Введите значение для поиска", "Поиск значения",
                  JOptionPane.QUESTION_MESSAGE);
            // Установить введенное значение в качестве иголки в визуализаторе
            renderer.setNeedle(value);
            // Обновить таблицу
            getContentPane().repaint();
      }
};
// Добавить действие в меню "Таблица"
searchValueMenuItem = tableMenu.add(searchValueAction);
// По умолчанию пункт меню является недоступным (данных ещё нет)
searchValueMenuItem.setEnabled(false);
```

Следующим этапом является непосредственное создание графического интерфейса.

Создание верхней панели (контейнера 1, см. рисунок 3.3) включает инициализацию элементов интерфейса и их добавление в контейнер. Так как последовательность этих действий аналогична описанной в лабораторной работе №2, то этот фрагмент программы будет пропущен (полный код приложения приведен в Приложении 1), а мы остановимся подробнее на обработчиках событий нажатия на кнопки «*Вычислить*» и «*Очистить поля*»:

```
// Создать кнопку "Вычислить"
JButton buttonCalc = new JButton("Вычислить");
// Задать действие на нажатие "Вычислить" и привязать к кнопке
buttonCalc.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
            try {
                  // Считать значения границ отрезка, шага из полей ввода
                  Double from = Double.parseDouble(textFieldFrom.getText());
```

```
                    Double to = Double.parseDouble(textFieldTo.getText());
                    Double step = Double.parseDouble(textFieldStep.getText());
                    // На основе считанных данных создать модель таблицы
                    data = new GornerTableModel(from, to, step,
                                MainFrame.this.coefficients);
                    // Создать новый экземпляр таблицы
                    JTable table = new JTable(data);
                    // Установить в качестве визуализатора ячеек для класса
                    // Double разработанный визуализатор
                    table.setDefaultRenderer(Double.class, renderer);
                    // Установить размер строки таблицы в 30 пикселов (рис. 3.4)
                    table.setRowHeight(30);
                    // Удалить все вложенные элементы из контейнера hBoxResult
                    hBoxResult.removeAll();
                    // Добавить в hBoxResult таблицу, "обёрнутую" в панель с
                    // полосами прокрутки
                    hBoxResult.add(new JScrollPane(table));
                    // Обновить область содержания главного окна
                    getContentPane().validate();
                    // Пометить ряд элементов меню как доступных
                    saveToTextMenuItem.setEnabled(true);
                    saveToGraphicsMenuItem.setEnabled(true);
                    searchValueMenuItem.setEnabled(true);
              } catch (NumberFormatException ex) {
              // В случае ошибки преобразования показать сообщение об ошибке
                    JOptionPane.showMessageDialog(MainFrame.this,
                          "Ошибка в формате записи числа с плавающей точкой",
                          "Ошибочный формат числа", OptionPane.WARNING_MESSAGE);
              }
        }
});
// Создать кнопку "Очистить поля"
JButton buttonReset = new JButton("Очистить поля");
// Задать действие на нажатие "Очистить поля" и привязать к кнопке
buttonReset.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent ev) {
            // Установить в полях ввода значения по умолчанию
            textFieldFrom.setText("0.0");
            textFieldTo.setText("1.0");
            textFieldStep.setText("0.1");
            // Удалить все вложенные элементы контейнера hBoxResult
            hBoxResult.removeAll();
            // Добавить в контейнер пустую панель
            hBoxResult.add(new JPanel());
            // Пометить элементы меню как недоступные
            saveToTextMenuItem.setEnabled(false);
            saveToGraphicsMenuItem.setEnabled(false);
            searchValueMenuItem.setEnabled(false);
            // Обновить область содержания главного окна
            getContentPane().validate();
      }
});
```

При щелчке на кнопку «*Вычислить*» происходит чтение значений из полей ввода с преобразованием из строкового представления в тип чисел с плавающей точкой:

```
      Double from = Double.parseDouble(textFieldFrom.getText());
      Double to = Double.parseDouble(textFieldTo.getText());
      Double step = Double.parseDouble(textFieldStep.getText());
```

На основе полученных значений строится модель данных таблицы, используемая при создании компонента графической таблицы:

```
data = new GornerTableModel(from,to, step, MainFrame.this.coefficients);
JTable table = new JTable(data);
```

Затем для таблицы устанавливается разработанный специализированный визуализатор (для визуализации ячеек, имеющих тип `Double`), и устанавливается высота строк таблицы:

```
table.setDefaultRenderer(Double.class, renderer);
table.setRowHeight(30);
```

Созданный компонент таблицы помещается в контейнер для результатов, расположенный в центральной области главного окна:

```
hBoxResult.removeAll();
hBoxResult.add(new JScrollPane(table));
getContentPane().validate();
```

После чего элементы меню, связанные с обработкой данных, делаются доступными (так как в результате вычислений данные появились):

```
saveToTextMenuItem.setEnabled(true);
saveToGraphicsMenuItem.setEnabled(true);
searchValueMenuItem.setEnabled(true);
```

Обработчик события нажатия на кнопку «Очистить поля» восстанавливает исходные значения полей ввода и заменяет в контейнере содержания компонент графической таблицы на пустую панель:

```
textFieldFrom.setText("0.0");
textFieldTo.setText("1.0");
textFieldStep.setText("0.1");
hBoxResult.removeAll();
hBoxResult.add(new JPanel());
saveToTextMenuItem.setEnabled(false);
saveToGraphicsMenuItem.setEnabled(false);
searchValueMenuItem.setEnabled(false);
getContentPane().validate();
```

Заключительным этапом сборки компоновки интерфейса главного окна является добавление кнопок в нижнюю область окна и установка пустой панели в контейнер для результатов:

```
// Поместить созданные кнопки в контейнер
Box hboxButtons = Box.createHorizontalBox();
hboxButtons.setBorder(BorderFactory.createBevelBorder(1));
```

```
        hboxButtons.add(Box.createHorizontalGlue());
        hboxButtons.add(buttonCalc);
        hboxButtons.add(Box.createHorizontalStrut(30));
        hboxButtons.add(buttonReset);
        hboxButtons.add(Box.createHorizontalGlue());
        // Установить предпочтительный размер области равным удвоенному
        // минимальному, чтобы при компоновке окна область совсем не сдавили
        hboxButtons.setPreferredSize(new Dimension(new
          Double(hboxButtons.getMaximumSize().getWidth()).intValue(), new
          Double(hboxButtons.getMinimumSize().getHeight()).intValue()*2));
        // Разместить контейнер с кнопками в нижней (южной) области компоновки
        getContentPane().add(hboxButtons, BorderLayout.SOUTH);
        // Область для вывода результата пока что пустая
        hBoxResult = Box.createHorizontalBox();
        hBoxResult.add(new JPanel());
        // Установить контейнер hBoxResult в главной области компоновки
        getContentPane().add(hBoxResult, BorderLayout.CENTER);
```

3.5.3 Реализация методов сохранения данных

Методы сохранения данных в данном случае представлены методами записи результатов в текстовый файл (в виде, понятном для человека) и в бинарный файл (нечитабельном для человека, но удобном для приложений).

В случае сохранения результатов вычисления в текстовый файл будем предварять их кратким заголовком, облегчающим восприятие данных – заголовок будет содержать внешний вид многочлена, границы интервала и шаг табулирования. Изначально, файл, переданный в качестве входного аргумента, используется для конструирования символьного потока вывода. В конце поток вывода необходимо закрыть. Остальная часть метода является достаточно простой и прямолинейной:

```
protected void saveToTextFile(File selectedFile) {
   try {
      // Создать новый символьный поток вывода, направленный в указанный файл
      PrintStream out = new PrintStream(selectedFile);
      // Записать в поток вывода заголовочные сведения
      out.println("Результаты табулирования многочлена по схеме Горнера");
      out.print("Многочлен: ");
      for (int i=0; i<coefficients.length; i++) {
            out.print(coefficients[i] + "*X^" + (coefficients.length-i-1));
            if (i!=coefficients.length-1)
                  out.print(" + ");
      }
      out.println("");
      out.println("Интервал от " + data.getFrom() + " до " +
            data.getTo() + " с шагом " + data.getStep());
      out.println("================================================");
      // Записать в поток вывода значения в точках
      for (int i = 0; i<data.getRowCount(); i++) {
            out.println("Значение в точке " + data.getValueAt(i,0) +
                  " равно " + data.getValueAt(i,1));
      }
      // Закрыть поток
      out.close();
   } catch (FileNotFoundException e) {
      // Исключительную ситуацию "ФайлНеНайден" в данном случае можно
```

```
        // не обрабатывать, так как мы файл создаём, а не открываем для чтения
    }
}
```

Потенциально любые операции, связанные с вводом-выводом могут стать причиной исключительной ситуации, поэтому приведенный выше фрагмент кода помещён в блок `try-catch`. Но так как мы производим запись в файл, а не чтение из него, то исключительную ситуацию типа `FileNotFoundException` можно не обрабатывать.

Сохранение результатов вычислений в двоичный файл является ещё более простой операцией, так как нет необходимости сохранения дополнительного заголовка:

```
protected void saveToGraphicsFile(File selectedFile) {
      try {
            // Создать байтовый поток вывода, направленный в указанный файл
            DataOutputStream out = new DataOutputStream(new
                  FileOutputStream(selectedFile));
            for (int i = 0; i<data.getRowCount(); i++) {
                  // Записать в поток вывода значение X в точке
                  out.writeDouble((Double)data.getValueAt(i,0));
                  // значение многочлена в точке
                  out.writeDouble((Double)data.getValueAt(i,0));
            }
            // Закрыть поток вывода
            out.close();
      } catch (Exception e) {
            // Исключительную ситуацию "ФайлНеНайден" в данном случае можно
            // не обрабатывать, так как мы файл создаём, а не открываем

      }
}
```

3.5.4 Реализация метода main()

Задачей главного метода `main()` приложения является обработка входных параметров программы – аргументов командной строки. Полагая, что в качестве аргументов переданы коэффициенты многочлена (начиная со старшей его степени) метод осуществляет преобразование строкового представления в тип `Double`, создаёт экземпляр главного окна приложения и показывает его:

```
public static void main(String[] args) {
      // Если не задано ни одного аргумента командной строки -
      // Продолжать вычисления невозможно, коэффиценты неизвестны
      if (args.length==0) {
            System.out.println("Невозможно табулировать многочлен, для
которого не задано ни одного коэффициента!");
            System.exit(-1);
      }
      // Зарезервировать места в массиве коэффициентов столько,
      // сколько аргументов командной строки
      Double[] coefficients = new Double[args.length];
      int i = 0;
```

```
        try {
              // Перебрать все аргументы, пытаясь преобразовать их в Double
              for (String arg: args) {
                    coefficients[i++] = Double.parseDouble(arg);
              }
        }
        catch (NumberFormatException ex) {
           // Если преобразование невозможно - сообщить об ошибке и завершиться
           System.out.println("Ошибка преобразования строки '" + args[i] +
              "' в число типа Double");
           System.exit(-2);
        }
        // Создать экземпляр главного окна, передав ему массив коэффициентов
        MainFrame frame = new MainFrame(coefficients);
        // Задать действие, выполняемое при закрытии окна
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        // Показать главное окно
        frame.setVisible(true);
}
```

4. Задания

4.1 Вариант А

а) Добавить в главное меню приложения пункт «Справка», а в нём элемент «О программе», активация которого показывает диалоговое окно с указанием фамилии и группы автора программы.

б) Модифицировать (по вариантам) модель таблицы.

№ п/п	Модель таблицы
1	Добавить в модель таблицы третий столбец «Значение больше нуля?», содержащий булевские значения и принимающий значение `true`, если значение многочлена больше нуля, и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.
2	Добавить в модель таблицы третий столбец «Целая часть является квадратом?», содержащий булевские значения и принимающий значение `true`, если целая часть значения многочлена в точке является квадратом целого числа, и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.
3	Добавить в модель таблицы третий столбец «Точное значение?», содержащий булевские значения и принимающий значение `true`, если дробная часть значения многочлена равна нулю, и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.
4	Добавить в модель таблицы третий столбец «Малое число?», содержащий булевские значения и принимающий значение `true`, если целая часть значения многочлена равна нулю, и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.
5	Добавить в модель таблицы третий столбец «Дробная часть является квадратом?», содержащий булевские значения и принимающий значение `true`, если дробная часть значения многочлена в точке является записью квадрата целого числа, и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.

6	Добавить в модель таблицы третий столбец «Близко к целому», содержащий булевские значения и принимающий значение `true`, если значение многочлена в точке находится в пределах 0.1 от целого числа, и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.
7	Добавить в модель таблицы третий столбец «Целая часть чётная», содержащий булевские значения и принимающий значение `true`, если целая часть значения многочлена в точке является записью чётного числа, и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.
8	Добавить в модель таблицы третий столбец «Дробная часть нечётная», содержащий булевские значения и принимающий значение `true`, если дробная часть значения многочлена в точке является записью нечётного числа, и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.

4.2 Вариант В

а) Добавить в главное меню приложения пункт «Справка», а в нём элемент «О программе», активация которого показывает диалоговое окно с указанием фамилии и группы автора программы.
б) Модифицировать (по вариантам) модель таблицы.
в) Модифицировать (по вариантам) визуализатор ячеек таблицы.

№ п/п	Модель таблицы	Визуализатор
1	Добавить в модель таблицы третий столбец «Две пары», содержащий булевские значения и принимающий значение `true`, если в строковой записи значения многочлена две пары стоящих рядом одинаковых цифр (например, 22 и 66, 33 и 99), и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.	Изменить визуализатор таким образом, чтобы ячейки, целая часть которых чётная окрашивались в один цвет, а ячейки, целая часть которых нечётная – в другой.
2	Добавить в модель таблицы третий столбец «Ограниченная симметрия», содержащий булевские значения и принимающий значение `true`, если в строковой записи значения многочлена слева и справа от точки стоят одинаковые цифры, и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.	Изменить визуализатор таким образом, чтобы ячейки, сумма цифр дробной части которых делится нацело на 10, окрашивались в специальный цвет.
3	Добавить в модель таблицы третий столбец «Краевая симметрия», содержащий булевские значения и принимающий значение `true`, если в строковой записи значения многочлена первая и последняя цифры совпадают,	Изменить визуализатор таким образом, чтобы ячейки, сумма цифр целой части которых делится нацело на 10, окрашивались в специальный цвет.

	и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.	
4	Добавить в модель таблицы третий столбец «Взаимно простые?», содержащий булевские значения и принимающий значение `true`, если целые части X и значения многочлена в не имеют общих делителей, и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.	Изменить визуализатор таким образом, чтобы ячейки, количество цифр в дробной части которых не превосходит трёх, окрашивались в специальный цвет.
5	Добавить в модель таблицы третий столбец «Значение простое?», содержащий булевские значения и принимающий значение `true`, если целая часть значения многочлена является простым числом, и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.	Изменить визуализатор таким образом, чтобы ячейки, в дробной части которых присутствуют только числа 1, 3, 5, окрашивались в специальный цвет.
6	Добавить в модель таблицы третий столбец «Целая часть палиндром?», содержащий булевские значения и принимающий значение `true`, если строковое представление целой части значения многочлена является палиндромом (слева направо читается также, как и справа налево), и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.	Изменить визуализатор таким образом, чтобы ячейки, запись целой часть которых совпадала с записью дробной части, окрашивались в специальный цвет.
7	Добавить в модель таблицы третий столбец «Последовательный ряд?», содержащий булевские значения и принимающий значение `true`, если в целой или дробной частях значения многочлена присутствует фрагмент из как минимум трёх последовательных цифр (например, 123 или 789), и `false` – в противном случае. Обеспечить отображение значений третьего столбца как флажков.	Изменить визуализатор таким образом, чтобы ячейки, знак значения многочлена в которых не совпадал со знаком значения X в точке, окрашивались в специальный цвет.
8	Добавить в модель таблицы третий столбец «Разностороннее», содержащий булевские значения и принимающий значение `true`, если в записи целой части числа используются только чётные цифры, а в дробной – только нечётные (и наоборот), и `false` – в противном случае. Обеспечить отображение	Изменить визуализатор таким образом, чтобы ячейки, в записи которых использованы только чётные цифры, окрашивались в специальный цвет.

	значений третьего столбца как флажков.	

4.3. Вариант C

а) Добавить в главное меню приложения пункт «Справка», а в нём элемент «О программе», активация которого показывает диалоговое окно с указанием фамилии, группы и фотографии автора программы.
б) Модифицировать (по вариантам) модель таблицы.
в) Модифицировать (по вариантам) визуализатор ячеек таблицы.
г) Добавить в меню «Таблица» элемент меню, предоставляющий дополнительную функциональность (по вариантам).
д) Реализовать возможность сохранения результатов вычислений в CSV-файл (comma separated values), каждая строка таблицы в котором начинается с новой строки, а значения разделены запятыми.

№ п/п	Модель таблицы	Визуализатор	Новая функциональность
1	Добавить в модель таблицы два дополнительных столбца. В третьем показывать вычисленное значение многочлена в точке, если считать коэффициенты при степенях X с другого конца (т.е. при коэффициентах 1 2 значение второго столбца вычисляется как X + 2, а третьего как 2*X + 1), в четвёртом – разница между значениями второго и третьего столбцов.	Изменить визуализатор таким образом, чтобы он раскрашивал ячейки таблицы в шахматном порядке. При этом, в том случае, если фон ячейки – чёрный, значение в ней должно быть написано белым.	Добавить пункт меню «Найти из диапазона», запрашивающий у пользователя значения начала и конца отрезка и выделяющий в таблице цветом все ячейки, значение которых принадлежит этому диапазону.
2	Добавить в модель таблицы два дополнительных столбца. В третьем показывать значение многочлена в точке, вычисленное не по схеме Горнера, а с помощью функции возведения в степень `Math.pow()`, в четвёртом – разница между значениями второго и третьего столбцов.	Изменить визуализатор таким образом, чтобы числа меньшие нуля он выравнивал по левому краю ячейки, большие нуля – по правому краю, а ноль – посередине.	Добавить пункт меню «Найти близкие к простым», выделяющий в таблице цветом все ячейки, значение которых находится в диапазоне ±0.1 от целого простого числа.
3	Добавить в модель таблицы два дополнительных столбца. В третьем	Изменить визуализатор таким образом, чтобы в случае совпадения	Добавить пункт меню «Найти палиндромы», выделяющий цветом все

	показывать вычисленное значение многочлена в точке, если коэффициенты и результат вычислений приводить к типу `float`, в четвёртом – разница между значениями второго и третьего столбцов.	значения ячейки с искомым в столбце «Значение многочлена» отображался отмеченный флажок, а не число с плавающей точкой.	ячейки, строковое представление которых без учёта знака, отделяющего дробную часть от целой, является палиндромом, т.е. одинаково читается как слева направо, так и справа налево.

Приложение 1. Исходный код приложения

Исходный код модели таблицы

```
package bsu.rfe.java.group7.lab3.Ivanov.varB4;

import javax.swing.table.AbstractTableModel;

@SuppressWarnings("serial")
public class GornerTableModel extends AbstractTableModel {

	private Double[] coefficients;
	private Double from;
	private Double to;
	private Double step;

	public GornerTableModel(Double from, Double to, Double step,
	                        Double[] coefficients) {
		this.from = from;
		this.to = to;
		this.step = step;
		this.coefficients = coefficients;
	}

	public Double getFrom() {
		return from;
	}

	public Double getTo() {
		return to;
	}

	public Double getStep() {
		return step;
	}

	public int getColumnCount() {
		// В данной модели два столбца
		return 2;
	}

	public int getRowCount() {
		// Вычислить количество точек между началом и концом отрезка
		// исходя из шага табулирования
		return new Double(Math.ceil((to-from)/step)).intValue()+1;
	}

	public Object getValueAt(int row, int col) {
		// Вычислить значение X как НАЧАЛО_ОТРЕЗКА + ШАГ*НОМЕР_СТРОКИ
		double x = from + step*row;
		if (col==0) {
			// Если запрашивается значение 1-го столбца, то это X
			return x;
		} else {
			// Если запрашивается значение 2-го столбца, то это значение
			// многочлена
			Double result = 0.0;
			// Вычисление значения в точке по схеме Горнера.
			// Вспомнить 1-ый курс и реализовать
			// ...
```

```
                return result;
            }
        }

        public String getColumnName(int col) {
            switch (col) {
            case 0:
                // Название 1-го столбца
                return "Значение X";
            default:
                // Название 2-го столбца
                return "Значение многочлена";
            }
        }

        public Class<?> getColumnClass(int col) {
            // И в 1-ом и во 2-ом столбце находятся значения типа Double
            return Double.class;
        }

}
```

Исходный код визуализатора ячеек

```
package bsu.rfe.java.group7.lab3.Ivanov.varB4;

import java.awt.Color;
import java.awt.Component;
import java.awt.FlowLayout;
import java.text.DecimalFormat;
import java.text.DecimalFormatSymbols;
import java.text.NumberFormat;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.JTable;
import javax.swing.table.TableCellRenderer;

public class GornerTableCellRenderer implements TableCellRenderer {

        private JPanel panel = new JPanel();
        private JLabel label = new JLabel();
        // Ищем ячейки, строковое представление которых совпадает с needle
        // (иголкой). Применяется аналогия поиска иголки в стоге сена, в роли
        // стога сена - таблица
        private String needle = null;

        private DecimalFormat formatter =
                        (DecimalFormat)NumberFormat.getInstance();

        public GornerTableCellRenderer() {
            // Показывать только 5 знаков после запятой
            formatter.setMaximumFractionDigits(5);
            // Не использовать группировку (т.е. не отделять тысячи
            // ни запятыми, ни пробелами), т.е. показывать число как "1000",
            // а не "1 000" или "1,000"
            formatter.setGroupingUsed(false);
            // Установить в качестве разделителя дробной части точку, а не
            // запятую. По умолчанию, в региональных настройках
            // Россия/Беларусь дробная часть отделяется запятой
            DecimalFormatSymbols dottedDouble =
                        formatter.getDecimalFormatSymbols();
            dottedDouble.setDecimalSeparator('.');
```

```
            formatter.setDecimalFormatSymbols(dottedDouble);
            // Разместить надпись внутри панели
            panel.add(label);
            // Установить выравнивание надписи по левому краю панели
            panel.setLayout(new FlowLayout(FlowLayout.LEFT));
      }

      public Component getTableCellRendererComponent(JTable table,
      Object value, boolean isSelected, boolean hasFocus, int row, int col) {
            // Преобразовать double в строку с помощью форматировщика
            String formattedDouble = formatter.format(value);
            // Установить текст надписи равным строковому представлению числа
            label.setText(formattedDouble);
            if (col==1 && needle!=null && needle.equals(formattedDouble)) {
                  // Номер столбца = 1 (т.е. второй столбец) + иголка не null
                  // (значит что-то ищем) +
                  // значение иголки совпадает со значением ячейки таблицы -
                  // окрасить задний фон панели в красный цвет
                  panel.setBackground(Color.RED);
            } else {
                  // Иначе - в обычный белый
                  panel.setBackground(Color.WHITE);
            }
            return panel;
      }

      public void setNeedle(String needle) {
            this.needle = needle;
      }

}
```

Исходный код главного класса

```
package bsu.rfe.java.group7.lab3.Ivanov.varB4;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.PrintStream;
import javax.swing.AbstractAction;
import javax.swing.Action;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.JButton;
import javax.swing.JFileChooser;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JMenu;
import javax.swing.JMenuBar;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTable;
```

```
import javax.swing.JTextField;

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

	// Константы с исходным размером окна приложения
	private static final int WIDTH = 700;
	private static final int HEIGHT = 500;

	// Массив коэффициентов многочлена
	private Double[] coefficients;

	// Объект диалогового окна для выбора файлов
	// Компонент не создаётся изначально, т.к. может и не понадобиться
	// пользователю если тот не собирается сохранять данные в файл
	private JFileChooser fileChooser = null;

	// Элементы меню вынесены в поля данных класса, так как ими необходимо
	// манипулировать из разных мест
	private JMenuItem saveToTextMenuItem;
	private JMenuItem saveToGraphicsMenuItem;
	private JMenuItem searchValueMenuItem;

	// Поля ввода для считывания значений переменных
	private JTextField textFieldFrom;
	private JTextField textFieldTo;
	private JTextField textFieldStep;

	private Box hBoxResult;

	// Визуализатор ячеек таблицы
	private GornerTableCellRenderer renderer = new
GornerTableCellRenderer();

	// Модель данных с результатами вычислений
	private GornerTableModel data;

	public MainFrame(Double[] coefficients) {
		// Обязательный вызов конструктора предка
		super("Табулирование многочлена на отрезке по схеме Горнера");
		// Запомнить во внутреннем поле переданные коэффициенты
		this.coefficients = coefficients;
		// Установить размеры окна
		setSize(WIDTH, HEIGHT);
		Toolkit kit = Toolkit.getDefaultToolkit();
		// Отцентрировать окно приложения на экране
		setLocation((kit.getScreenSize().width - WIDTH)/2,
				(kit.getScreenSize().height - HEIGHT)/2);

		// Создать меню
		JMenuBar menuBar = new JMenuBar();
		// Установить меню в качестве главного меню приложения
		setJMenuBar(menuBar);
		// Добавить в меню пункт меню "Файл"
		JMenu fileMenu = new JMenu("Файл");
		// Добавить его в главное меню
		menuBar.add(fileMenu);
		// Создать пункт меню "Таблица"
		JMenu tableMenu = new JMenu("Таблица");
		// Добавить его в главное меню
		menuBar.add(tableMenu);

		// Создать новое "действие" по сохранению в текстовый файл
```

```
            Action saveToTextAction = new AbstractAction("Сохранить в
текстовый файл") {
                  public void actionPerformed(ActionEvent event) {
                        if (fileChooser==null) {
                              // Если экземпляр диалогового окна "Открыть
файл" ещё не создан,
                              // то создать его
                              fileChooser = new JFileChooser();
                              // и инициализировать текущей директорией
                              fileChooser.setCurrentDirectory(new File("."));
                        }
                        // Показать диалоговое окно
                        if (fileChooser.showSaveDialog(MainFrame.this) ==
JFileChooser.APPROVE_OPTION)
                              // Если результат его показа успешный,
                              // сохранить данные в текстовый файл
                              saveToTextFile(fileChooser.getSelectedFile());
                  }
            };
            // Добавить соответствующий пункт подменю в меню "Файл"
            saveToTextMenuItem = fileMenu.add(saveToTextAction);
            // По умолчанию пункт меню является недоступным (данных ещё нет)
            saveToTextMenuItem.setEnabled(false);

            // Создать новое "действие" по сохранению в текстовый файл
            Action saveToGraphicsAction = new AbstractAction("Сохранить данные
для построения графика") {
                  public void actionPerformed(ActionEvent event) {
                        if (fileChooser==null) {
                              // Если экземпляр диалогового окна
                              // "Открыть файл" ещё не создан,
                              // то создать его
                              fileChooser = new JFileChooser();
                              // и инициализировать текущей директорией
                              fileChooser.setCurrentDirectory(new File("."));
                        }
                        // Показать диалоговое окно
                        if (fileChooser.showSaveDialog(MainFrame.this) ==
JFileChooser.APPROVE_OPTION);
                              // Если результат его показа успешный,
                              // сохранить данные в двоичный файл
                              saveToGraphicsFile(
                                    fileChooser.getSelectedFile());
                  }
            };
            // Добавить соответствующий пункт подменю в меню "Файл"
            saveToGraphicsMenuItem = fileMenu.add(saveToGraphicsAction);
            // По умолчанию пункт меню является недоступным(данных ещё нет)
            saveToGraphicsMenuItem.setEnabled(false);

            // Создать новое действие по поиску значений многочлена
            Action searchValueAction = new AbstractAction("Найти значение
многочлена") {
                  public void actionPerformed(ActionEvent event) {
                        // Запросить пользователя ввести искомую строку
                        String value =
JOptionPane.showInputDialog(MainFrame.this, "Введите значение для поиска",
"Поиск значения", JOptionPane.QUESTION_MESSAGE);
                        // Установить введенное значение в качестве иголки
                        renderer.setNeedle(value);
                        // Обновить таблицу
                        getContentPane().repaint();
                  }
            };
```

```
        // Добавить действие в меню "Таблица"
        searchValueMenuItem = tableMenu.add(searchValueAction);
        // По умолчанию пункт меню является недоступным (данных ещё нет)

        searchValueMenuItem.setEnabled(false);

        // Создать область с полями ввода для границ отрезка и шага
        // Создать подпись для ввода левой границы отрезка
        JLabel labelForFrom = new JLabel("X изменяется на интервале от:");
        // Создать текстовое поле для ввода значения длиной в 10 символов
        // со значением по умолчанию 0.0
        textFieldFrom = new JTextField("0.0", 10);
        // Установить максимальный размер равный предпочтительному, чтобы
        // предотвратить увеличение размера поля ввода
        textFieldFrom.setMaximumSize(textFieldFrom.getPreferredSize());
        // Создать подпись для ввода левой границы отрезка
        JLabel labelForTo = new JLabel("до:");
        // Создать текстовое поле для ввода значения длиной в 10 символов
        // со значением по умолчанию 1.0
        textFieldTo = new JTextField("1.0", 10);
        // Установить максимальный размер равный предпочтительному, чтобы
        // предотвратить увеличение размера поля ввода
        textFieldTo.setMaximumSize(textFieldTo.getPreferredSize());
        // Создать подпись для ввода шага табулирования
        JLabel labelForStep = new JLabel("с шагом:");
        // Создать текстовое поле для ввода значения длиной в 10 символов
        // со значением по умолчанию 1.0
        textFieldStep = new JTextField("0.1", 10);
        // Установить максимальный размер равный предпочтительному, чтобы
        // предотвратить увеличение размера поля ввода
        textFieldStep.setMaximumSize(textFieldStep.getPreferredSize());
        // Создать контейнер 1 типа "коробка с горизонтальной укладкой"
        Box hboxRange = Box.createHorizontalBox();
        // Задать для контейнера тип рамки "объёмная"
        hboxRange.setBorder(BorderFactory.createBevelBorder(1));
        // Добавить "клей" С1-Н1
        hboxRange.add(Box.createHorizontalGlue());
        // Добавить подпись "От"
        hboxRange.add(labelForFrom);
        // Добавить "распорку" С1-Н2
        hboxRange.add(Box.createHorizontalStrut(10));
        // Добавить поле ввода "От"
        hboxRange.add(textFieldFrom);
        // Добавить "распорку" С1-Н3
        hboxRange.add(Box.createHorizontalStrut(20));
        // Добавить подпись "До"
        hboxRange.add(labelForTo);
        // Добавить "распорку" С1-Н4
        hboxRange.add(Box.createHorizontalStrut(10));
        // Добавить поле ввода "До"
        hboxRange.add(textFieldTo);
        // Добавить "распорку" С1-Н5
        hboxRange.add(Box.createHorizontalStrut(20));
        // Добавить подпись "с шагом"
        hboxRange.add(labelForStep);
        // Добавить "распорку" С1-Н6
        hboxRange.add(Box.createHorizontalStrut(10));
        // Добавить поле для ввода шага табулирования
        hboxRange.add(textFieldStep);
        // Добавить "клей" С1-Н7
        hboxRange.add(Box.createHorizontalGlue());
        // Установить предпочтительный размер области равным удвоенному
        // минимальному, чтобы при  компоновке область совсем не сдавили
        hboxRange.setPreferredSize(new Dimension(
```

```
                new Double(hboxRange.getMaximumSize().getWidth()).intValue(),
                new Double(hboxRange.getMinimumSize().getHeight()).intValue()*2));
                // Установить область в верхнюю (северную) часть компоновки
                getContentPane().add(hboxRange, BorderLayout.NORTH);

                // Создать кнопку "Вычислить"
                JButton buttonCalc = new JButton("Вычислить");
                // Задать действие на нажатие "Вычислить" и привязать к кнопке
                buttonCalc.addActionListener(new ActionListener() {
                      public void actionPerformed(ActionEvent ev) {
                            try {
                                  // Считать значения начала и конца отрезка, шага
                                  Double from =
Double.parseDouble(textFieldFrom.getText());
                                  Double to =
Double.parseDouble(textFieldTo.getText());
                                  Double step =
Double.parseDouble(textFieldStep.getText());
                                  // На основе считанных данных создать новый
экземпляр модели таблицы
                                  data = new GornerTableModel(from, to, step,
MainFrame.this.coefficients);
                                  // Создать новый экземпляр таблицы
                                  JTable table = new JTable(data);
                                  // Установить в качестве визуализатора ячеек для
класса Double разработанный визуализатор
                                  table.setDefaultRenderer(Double.class,
renderer);
                                  // Установить размер строки таблицы в 30
пикселов
                                  table.setRowHeight(30);
                                  // Удалить все вложенные элементы из контейнера
hBoxResult
                                  hBoxResult.removeAll();
                                  // Добавить в hBoxResult таблицу, "обёрнутую" в
панель с полосами прокрутки
                                  hBoxResult.add(new JScrollPane(table));
                                  // Обновить область содержания главного окна
                                  getContentPane().validate();
                                  // Пометить ряд элементов меню как доступных
                                  saveToTextMenuItem.setEnabled(true);
                                  saveToGraphicsMenuItem.setEnabled(true);
                                  searchValueMenuItem.setEnabled(true);
                            } catch (NumberFormatException ex) {
                                  // В случае ошибки преобразования чисел показать
сообщение об ошибке
                                  JOptionPane.showMessageDialog(MainFrame.this,
"Ошибка в формате записи числа с плавающей точкой", "Ошибочный формат числа",
JOptionPane.WARNING_MESSAGE);
                            }
                      }
                });
                // Создать кнопку "Очистить поля"
                JButton buttonReset = new JButton("Очистить поля");
                // Задать действие на нажатие "Очистить поля" и привязать к кнопке

                buttonReset.addActionListener(new ActionListener() {
                      public void actionPerformed(ActionEvent ev) {
                            // Установить в полях ввода значения по умолчанию
                            textFieldFrom.setText("0.0");
                            textFieldTo.setText("1.0");
                            textFieldStep.setText("0.1");
                            // Удалить все вложенные элементы контейнера
hBoxResult
```

```
                        hBoxResult.removeAll();
                        // Добавить в контейнер пустую панель
                        hBoxResult.add(new JPanel());
                        // Пометить элементы меню как недоступные
                        saveToTextMenuItem.setEnabled(false);
                        saveToGraphicsMenuItem.setEnabled(false);
                        searchValueMenuItem.setEnabled(false);
                        // Обновить область содержания главного окна
                        getContentPane().validate();
                  }
            });
            // Поместить созданные кнопки в контейнер
            Box hboxButtons = Box.createHorizontalBox();
            hboxButtons.setBorder(BorderFactory.createBevelBorder(1));
            hboxButtons.add(Box.createHorizontalGlue());
            hboxButtons.add(buttonCalc);
            hboxButtons.add(Box.createHorizontalStrut(30));
            hboxButtons.add(buttonReset);
            hboxButtons.add(Box.createHorizontalGlue());
            // Установить предпочтительный размер области равным удвоенному
минимальному, чтобы при
            // компоновке окна область совсем не сдавили
            hboxButtons.setPreferredSize(new Dimension(new
Double(hboxButtons.getMaximumSize().getWidth()).intValue(), new
Double(hboxButtons.getMinimumSize().getHeight()).intValue()*2));
            // Разместить контейнер с кнопками в нижней (южной) области
граничной компоновки
            getContentPane().add(hboxButtons, BorderLayout.SOUTH);
            // Область для вывода результата пока что пустая
            hBoxResult = Box.createHorizontalBox();
            hBoxResult.add(new JPanel());
            // Установить контейнер hBoxResult в главной (центральной) области
граничной компоновки
            getContentPane().add(hBoxResult, BorderLayout.CENTER);
      }

      protected void saveToGraphicsFile(File selectedFile) {
            try {
                  // Создать новый байтовый поток вывода, направленный в
указанный файл
                  DataOutputStream out = new DataOutputStream(new
FileOutputStream(selectedFile));
                  // Записать в поток вывода попарно значение X в точке,
значение многочлена в точке
                  for (int i = 0; i<data.getRowCount(); i++) {
                        out.writeDouble((Double)data.getValueAt(i,0));
                        out.writeDouble((Double)data.getValueAt(i,1));
                  }
                  // Закрыть поток вывода
                  out.close();
            } catch (Exception e) {
                  // Исключительную ситуацию "ФайлНеНайден" в данном случае
можно не обрабатывать,
                  // так как мы файл создаём, а не открываем для чтения
            }
      }

      protected void saveToTextFile(File selectedFile) {
            try {
                  // Создать новый символьный поток вывода, направленный в
указанный файл
                  PrintStream out = new PrintStream(selectedFile);
                  // Записать в поток вывода заголовочные сведения
```

```
                  out.println("Результаты табулирования многочлена по схеме
Горнера");
                  out.print("Многочлен: ");
                  for (int i=0; i<coefficients.length; i++) {
                        out.print(coefficients[i] + "*X^" +
(coefficients.length-i-1));
                        if (i!=coefficients.length-1)
                              out.print(" + ");
                  }
                  out.println("");
                  out.println("Интервал от " + data.getFrom() + " до " +
data.getTo() + " с шагом " + data.getStep());

      out.println("=====================================================");
                  // Записать в поток вывода значения в точках
                  for (int i = 0; i<data.getRowCount(); i++) {
                        out.println("Значение в точке " + data.getValueAt(i,0)
+ " равно " + data.getValueAt(i,1));
                  }
                  // Закрыть поток
                  out.close();
            } catch (FileNotFoundException e) {
                  // Исключительную ситуацию "ФайлНеНайден" можно не
                  // обрабатывать, так как мы файл создаём, а не открываем
            }
      }

      public static void main(String[] args) {
            // Если не задано ни одного аргумента командной строки -
            // Продолжать вычисления невозможно, коэффициенты неизвестны
            if (args.length==0) {
                  System.out.println("Невозможно табулировать многочлен, для
которого не задано ни одного коэффициента!");
                  System.exit(-1);
            }
            // Зарезервировать места в массиве коэффициентов столько, сколько
аргументов командной строки
            Double[] coefficients = new Double[args.length];
            int i = 0;
            try {
                  // Перебрать аргументы, пытаясь преобразовать их в Double
                  for (String arg: args) {
                        coefficients[i++] = Double.parseDouble(arg);
                  }
            }
            catch (NumberFormatException ex) {
                  // Если преобразование невозможно - сообщить об ошибке и
завершиться
                  System.out.println("Ошибка преобразования строки '" +
args[i] + "' в число типа Double");
                  System.exit(-2);
            }
            // Создать экземпляр главного окна, передав ему коэффициенты
            MainFrame frame = new MainFrame(coefficients);
            // Задать действие, выполняемое при закрытии окна
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
      }
}
```