# ТЕМА 2
# Разработка простейшего приложения с оконным интерфейсом

## Цель лабораторной работы

Освоить основные принципы написания оконных приложений с графическим интерфейсом на языке Java в среде Eclipse.

## 1 Особенности построения приложений с оконным интерфейсом

В лабораторной работе №1 мы разработали консольное приложение, реализующее взаимодействие с пользователем только через стандартные потоки ввода-вывода (предоставляемые операционной системой) и не использующее средств графического интерфейса для организации этого взаимодействия. Как мы видели, при запуске в ОС Windows консольного приложения появляется окно, содержащее выводимый приложением текст. Однако данное окно является лишь выбранным в ОС Windows способом отображения потока вывода и организации потока ввода, а не окном приложения. В ОС Linux, например, никакого окна не возникло бы, а текст был бы отображён просто в текущем окне.

В отличие от консольного приложения, приложение с графическим (оконным) интерфейсом обладает одним или несколькими окнами верхнего уровня (т.е. окнами, которые не являются вложенными в другие окна), называемыми в Java *фреймами*.

**Замечание:** так как в данной работе рассматриваются только окна верхнего уровня, то фрейм и окно будут использоваться как синонимы.

Фрейм содержит обязательные компоненты – полосу заголовка, содержащую название окна, пиктограмму для системного меню и набор кнопок управления состоянием окна (минимизации, максимизации, закрытия и т.п.). Фрейм является контейнером, т.е. может содержать вложенные элементы графического интерфейса пользователя (GUI – Graphical User Interface): надписи, поля ввода, выпадающие списки, полосы прокрутки, меню и т.д., позволяющие организовать взаимодействие с пользователем, реагируя на его действия – нажатие клавиш на клавиатуре, движения и щелчки мышью.

С выходом Java 1.0 для разработки GUI была предложена библиотека классов «абстрактного оконного инструментария» (AWT – Abstract Window Toolkit). Основой функционирования элементов интерфейса было делегирование механизмов создания и поведения встроенному инструментарию GUI операционной системы. Это должно было позволить создавать переносимые приложения, внешний вид которых определялся той ОС, в которой работало приложение. Тем не менее, отличия в поведении одних и тех же компонентов в различных ОС и серьёзная разница в

возможностях и количестве доступных компонентов на различных платформах существенно ограничили полезность и применимость данной библиотеки. Альтернативным подходом, предложенным Netscape в 1996 году, стала библиотека основных классов Интернет IFC (Internet Foundation Classes), основанная на принципе отображения всех компонентов интерфейса в пространстве пустых окон. Единственным требованием к использованию IFC было наличие механизмов создания окна и рисования на его поверхности. При этом компоненты IFC выглядели и вели себя одинаково, не завися от платформы, на которой запущено приложение. В дальнейшем, совместно с компанией Sun данный подход был усовершенствован, и результатом сотрудничества стала библиотека GUI-компонентов, названная Swing.

Для представления фреймов в библиотеке AWT существует класс `Frame`. Так как классы AWT тесно связаны с встроенными компонентами ОС, то внешний вид окна (рамка, заголовок, иконки, управляющие кнопки) будет зависеть от ОС, в которой работает приложение.

В библиотеке Swing для представления фрейма существует класс `JFrame`. Так как `JFrame` является потомком `Frame`, то его внешний вид также отображается оконной системой ОС, а не Swing.

Другим важным отличием консольных приложений от приложений с графическим интерфейсом является принципиально различная схема выполнения. Большинство консольных приложений разрабатывается для решения конкретной задачи и может быть описано одним или несколькими алгоритмами, выполняемыми последовательно или параллельно. Если на каком-либо этапе вычислений от пользователя требуются дополнительные сведения, то до момента их получения выполнение блокируется и возобновляется только после ввода данных. Это делает подобные приложения последовательными (например, на рисунке 1.1 показана блок-схема консольного приложения табулирования функции на отрезке). Приложение с графическим интерфейсом значительную часть времени проводит в ожидании событий, причиной которых в большинстве случаев являются действия пользователя (причина ряда событий может быть и другой, например – уведомление от таймера). По этой причине модель вычислений в таких приложениях не последовательная, а событийно-управляемая. Программа постоянно находится в бесконечном цикле обработки сообщений о произошедших событиях, реагируя на них определённым разработчиком способом (на рисунке 1.2 показана упрощённая блок-схема приложения с графическим интерфейсом для табулирования функции на отрезке). Средством выхода из этого бесконечного цикла является событие закрытия окна приложения, для

которого можно задать различную реакцию (в частности – завершение всего приложения).

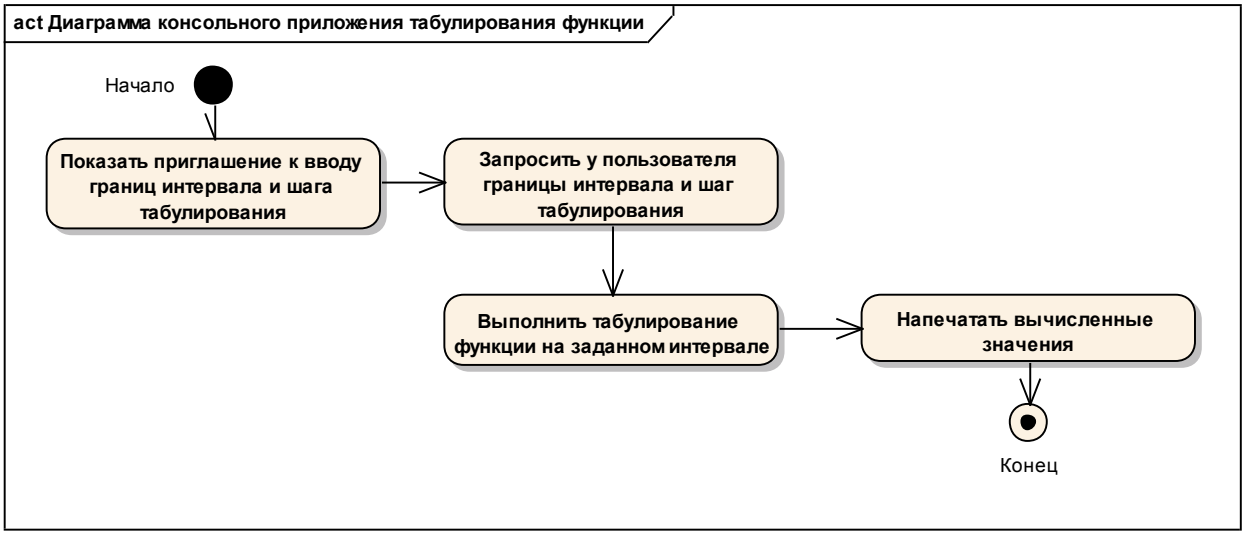


**Рисунок 1.1 – Диаграмма консольного приложения табулирования функции на отрезке**

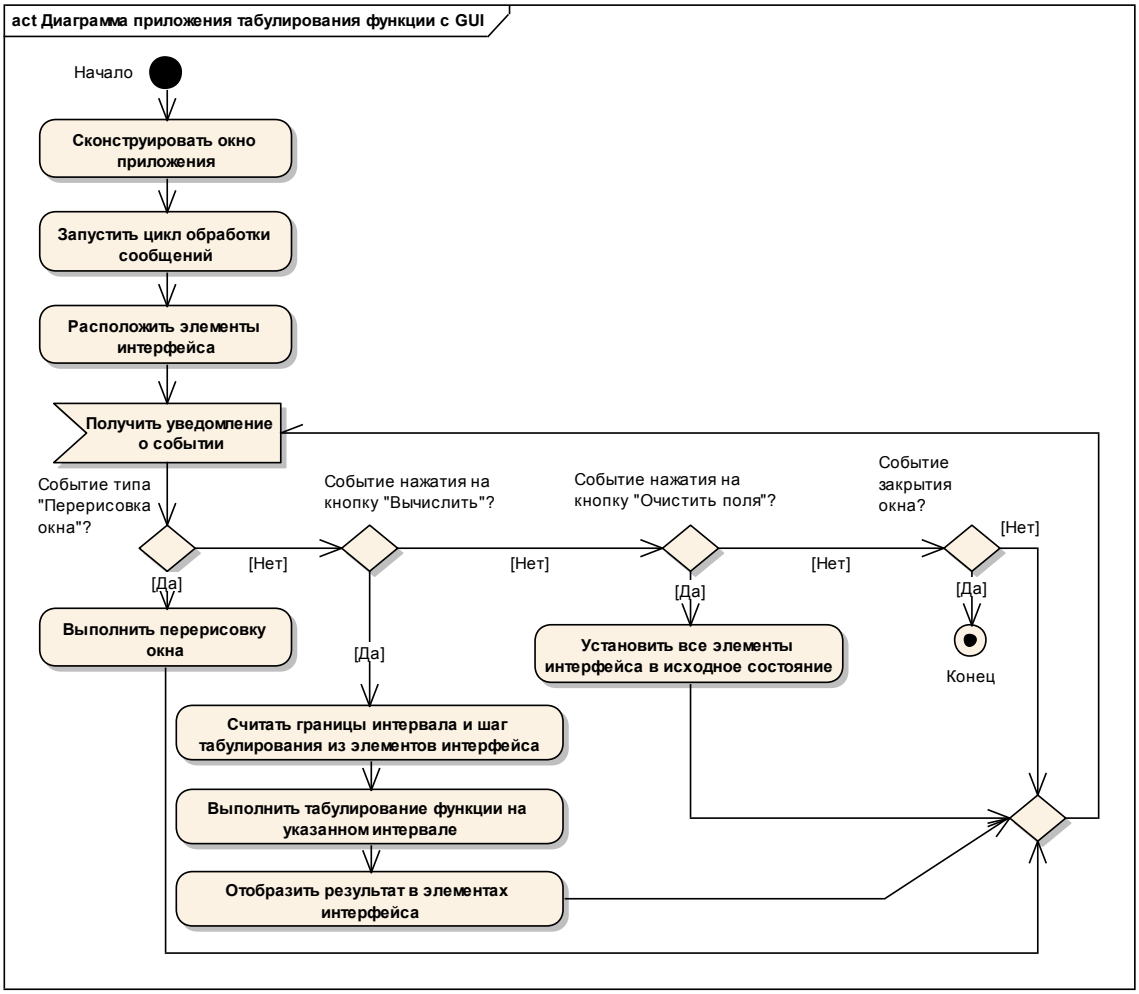


**Рисунок 1.2 – Диаграмма приложения табулирования функции, обладающего графическим интерфейсом**

Подобная схема работы обуславливает то, что в отличие от простейшего консольного приложения (являющегося однопоточным), простейшее приложение с графическим интерфейсом обладает как минимум двумя потоками.

**В первом потоке** запускается главный метод `main()` главного класса приложения, который:

- создаёт экземпляр фрейма;
- задаёт реакцию на закрытие окна приложения (например, завершение приложения);
- отображает фрейм на экране.

При этом создаётся второй поток для обработки поступающих сообщений о событиях, `main()` завершается, но приложение продолжает функционировать.

**Во втором потоке** в цикле анализируется очередь поступивших событий, для которых вызываются соответствующие обработчики.

Ниже приведен пример простейшего метода `main()` запускаемого класса приложения:

```
public static void main(String[] args) {
	// Конструируем экземпляр фрейма
	JFrame frame = new JFrame();
	// Задаём реакцию на нажатие кнопки закрытия фрейма
	frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	// Показываем фрейм на экране
	frame.setVisible(true);
}
```

При его выполнении на экране появится окно (рисунок 1.3), имеющее по умолчанию размер 0x0 точек. Окно может быть увеличено до нужного размера стандартным для Windows способом (потянув за угол окна), см. рисунок 1.4. Как видно, фрейм обладает рамкой, строкой заголовка (изначально заголовок отсутствует), иконкой системного меню и кнопками управления состоянием.

**Рисунок 1.3 – Пример пустого фрейма**

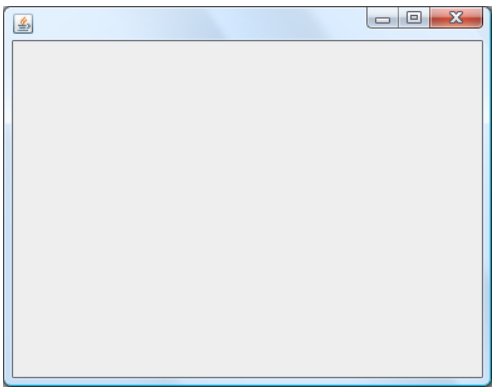

**Рисунок 1.4 – Увеличенный (вручную) пустой фрейм**

Важным отличием процесса создания графического интерфейса пользователя в Java от других платформ и сред разработки (Visual Basic, Delphi и т.д.) является принципиально другая схема размещения элементов интерфейса в окне. Например, в Delphi элементы размещаются разработчиком в пространстве окна с помощью специализированного редактора, после чего их положение и размеры остаются неизменными. Сложность данного подхода заключается в его низкой универсальности и переносимости:

- Пользователь может установить в ОС увеличенный размер шрифта, чтобы облегчить читаемость текста, и размер элементов интерфейса будет недостаточным для того, чтобы вместить надписи, напечатанные шрифтом такого размера.
- При локализации разработанного программного продукта для другого региона или страны может оказаться, что названия элементов интерфейса, переведенные на другой язык, будут длиннее, и, опять же, не смогут поместиться в область, зарезервированную разработчиком.

Для исправления этих недостатков в Java принята концепция размещения элементов интерфейса на основе *менеджеров компоновки*. В задачи менеджера входит расположение и масштабирование компонентов друг относительно друга. Разработчик, тем не менее, может задавать общие правила размещения, например: размещать компоненты в одной строке слева направо, по достижении правого края экрана продолжить размещение с новой строки; по возможности максимально удалить компоненты друг от друга; установить предпочтительный размер компонента 200x50 точек и т.д. Таким образом, вместо жёсткого задания размеров и привязки элементов интерфейса имеет место их динамическое позиционирование с учётом правил, заданных разработчиком.

# 2 Краткая справка по необходимым программным компонентам

## 2.1 Представление фрейма – класс JFrame

Создание окна верхнего уровня, представляемого в Java классом `JFrame`, рассмотрим на приведенном выше примере метода `main()` выполняемого класса приложения:

```
JFrame frame = new JFrame();
```

Создание экземпляра окна не приводит автоматически к его появлению на экране. Размеры окна (по умолчанию 0x0 точек) устанавливаются в конструкторе класса-потомка с помощью вызова метода `setSize(newWidth, newHeight)`.

Важным является задание реакции на закрытие окна приложения, осуществляемое с помощью метода `setDefaultCloseOperation`:

```
// завершение приложения при закрытии окна
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

По умолчанию, в Java окно при закрытии становится невидимым, но приложение не завершается (что активно используется в случае многооконных приложений, так как закрытие одного из нескольких открытых окон не приводит к завершению приложения). Для однооконного приложения реакция системы на закрытие окна требует переопределения, так как закрытие единственного окна лишает пользователя возможности взаимодействия с приложением (в этом случае оно может быть удалено из системы только с помощью специализированных средств – в случае Windows – диспетчера задач).

Так как создание экземпляра окна автоматически не приводит к его появлению на экране, это позволяет разработчику добавить компоненты интерфейса перед показом окна. Для отображения окна используется метод `setVisible(true)`.

Класс `JFrame` обладает также рядом полезных методов для задания/считывания его характеристик (таблица 2.1). Большинство этих методов определено в предках класса `JFrame` – `Component` и `Frame`.

Таблица 2.1 – Методы класса фрейма

| Название метода | Реализован в классе | Описание |
|---|---|---|
| `boolean isVisible()` | `Component` | Позволяет проверить состояние видимости компонента (в случае `JFrame` – окна) |
| `void setVisible(boolean b)` | `Component` | Задаёт видимость компонента |
| `boolean isEnabled()` | `Component` | Проверяет доступность компонента |
| `void setEnabled(boolean` | `Component` | Делает компонент (окно) доступным |

| | | |
|---|---|---|
| isEnabled) | | |
| Point getLocation() | Component | Возвращает положение верхнего левого угла компонента по отношению к левому верхнему углу компонента-контейнера |
| setLocation(int xPos, int yPos) | Component | Задаёт положение верхнего левого угла компонента относительно верхнего левого угла компонента-контейнера. В случае окна (контейнера верхнего уровня) – относительно верхнего левого угла экрана. |
| Dimension getSize() | Component | Возвращает текущие размеры компонента |
| setSize(int width, int height)<br>setSize(Dimension d) | Component | Задаёт размеры компонента |
| setResizable(boolean isResizable) | Frame | Задаёт возможность изменения размеров окна пользователем |
| setTitle(String title) | Frame | Задаёт название окна в заголовке |
| setIconImage(Image icon) | Frame | Задаёт изображение для использования в виде иконки окна |

С точки зрения отображения информации на экране, внутренняя структура фрейма является достаточно сложной (рисунок 2.1).

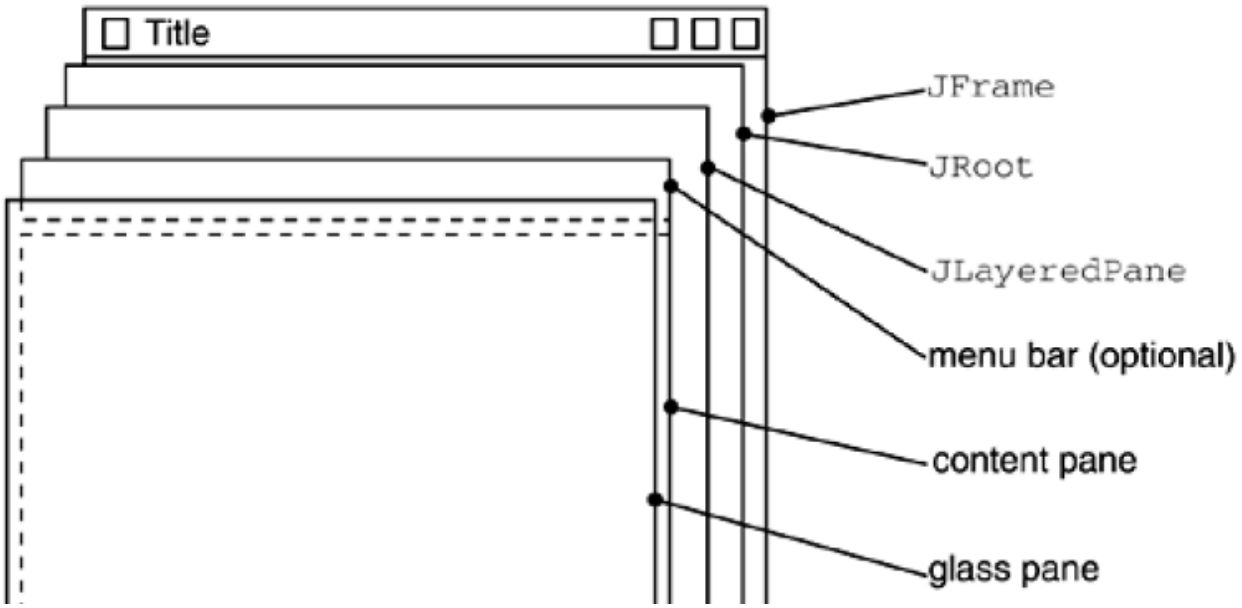


**Рисунок 2.1 – Внутренняя структура фрейма**

Основное содержание фрейма находится на поверхности содержания (content pane). Вывод информации непосредственно на поверхность фрейма считается признаком плохого стиля. В Java окна выступают в роли контейнеров, содержащих компоненты, поэтому обычно информация выводится на специальные компоненты, называемые панелями (класс `JPanel`), которые добавляются в окно. Диаграмма классов для компонентов `JFrame` и `JPanel` представлена на рисунке 2.2.

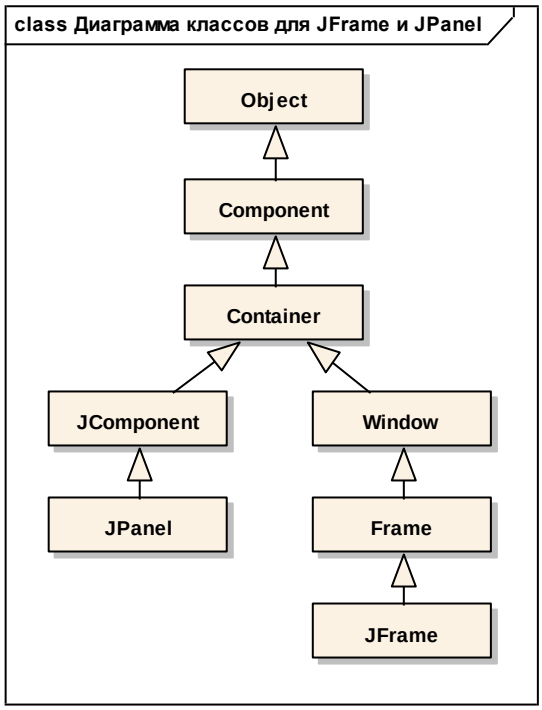


**Рисунок 2.2 – Диаграмма классов для компонентов `JFrame` и `JPanel`**

## 2.2 Получение информации от операционной системы – класс Toolkit

В некоторых случаях, например при позиционировании или масштабировании окна, для получения ряда характеристик (разрешения экрана и т.п.), необходимо взаимодействие с ОС, в которой выполняется приложение. Подобная информация получается в Java с помощью класса `Toolkit` (инструментарий), для получения экземпляра которого следует воспользоваться методом `getDefaultToolkit()`:

```
Toolkit kit = Toolkit.getDefaultToolkit();
```

Например, получить размеры экрана можно следующим образом:

```
// получить размеры экрана
Dimension screenSize = kit.getScreenSize();
int screenWidth = screenSize.width;
int screenHeight = screenSize.height;
```

Операция загрузки изображения (зависящая от операционной системы) также выполняется с помощью `Toolkit`:

```
// загрузка изображения и установка его в качестве иконки
Image img = kit.getImage("icon.gif");
setIconImage(img);
```

## 2.3 Управление компоновками компонентов в окне

Как уже отмечалось, размещение компонентов внутри окон в Java решается посредством *менеджеров компоновки*, в задачи которых входит относительное позиционирование элементов друг относительно друга.

### *2.3.1 «Плавающая» компоновка*

Для компонента *панель* (`JPanel`) «плавающая» компоновка устанавливается по умолчанию. При этом принцип размещения элементов таков: компоненты выстраиваются горизонтально, пока хватает места до правой границы окна, после чего начинается новая строка компонентов. При изменении размеров контейнера, менеджер автоматически обновляет размещение компонентов для заполнения доступного пространства. Для компоновки можно задать способ выравнивания компонентов в каждой строке: по левому краю, по правому краю, по центру (по умолчанию). Смена менеджера компоновки осуществляется следующим образом:

```
container.setLayout(new FlowLayout(FlowLayout.LEFT));
```

Конструктор `FlowLayout(int align, int hgap, int vgap)` позволяет задать горизонтальные и вертикальные промежутки между компонентами.

### *2.3.2 «Граничная» компоновка*

«Граничная» компоновка является компоновкой по умолчанию любого фрейма (`JFrame`). В отличие от плавающей компоновки, она позволяет задавать места размещения компонентов: центр, север, юг, запад, восток (рисунок 2.3).

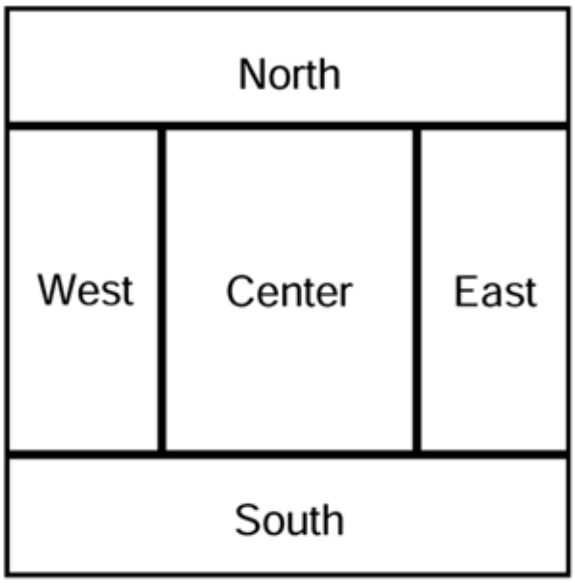


**Рисунок 2.3 – Области «граничной» компоновки**

Принцип размещения таков: сначала рассчитывается место, занимаемое краевыми компонентами, после чего всё оставшееся место отводится

центральной области. При изменении размером контейнера изменяются размеры только центральной области.

Инициализация граничной компоновки, добавление в неё компонентов и активация осуществляются следующим образом:

```
// Создать экземпляр граничной компоновки с промежутками между ячейками
// по горизонтали – hgap, по вертикали - vgap
BorderLayout myLayout = new BorderLayout(hgap, vgap);
// Добавить myComponent1 в верхнюю часть компоновки (север)
myLayout.add(myComponent1, BorderLayout.NORTH);
// Добавить myComponent2 в нижнюю часть компоновки (юг)
myLayout.add(myComponent2, BorderLayout.SOUTH);
// Установить граничную компоновку в качестве активной компоновки
// контейнера container
container.setLayout(myLayout);
```

**Внимание:** в отличие от плавающей компоновки, граничная компоновка изменяет размер всех компонентов для заполнения доступного пространства (рисунок 2.4).


**Рисунок 2.4 – Увеличение размера кнопки в граничной компоновке**

**Совет:** для предотвращения масштабирования компонентов, добавляемых в граничную компоновку, следует использовать контейнер панель (`JPanel`), имеющий компоновкой по умолчанию плавающую компоновку (рисунок 2.5):

```
// Создать экземпляр граничной компоновки с промежутками между ячейками
// по горизонтали – hgap, по вертикали - vgap
BorderLayout myLayout = new BorderLayout(hgap, vgap);
// Создать новый экземпляр контейнера JPanel
JPanel myPanel = new JPanel();
// Добавить в контейнер myComponent1
myPanel.add(myComponent1);
// Добавить в контейнер myComponent1
myPanel.add(myComponent2);
// Добавить контейнер myPanel в центральную часть граничной компоновки
myLayout.add(myPanel, BorderLayout.CENTER);
// Установить граничную компоновку в качестве активной компоновки
// контейнера container
container.setLayout(myLayout);
```

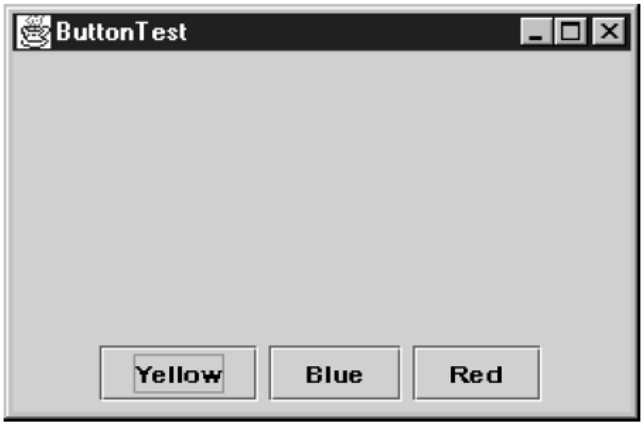


**Рисунок 2.5 – Размер компонентов при использовании контейнера `JPanel`**

### *2.3.3 «Табличная» компоновка*

Табличная компоновка организует все компоненты в строки и столбцы, как в таблице. Её особенностью является равенство размеров всех ячеек (рисунок 2.6).

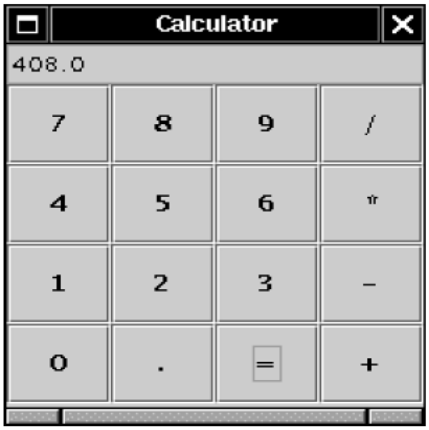


**Рисунок 2.6 – Табличная компоновка с ячейками одинакового размера**

Для создания компоновки предложены два альтернативных конструктора (один из которых позволяет также задавать горизонтальные и вертикальные промежутки между ячейками):

```
// 5 строк, 4 столбца
GridLayout myLayout1 = new GridLayout(5,4);
// 5 строк, 4 столбца, гор. промежуток - 10, верт. - 5
GridLayout myLayout2 = new GridLayout(5, 4, 10, 5);
```

Добавление компонентов в компоновку происходит с помощью метода `add(Component c)` последовательно: сначала заполняется 1-ый столбец 1-ой строки, потом 2-ой столбец 1-ой строки и т.д.

### *2.3.4 «Коробочная» компоновка*

Коробочная компоновка позволяет поместить строку или столбец компонентов, что, с учётом возможности вложения контейнеров,

предоставляет более гибкие возможности, чем табличная компоновка. Горизонтальная и вертикальная компоновки создаются по-разному:

```
// создать «коробку» с горизонтальной укладкой
Box b1 = Box.createHorizontalBox();
// создать «коробку» с вертикальной укладкой
Box b2 = Box.createVerticalBox();
```

При размещении каждый компонент имеет три размера: *желаемый*, *максимальный* и *минимальный*. При вычислении размеров менеджер руководствуется алгоритмом:

1. Вычислить максимальную высоту самого высокого компонента.
2. Попытаться растянуть высоту всех компонентов до этого уровня.
3. Если какой-либо компонент не желает растягиваться до такой высоты, то определить его вертикальное выравнивание посредством метода `getAlignmentY()`. Метод возвращает число с плавающей точкой, которое интерпретируется так: 0 – выровнять по верху, 1 – выровнять по низу, 0.5 – выровнять по центру. По вертикали компонент выравнивается аналогично.
4. Получить желаемую ширину каждого компонента. Сложить полученные для всех компонентов значения.
5. Если общая желаемая ширина меньше доступного места, расширить компоненты до их максимальной ширины. После этого выстроить компоненты слева направо без дополнительных пробелов между ними. Если желаемая ширина больше доступного места, то сжать компоненты до их минимального размера, но не более. Если все компоненты не помещаются даже при минимальной ширине, показать не все из них.

Для вертикальной укладки алгоритм аналогичен.

**Замечание:** так как коробочная компоновка увеличивает размер компонентов за пределы желаемого, то для компонентов, масштабирование которых нежелательно (например, поля ввода), необходимо принудительно сделать максимальный размер равным желаемому:

```
textField.setMaximumSize(textField.getPreferredSize());
```

Так как по умолчанию между компонентами в коробочной компоновке нет промежутков, то предусмотрены три типа невидимых *наполнителей*:

- распорки (strut), добавляющие некоторый фиксированный промежуток между компонентами:

```
box.add(label);
box.add(Box.createHorizontalStrut(10));
box.add(textField);
```

- неподвижные области (rigid area), добавляющие не только промежуток, но и минимальный/максимальный/желаемый размер по другому направлению:

```
box.add(Box.createRigidArea(new Dimension(5, 20));
```

- клей (glue), максимальным образом удаляющий компоненты друг от друга:

```
box.add(button1);
box.add(Box.createGlue());
box.add(button2);
```

#### *2.3.5 «Смешанная» компоновка*

Является наиболее сложной компоновкой, которую можно рассматривать как табличную компоновку без ограничений: её ячейки могут объединяться, вложенные компоненты не масштабируются на всё доступное пространство, их выравнивание может быть задано индивидуально для каждой ячейки.

**Замечание:** смешанная компоновка является достаточно сложной в реализации.

#### *2.3.6 Произвольная компоновка*

В некоторых случаях может оказаться необходимым поместить компоненты в фиксированных областях (т.н. *абсолютное позиционирование*). Этот подход не рекомендуется для переносимых приложений, но может успешно использоваться для быстрого создания прототипов приложений.

В этом случае необходимо: 1) установить менеджер компоновки равным `null`; 2) добавить компоненты в контейнер; 3) указать размер и положение каждого компонента:

```
contentPane.setLayout(null);
JButton ok = new JButton("Ok");
contentPane.add(ok);
ok.setBounds(10, 10, 30, 15);
```

### 2.4 Показ диалоговых окон

Для вывода простейших сообщений класс `JOptionPane` предоставляет 4 статических метода (таблица 2.2).

Таблица 2.2 – Методы для вывода диалоговых окон

| Метод | Описание |
|---|---|
| `showMessageDialog` | Выводит сообщение и ожидает, пока пользователь не нажмёт |

| | |
|---|---|
| | OK |
| `showConfirmDialog` | Выводит сообщение и ожидает подтверждения (OK / Cancel) |
| `showOptionDialog` | Выводит сообщение и ожидает выбора одного из нескольких вариантов |
| `showInputDialog` | Выводит сообщение и ожидает от пользователя ввода строки |

Выводимое сообщение может иметь вид текстовой строки, изображения, компонента интерфейса. Пиктограмма диалогового окна определяется его типом (таблица 2.3):

Таблица 2.3 – Типы диалоговых окон

| Тип | Описание |
|---|---|
| ERROR_MESSAGE | Сообщение об ошибке (обычно, красный знак STOP) |
| INFORMATION_MESSAGE | Информационное сообщение (обычно, синий знак i) |
| WARNING_MESSAGE | Предупреждение (обычно, жёлтый восклицательный знак) |
| QUESTION_MESSAGE | Вопросительное сообщение (обычно, вопросительный знак) |
| PLAIN_MESSAGE | Сообщение без пиктограммы сбоку |

### 2.5 Вывод графической информации

Для вывода на экран графической информации (в том числе и текста с использованием TrueType-шрифтов) обычно используется компонент `JPanel`. Причин его популярности две:

- панель содержит поверхность, на которой можно рисовать (переопределив метод перерисовки);
- панель сама по себе является контейнером, т.е. может содержать вложенные компоненты.

За прорисовку панели отвечает метод `paintComponent()`, принимающий в качестве аргумента экземпляр класса `Graphics` – представление контекста отображения информации. Каждый раз, когда окно требует перерисовки (по любой причине), обработчик событий уведомляет компоненты, что приводит к вызову `paintComponent()` для всех компонентов.

Экземпляр `Graphics` измеряет размеры в пикселах, значение (0, 0) соответствует верхнему левому углу перерисовываемого компонента.

Для вывода изображения в область окна, представляемую экземпляром `Graphics`, следует использовать вызовы:

```
// вывести изображение в оригинальном размере
g.drawImage(imageToDraw, xPos, yPos, null);
```

```
// вывести изображение с масштабированием
g.drawImage(imageToDraw, xPos, yPos, width, height, null);
// для копирования области вывода
// deltaX – смещение в пикселах по горизонтали от fromXPos
// deltaY – смещение в пикселах по вертикали от точки fromYPos
g.copyArea(fromXPos, fromYPos, width, height, deltaX, deltaY);
```

### 2.6 Прочие компоненты интерфейса пользователя

Приложения, реализующие графический интерфейс, сталкиваются с необходимостью обработки реакции на действия пользователя с компонентами интерфейса. Используемая в Java модель (делегирование обработки событий) основывается на следующих базовых понятиях:

- событие – объект, содержащий сведения о произошедшем событии, являющийся экземпляром потомка класса `java.util.EventObject`;
- «слушатель» – экземпляр класса, реализующий специальный интерфейс слушателя событий;
- источник событий – объект, регистрирующий заинтересованных слушателей и передающий им экземпляры объектов, описывающих произошедшие события (например, кнопка, полоса прокрутки).

Весь цикл обработки событий состоит из следующих этапов:

1. Слушатель событий регистрируется у источника событий (источник включает слушателя во внутренний список заинтересованных слушателей) обращаясь к его методу:

```
eventSourceObject.addEventListener(eventListenerObject);
```

2. При возникновении события источник событий создаёт экземпляр класса соответствующего класса, описывающего произошедшее событие.
3. Источник событий для каждого заинтересованного слушателя вызывает метод обработки события, определённый интерфейсом слушателя, передавая в качестве аргумента экземпляр события.

Большинство компонентов пользовательского интерфейса генерируют событие типа `ActionEvent`, сообщающее о действии пользователя. Объекты, заинтересованные в получении уведомлений о событиях данного типа, должны реализовывать интерфейс `ActionListener`, требующий наличия метода `actionPerformed(ActionEvent event)`.

#### *2.6.1 Компонент «кнопка» (JButton)*

Данный компонент представляет прямоугольную кнопку, снабжённую надписью и (возможно) иконкой. При нажатии на кнопку генерируется событие `ActionEvent`, передаваемое всем заинтересованным слушателям. Конструкторы:

```
// кнопка с надписью
JButton b1 = new JButton("OK");
// кнопка с иконкой
JButton b2 = new JButton(icon);
// кнопка с надписью и иконкой
JButton b3 = new JButton("OK", iconOK);
```

Для создания объекта-иконки на основе графического файла следует применять конструктор `ImageIcon(String filename)`:

```
JButton b4 = new JButton("Cancel", new ImageIcon("button-cancel.gif"));
```

После создания экземпляра кнопки необходимо зарегистрировать слушателя, заинтересованного в уведомлениях о нажатии на ней:

```
myButton1.addActionListener(okButtonActionListener);
```

### *2.6.2 Компонент «радио-кнопка» (JRadioButton)*

Данный компонент представляет круглую кнопку, позволяющую выбрать один (и только один) вариант из целой группы. Когда выбирается другая кнопка, то выделение предыдущей кнопки автоматически пропадает. Обеспечение подобного поведения требует включения ряда радио-кнопок (`JRadioButton`) в группу кнопок (`ButtonGroup`). Именно объект класса `ButtonGroup` и несёт ответственность за «выключение» включенной кнопки при активации новой. При щелчке на радио-кнопку, как и для обычной кнопки, генерируется событие типа `ActionEvent`.

```
ButtonGroup myButtons = new ButtonGroup();
// вариант 1 по умолчанию «включен»
JRadioButton radio1 = new JRadioButton("Вариант 1", true);
myButtons.add(radio1);
// вариант 2 по умолчанию «выключен»
JRadioButton radio2 = new JRadioButton("Вариант 2", false);
myButtons.add(radio2);
```

**Замечание:** добавление радио-кнопок в группу не избавляет от необходимости добавить каждую из них в объект контейнер, в противном случае радио-кнопка не будет отображена. Объект `ButtonGroup` отвечает только за единственность выделения кнопки в группе, но не за её размещение на экране:

```
JPanel myPanel = new JPanel();
myPanel.add(radio1);
myPanel.add(radio2).
myFrame.getContentPane().add(myPanel);
```

Для определения слушателей событий щелчков на радио-кнопках необходимо действовать по аналогии с добавлением слушателей на обычные кнопки:

```
// предполагается, что экземпляры объектов-слушателей уже созданы
radio1.addActionListener(radio1ActionListener);
radio2.addActionListener(radio2ActionListener);
```

### *2.6.3 Компонент «надпись» (JLabel)*

Данный компонент представляет надпись в окне, по умолчанию не имеет визуального декорирования (например, рамки) и не взаимодействует с пользователем. Текст, отображаемый надписью, может содержать HTML-разметку. Для изменения надписи во время работы приложения следует использовать метод `setText(String newText)`.

```
JPanel container = new JPanel();
// создать надпись с выравниванием по правому краю
JLabel label1 = new JLabel("Введите текст:", JLabel.RIGHT);
container.add(label1);
// создать надпись с жирным начертанием
JLabel label2 = new JLabel("<HTML><B>Введите число:</B></HTML>");
Container.add(label2);
```

### *2.6.4 Компонент «текстовое поле ввода» (JTextField)*

Данный компонент представляет прямоугольное поле ввода для одной строки. Для считывания значения поля используется метод `getText()`, для установки – `setText(String newText)`. Метод `setEditable(boolean b)` позволяет задать, разрешено ли пользователю редактировать значение поля.

Добавление поля ввода осуществляется общим для всех компонентов способом – вызовом метода `add()` для контейнера с передачей ссылки на экземпляр кнопки в качестве аргумента:

```
JPanel panel = new JPanel();
JTextField textField = new JTextField("Default input", 20);
panel.add(textField);
```

Вторым аргументом конструктору поля ввода передаётся максимальное число символов, которое может быть введено в данное поле.

## 2.7 Использование анонимных классов

Для уменьшения количества классов в приложении, объёма кода и упрощения понимания логики приложения Java предлагает механизм под названием «анонимные классы», позволяющий разово определить новый класс и создать его экземпляр. Особенно удобной данная возможность

является при определении слушателей событий взаимодействия с компонентами интерфейса. Сравните фрагменты кода:

```
// определение класса-слушателя события
class OkButtonActionListener implements ActionListener {
      private Component parent;
      public OkButtonActionListener(Component parent) {
            this.parent = parent;
      }
      public void actionPerformed(ActionEvent ev) {
            JOptionPane.showMessageDialog(parent,
                  "Нажата кнопка OK",
                  "Событие", JOptionPane.INFORMATION_MESSAGE);
      }
}

class MyFrame extends JFrame {
      public MyFrame() {
            …
            // регистрация слушателя события
            JButton buttonOK = new JButton("OK");
            buttonOK.addActionListener(new OkButtonActionListener(this));
            …
      }
      …
}
```

и

```
class MyFrame extends JFrame {
      public MyFrame() {
            …
            JButton buttonOK = new JButton("OK");
            buttonOK.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent ev) {
                        JOptionPane.showMessageDialog(MyFrame.this,
                        "Нажата кнопка OK",
                        "Событие", JOptionPane.INFORMATION_MESSAGE);
                  }
            });
            …
      }
      …
}
```

В первом фрагменте потребовалось отдельно определять класс-слушатель события нажатия на кнопку «ОК», после чего был создан единственный экземпляр этого класса. Во втором фрагменте класс-слушатель был одновременно определён и передан в качестве параметра в метод регистрации слушателя события.

## 3 Пример приложения

**Задание:** составить программу вычисления значений двух функций двух переменных. Выбор функции осуществляется с помощью радио-кнопок; ввод

значений переменных реализуется с помощью текстовых полей. Расчёт выполняется при нажатии на кнопку «*Вычислить*», результат отображается в текстовом поле «*Результат*». Если вычислить значение функции в заданной точке невозможно (например, в случае логарифма от отрицательного числа), показать сообщение об ошибке вычислений. Кнопка «*Очистить поля*» используется для установки в 0 всех текстовых полей. Главное окно приложения должно иметь заголовок (например, «*Вычисление формулы*»), внешний вид окна представлен на рисунке 3.1.

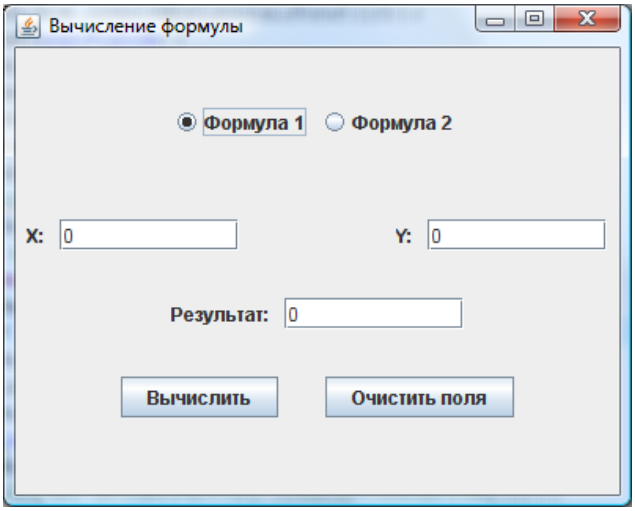


**Рисунок 3.1 – Внешний вид главного окна программы**

### 3.1 Структура приложения и размещение элементов интерфейса

Структура приложения состоит из единственного класса – класса фрейма, являющегося потомком `JFrame` и содержащего элементы графического интерфейса.

Внутренняя структура класса фрейма (допустим, он называется `MainFrame`) включает (рисунок 3.2):

- ряд внутренних полей данных класса;
- конструктор;
- главный метод `main();`
- несколько вспомогательных методов (так называемых методов-помощников).

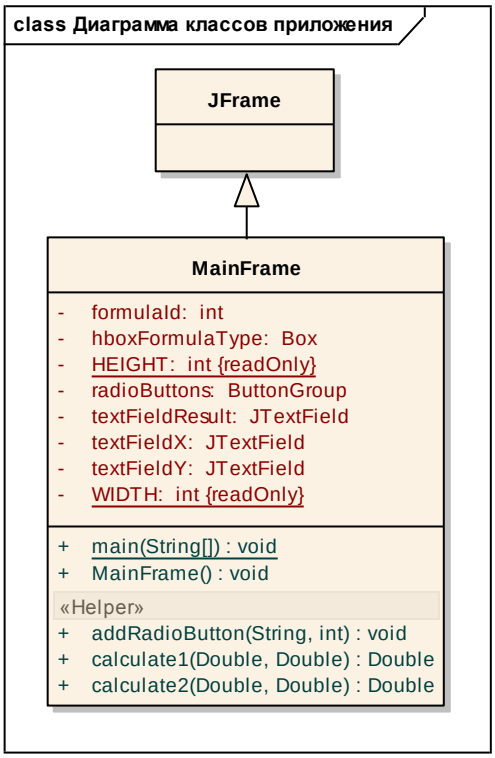


**Рисунок 3.2 – Диаграмма классов приложения**

Задачей главного метода является создание экземпляра окна приложения и его показ на экране, конструктора – создание элементов интерфейса и размещение их в пространстве фрейма, методы-помощники оформляют в виде подпрограмм некоторые многократно выполняемые последовательности инструкций.

Рассмотрим более подробно способ размещения элементов интерфейса в пространстве фрейма. Будем использовать для этого менеджер «коробочной» компоновки. С учётом его принципов расположения компонентов, можно предложить следующую компоновку окна программы (рисунок 3.3):

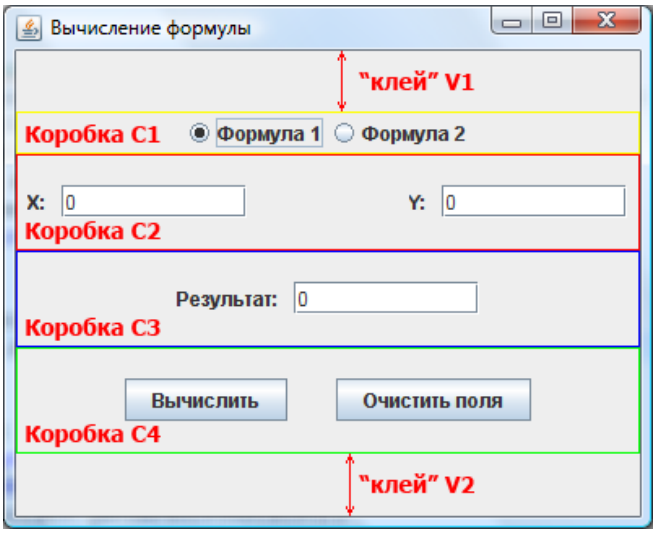


**Рисунок 3.3 – Выделение границ горизонтальных контейнеров в главном окне приложения**

На рисунке 3.3 видно, что компоновкой, используемой во фрейме, является контейнер типа «вертикальная коробка», содержащий четыре независимых контейнера (C1, C2, C3, C4) типа «горизонтальная коробка», в которых размещены элементы управления. Отступы от нижней и верхней границ фрейма обеспечиваются посредством элементов типа «клей» (V1, V2), стремящихся максимально разделить соседние элементы (край фрейма и контейнер). Для компоновки элементов внутри горизонтальных контейнеров применяются элементы типа «клей» и «распорка»: распорки для обеспечения минимального отступа между соседними элементами; клей для максимального возможного удаления соседствующих элементов (рисунок 3.4).

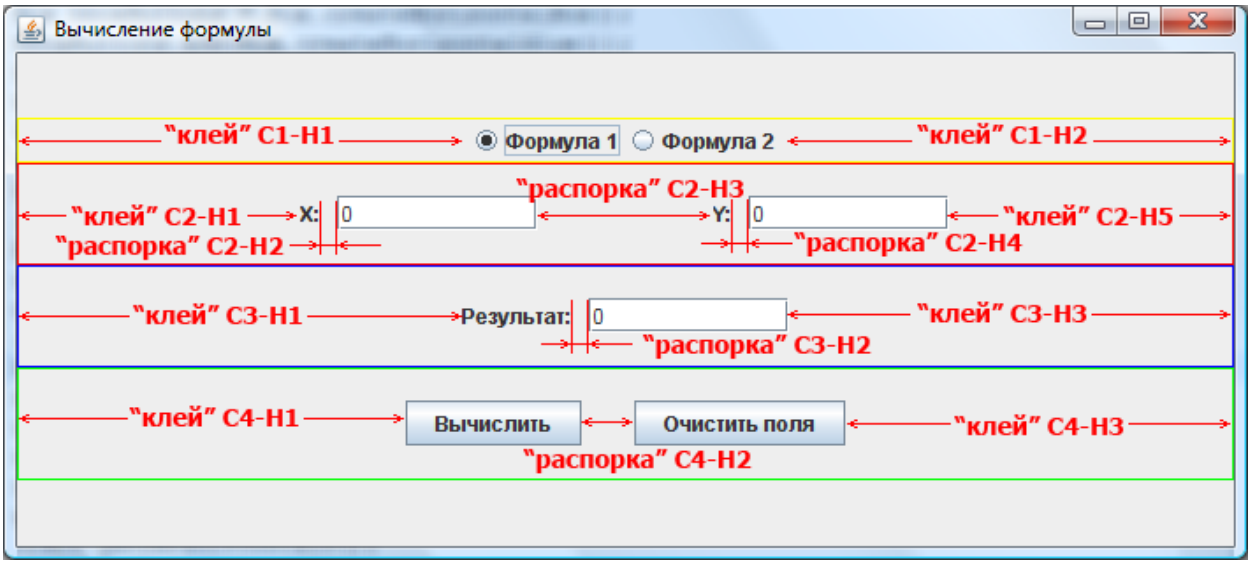


**Рисунок 3.4 – Компоновка элементов в горизонтальных контейнерах**

### 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 Определение полей данных класса

Как мы знаем, внутренними полями данных класса целесообразно делать:

- Константы (определяя их с модификаторами `static final`);
- Объекты, совместно используемые несколькими методами;
- Объекты, предполагающие повторное использование своих существующих экземпляров из-за значительных затрат вычислительных ресурсов при создании.

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 | |
| `JTextField textFieldResult` | Текстовое поле для отображения результата, инициализируется в конструкторе, используется в обработчиках событий |
| `JTextField textFieldX` | Текстовые поля для считывания значений переменных X и Y, инициализируются в конструкторе, используются в обработчиках событий |
| `JTextField textFieldY` | |
| `ButtonGroup radioButtons` | Группа кнопок для обеспечения уникальности выделения радио-кнопки в группе, инициализируется в конструкторе, наполняется в методе-помощнике |
| `Box hboxFormulaType` | Контейнер, в который будут добавляться радио-кнопки, инициализируется в конструкторе, наполняется в методе-помощнике |
| `int formulaId` | Переменная, указывающая, какая из формул является активной в данный момент, совместно используется в обработчиках событий |
| Повторно используемые объекты | |
| Отсутствуют | |

Определим поля класса следующим образом:

```
public class Formula extends JFrame {

    // Размеры окна приложения в виде констант
    private static final int WIDTH = 400;
    private static final int HEIGHT = 320;

    // Текстовые поля для считывания значений переменных X и Y,
    // как компоненты, совместно используемые в различных методах
    private JTextField textFieldX;
    private JTextField textFieldY;

    // Текстовое поле для отображения результата,
```

```
// как компонент, совместно используемый в различных методах
private JTextField textFieldResult;

// Группа радио-кнопок для обеспечения уникальности выделения в группе
private ButtonGroup radioButtons = new ButtonGroup();

// Контейнер для отображения радио-кнопок
private Box hboxFormulaType = Box.createHorizontalBox();

// Идентификатор выбранной формулы
private int formulaId = 1;

...
```

### 3.4 Реализация методов-помощников вычисления значения функций

Вычисление значений функций по заданным формулам удобно оформить в виде независимых методов `calculate1()` и `calculate2()`, получающих два аргумента типа `Double` и возвращающих результат типа `Double`, например:

```
// Формула №1 для рассчёта
public Double calculate1(Double x, Double y) {
	return x*x + y*y;
}

// Формула №2 для рассчёта
public Double calculate2(Double x, Double y) {
	return x*x*x + 1/y;
}
```

### 3.5 Реализация метода-помощника для добавления радио-кнопок

Добавление каждой радио-кнопки включает одинаковый набор этапов:

1. Создание экземпляра радио-кнопки.
2. Объявление и регистрация слушателя события нажатия на радио-кнопку (логика его действия для всех радио-кнопок одинакова, различается лишь идентификатором выбираемой формулы).
3. Добавление радио-кнопки в группу радио-кнопок.
4. Добавление радио-кнопки в контейнер, который будет включен в общую компоновку элементов интерфейса окна.

Все эти действия целесообразно реализовать совместно во вспомогательном методе `addRadioButton()`, вызываемом для добавления каждой радио-кнопки:

```
// buttonName – текст рядом с кнопкой, formulaId – идентификатор формулы
private void addRadioButton(String buttonName, final int formulaId) {
	// Создать экземпляр радио-кнопки с заданным текстом
	JRadioButton button = new JRadioButton(buttonName);
	// Определить и зарегистрировать обработчик
	button.addActionListener(new ActionListener() {
		public void actionPerformed(ActionEvent ev) {
```

```
                    // Который будет устанавливать идентификатор выбранной
                    // формулы в классе Formula равным formulaId
                    Formula.this.formulaId = formulaId;
                }
            });
            // Добавить радио-кнопку в группу
            radioButtons.add(button);
            // Добавить радио-кнопку в контейнер
            // Для этого ссылка на контейнер сделана полем данных класса
            hboxFormulaType.add(button);
        }
```

### 3.6 Инициализация окна приложения

Расположение элементов в окне приложения определяется при его инициализации (в конструкторе класса). Инициализация окна приложения состоит из нескольких этапов:

1. Вызов конструктора предка (в частности, для задания заголовка окна).
2. Установка размеров и положения окна.
3. Добавление полей ввода для переменных.
4. Добавление области для вывода результатов.
5. Добавление кнопок «*Вычислить*» и «*Очистить*».
6. Сборка всех элементов воедино.

#### *3.6.1 Вызов конструктора предка*

Для обращения к конструктору унаследованного класса в Java используется ключевое слово `super`, после чего в скобках передаются аргументы для конструктора. Будем использовать версию конструктора `JFrame`, использующего только заголовок окна:

```
super("Вычисление формулы");
```

#### *3.6.2 Установка размеров и положения окна*

Задание размера и позиционирование окна осуществляется с помощью методов `setSize()`, `setLocation()`, а также экземпляра инструментария (`Toolkit`):

```
setSize(WIDTH, HEIGHT);
Toolkit kit = Toolkit.getDefaultToolkit();
// Центрировать окно приложения на экране
setLocation((kit.getScreenSize().width - WIDTH)/2,
            (kit.getScreenSize().height - HEIGHT)/2);
```

### *3.6.3 Добавление радио-кнопок выбора формулы*

С использованием метода-помощника `addRadioButton()` создание радио-кнопок и включение их в компоновку окна (обозначение элементов компоновки показано на рисунке 3.3) выполняется следующим образом:

```
        // Добавить «клей» C1-H1 с левой стороны
        hboxFormulaType.add(Box.createHorizontalGlue());
        // Создать радио-кнопку для формулы 1
        addRadioButton("Формула 1", 1);
        // Создать радио-кнопку для формулы 2
        addRadioButton("Формула 2", 2);
        // Установить выделенной 1-ую кнопку из группы
        radioButtons.setSelected(radioButtons.getElements().nextElement().getMod
el(), true);
        // Добавить «клей» C1-H2 с правой стороны
        hboxFormulaType.add(Box.createHorizontalGlue());
        // Задать рамку для коробки с помощью класса BorderFactory
        hboxVariables.setBorder(BorderFactory.createLineBorder(Color.YELLOW));
```

### *3.6.4 Добавление текстовых полей для переменных*

Добавление текстовых полей в окно приложения включает создание их экземпляров и включение в компоновку окна:

```
        // Создать подпись “X:” для переменной X
        JLabel labelForX = new JLabel("X:");
        // Создать текстовое поле для ввода значения переменной X,
        // (по умолчанию 0)
        textFieldX = new JTextField("0", 10);
        // Установить макс размер = желаемому для предотвращения масштабирования
        textFieldX.setMaximumSize(textFieldX.getPreferredSize());
        // Создать подпись “Y:” для переменной Y
        JLabel labelForY = new JLabel("Y:");
        // Создать текстовое поле для ввода значения переменной Y,
        // (по умолчанию 0)
        textFieldY = new JTextField("0", 10);
        // Установить макс размер = желаемому для предотвращения масштабирования
        textFieldY.setMaximumSize(textFieldY.getPreferredSize());
        // Создать контейнер «коробка с горизонтальной укладкой»
        Box hboxVariables = Box.createHorizontalBox();
        // Задать рамку для коробки с помощью класса BorderFactory
        hboxVariables.setBorder(BorderFactory.createLineBorder(Color.RED));
        // Добавить в контейнер ряд объектов:
        // Добавить «клей» C2-H1 – для максимального удаления от левого края
        hboxVariables.add(Box.createHorizontalGlue());
        // Добавить подпись для переменной X
        hboxVariables.add(labelForX);
        // Добавить «распорку» C2-H2 шириной 10 пикселов для отступа между
        // надписью и текстовым полем для ввода значения X
        hboxVariables.add(Box.createHorizontalStrut(10));
        // Добавить само текстовое поле для ввода X
        hboxVariables.add(textFieldX);
        // Добавить «распорку» C2-H3 шириной 100 пикселов для отступа между
        // текстовым полем для ввода X и подписью для Y
        hboxVariables.add(Box.createHorizontalStrut(100));
        // Добавить подпись для переменной Y
        hboxVariables.add(labelForY);
        // Добавить «распорку» C2-H4 шириной 10 пикселов для отступа между
```

```
        // надписью и текстовым полем для ввода значения Y
        hboxVariables.add(Box.createHorizontalStrut(10));
        // Добавить само текстовое поле для ввода Y
        hboxVariables.add(textFieldY);
        // Добавить «клей» С2-Н5 для максимального удаления от правого края
        hboxVariables.add(Box.createHorizontalGlue());
```

### 3.6.5 Добавление области для вывода результатов

Результат вычислений будет выводиться в текстовое поле, которое будет располагаться в отдельном контейнере типа «коробка»:

```
        // Создать подпись для поля с результатом
        JLabel labelForResult = new JLabel("Результат:");
        // Создать текстовое поле для вывода результата, начальное значение - 0
        textFieldResult = new JTextField("0", 10);
        // Создать контейнер «коробка с горизонтальной укладкой»
        Box hboxResult = Box.createHorizontalBox();
        // Добавить в контейнер ряд объектов
        // Добавить «клей» С3-Н1 для отступа от левого края
        hboxResult.add(Box.createHorizontalGlue());
        // Добавить подпись для результата
        hboxResult.add(labelForResult);
        // Добавить «распорку» С3-Н2 в 10 пикселов между подписью и полем
        // результата
        hboxResult.add(Box.createHorizontalStrut(10));
        // Добавить текстовое поле для вывода результата
        hboxResult.add(labelResult);
        // Добавить «клей» С3-Н3 справа
        hboxResult.add(Box.createHorizontalGlue());
        // Задать рамку для контейнера
        hboxResult.setBorder(BorderFactory.createLineBorder(Color.BLUE));
```

### 3.6.6 Создание кнопок

Добавление кнопок в окно предполагает создание экземпляров кнопок, задание обработчиков нажатий на них (с использованием анонимных классов) и включение кнопок в компоновку:

```
        // Создать кнопку «Вычислить»
        JButton buttonCalc = new JButton("Вычислить");
        // Определить и зарегистрировать обработчик нажатия на кнопку
        buttonCalc.addActionListener(new ActionListener() {
              public void actionPerformed(ActionEvent ev) {
// Преобразование введенных строк в числа с плавающей точкой может
// спровоцировать исключительную ситуацию при неправильном формате чисел,
// поэтому необходим блок try-catch
                    try {
                          // Получить значение X
                          Double x = Double.parseDouble(textFieldX.getText());
                          // Получить значение Y
                          Double y = Double.parseDouble(textFieldY.getText());
                          // Вычислить результат
                          if (formulaId==1)
                                result = calculate1(x, y);
                          else
                                result = calculate2(x, y);
```

```
                        // Установить текст надписи равным результату
                        labelResult.setText(result.toString());
                    } catch (NumberFormatException ex) {
                        // В случае исключительной ситуации показать сообщение
                        JOptionPane.showMessageDialog(Formula.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");
            // Определить и зарегистрировать обработчик нажатия на кнопку
            buttonReset.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent ev) {
                    textFieldX.setText("0");
                    textFieldY.setText("0");
                    textFieldResult.setText("0");
                }
            });
            // Создать коробку с горизонтальной укладкой
            Box hboxButtons = Box.createHorizontalBox();
            // Добавить «клей» C4-H1 с левой стороны
            hboxButtons.add(Box.createHorizontalGlue());
            // Добавить кнопку «Вычислить» в компоновку
            hboxButtons.add(buttonCalc);
            // Добавить распорку в 30 пикселов C4-H2 между кнопками
            hboxButtons.add(Box.createHorizontalStrut(30));
            // Добавить кнопку «Очистить поля» в компоновку
            hboxButtons.add(buttonReset);
            // Добавить «клей» C4-H3 с правой стороны
            hboxButtons.add(Box.createHorizontalGlue());
            // Задать рамку для контейнера
            hboxButtons.setBorder(BorderFactory.createLineBorder(Color.GREEN));
```

### *3.6.7 Сборка панелей окна*

Заключительным этапом компоновки окна является сборка независимых контейнеров (с радио-кнопками выбора формулы, текстовыми полями ввода, текстовым полем вывода результата и кнопками) в единую компоновку «коробка с вертикальной укладкой»:

```
            // Создать контейнер «коробка с вертикальной укладкой»
            Box contentBox = Box.createVerticalBox();
            // Добавить «клей» V1 сверху
            contentBox.add(Box.createVerticalGlue());
            // Добавить контейнер с выбором формулы
            contentBox.add(hboxFormulaType);
            // Добавить контейнер с переменными
            contentBox.add(hboxVariables);
            // Добавить контейнер с результатом вычислений
            contentBox.add(hboxResult);
            // Добавить контейнер с кнопками
            contentBox.add(hboxButtons);
            // Добавить «клей» V2 снизу
            contentBox.add(Box.createVerticalGlue());
            // Установить «вертикальную коробку» в область содержания главного окна
            getContentPane().add(contentBox, BorderLayout.CENTER);
```

### 3.7 Реализация главного метода main()

Метод `main()` главного класса приложения в данном случае является очень простым. Он включает: создание экземпляра окна, задание действия по закрытию окна приложения, показ окна.

```
public static void main(String[] args) {
	MainFrame frame = new MainFrame();
	frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
	frame.setVisible(true);
}
```

## 4 Задания

### 4.1 Вариант А

а) Реализовать (по вариантам) вычисление функций трёх переменных.

б) Добавить в приложение кнопки «*MC*» и «*M+*», предоставляющие возможность суммирования с накоплением результата во внутренней переменной `sum` (типа `Double`) класса `Formula`. Нажатие на «*MC*» очищает значение внутренней суммы, нажатие на «*M+*» приводит к суммированию результата текущего вычисления с текущим содержимым переменной `sum` и показе суммы в области вывода результата.

| № п/п | Формулы |
|---|---|
| 1 | Формула №1: $F(x,y,z)=(\cos e^{x}+\ln(1+y)^{2}+\sqrt{e^{\cos x}+\sin^{2}\pi z}+\sqrt{1/x}+\cos y^{2})^{\sin z}$ Формула №2: $F(x,y,z)=\frac{\sqrt[y]{1+x^{2}}}{e^{\sin(z)+x}}$ |
| 2 | Формула №1: $F(x,y,z)=\frac{1/\sqrt{x}+\cos e^{y}+\cos z^{2}}{\sqrt[3]{\ln(1+z)^{2}+\sqrt{e^{\cos y}+\sin^{2}\pi x}}}$ Формула №2: $F(x,y,z)=\sqrt[z]{y+x^{3}}/\ln z$ |
| 3 | Формула №1: $F(x,y,z)=\frac{(\sin\pi y^{2}+\ln y^{2})}{\sin\pi z^{2}+\sin x+\ln z^{2}+x^{2}+e^{\cos zx}}$ Формула №2: $F(x,y,z)=y\cdot\frac{x^{2}}{\lg(z^{y})+\cos^{2}(\sqrt[3]{x})}$ |
| 4 | Формула №1: |

| | |
|---|---|
| | $F(x,y,z)=\sqrt{(\sin y+y^2+e^{\cos y})^2+(\ln z^2+\sin \pi x^2)^3}$<br>Формула №2:<br>$F(x,y,z)=\sqrt{y}\cdot\frac{3\cdot z^x}{\sqrt{1+y^3}}$ |
| 5 | Формула №1:<br>$F(x,y,z)=(\ln(1+x)^2+\cos \pi z^3)^{\sin y}+(e^{x^2}+\cos e^z+\sqrt{1/y})^{1/x}$<br>Формула №2:<br>$F(x,y,z)=x\cdot\frac{\cos^3(y^2)}{\sqrt[x]{z}}$ |
| 6 | Формула №1:<br>$F(x,y,z)=\frac{\sqrt[4]{\cos e^y+e^{y^2}+\sqrt{1/x}}}{(\cos \pi z^3+\ln(1+z)^2)^{\sin y}}$<br>Формула №2:<br>$F(x,y,z)=\frac{1+x^z+\ln y^2}{\sqrt{x^3+1}}(1-\sin(y\cdot z))$ |
| 7 | Формула №1:<br>$F(x,y,z)=\frac{\sqrt[4]{\ln(1+z)^2+\cos \pi y^3}}{(\cos e^x+\sqrt{1/x}+e^{x^2})^{\sin x}}$<br>Формула №2:<br>$F(x,y,z)=\frac{\sin^2(z^y)}{\sqrt{1+x^3}}$ |
| 8 | Формула №1:<br>$F(x,y,z)=\sqrt[4]{\cos \pi x^3+\ln(1+y)^2}\,(e^{z^2}+\sqrt{1/x}+\cos e^y)$<br>Формула №2:<br>$F(x,y,z)=\frac{x^x}{\sqrt{y^3+1}+\ln z}$ |
| 9 | Формула №1:<br>$F(x,y,z)=\sin(\sin y+e^{\cos y}+z^2)\sqrt[4]{\sin \pi y^2+\ln x^2}$<br>Формула №2:<br>$F(x,y,z)=\frac{\operatorname{arctg}(\sqrt[x]{z})}{y^2+z\cdot\sin(\ln x)}$ |
| 10 | Формула №1:<br>$F(x,y,z)=\sin(\ln y+\sin \pi y^2)\sqrt[4]{x^2+\sin z+e^{\cos z}}$<br>Формула №2: |

| | |
|---|---|
| | $F(x,y,z)=\dfrac{e^{0,5\cdot x}}{\sqrt{z+y}\cdot \ln x^z}$ |
| 11 | Формула №1:<br>$F(x,y,z)=\dfrac{\sqrt[4]{\ln z+\sin \pi z^2}}{(y^2+e^{\cos x}+\sin y)^{\sin x}}$<br>Формула №2:<br>$F(x,y,z)=\dfrac{1+\sqrt{z\cdot x}}{\sqrt[y]{1+x^3}}$ |
| 12 | Формула №1:<br>$F(x,y,z)=(\sin y+y^2+e^{\cos y})\sqrt[4]{\ln z+\sin \pi x^2}$<br>Формула №2:<br>$F(x,y,z)=\dfrac{\mathrm{tg}(x^2)+\sqrt{y}}{z\cdot \lg(x+y)}$ |

### 4.2 Вариант В

а) Реализовать (по вариантам) вычисление значений двух функций трёх переменных. Выбор функции осуществляется с помощью радио-кнопок.

б) Добавить в приложение внутреннюю память из трёх переменных `mem1`, `mem2`, `mem3` типа `Double`. Кнопки «*MC*» и «*M+*» работают с выбранной переменной и предоставляют возможность суммирования с накоплением результата. Нажатие на «*MC*» очищает значение активной переменной, нажатие на «*M+*» приводит к суммированию результата текущего вычисления с текущим значением активной переменной и показе суммы в текстовом поле вывода результата.

в) Для переключения активной переменной добавить в приложение область с тремя радио-кнопками: «Переменная 1», «Переменная 2», «Переменная 3». Щелчок на радио-кнопке активирует соответствующую переменную, последующие нажатия на кнопки «*MC*» и «*M+*» изменяют только её содержимое, значения других переменных остаются неизменными.

| № п/п | Формулы |
|---|---|
| 1 | Формула №1:<br>$F(x,y,z)=(\sin y+y^2+e^{\cos y})\sqrt[4]{\ln z+\sin \pi x^2}$<br>Формула №2:<br>$F(x,y,z)=\sqrt[z]{y+x^3}\,/\ln z$ |
| 2 | Формула №1: |

| | |
|---|---|
| | $F(x,y,z)=\frac{\sqrt[4]{\ln z+\sin \pi z^2}}{(y^2+e^{\cos x}+\sin y)^{\sin x}}$<br>Формула №2:<br>$F(x,y,z)=\sqrt{y}\cdot\frac{3\cdot z^x}{\sqrt{1+y^3}}$ |
| 3 | Формула №1:<br>$F(x,y,z)=\sin(\ln y+\sin \pi y^2)\sqrt[4]{x^2+\sin z+e^{\cos z}}$<br>Формула №2:<br>$F(x,y,z)=\frac{1+x^z+\ln y^2}{\sqrt{x^3+1}}(1-\sin(y\cdot z))$ |
| 4 | Формула №1:<br>$F(x,y,z)=\sin(\sin y+e^{\cos y}+z^2)\sqrt[4]{\sin \pi y^2+\ln x^2}$<br>Формула №2:<br>$F(x,y,z)=\frac{x^x}{\sqrt{y^3+1}+\ln z}$ |
| 5 | Формула №1:<br>$F(x,y,z)=\sqrt[4]{\cos \pi x^3+\ln(1+y)^2}\,(e^{z^2}+\sqrt{1/x}+\cos e^y)$<br>Формула №2:<br>$F(x,y,z)=\frac{e^{0,5\cdot x}}{\sqrt{z+y}\cdot\ln x^z}$ |
| 6 | Формула №1:<br>$F(x,y,z)=\frac{\sqrt[4]{\ln(1+z)^2+\cos \pi y^3}}{(\cos e^x+\sqrt{1/x}+e^{x^2})^{\sin x}}$<br>Формула №2:<br>$F(x,y,z)=\frac{\mathrm{tg}(x^2)+\sqrt{y}}{z\cdot\lg(x+y)}$ |
| 7 | Формула №1:<br>$F(x,y,z)=\frac{\sqrt[4]{\cos e^y+e^{y^2}+\sqrt{1/x}}}{(\cos \pi z^3+\ln(1+z)^2)^{\sin y}}$<br>Формула №2:<br>$F(x,y,z)=\frac{\sqrt[y]{1+x^2}}{e^{\sin(z)+x}}$ |
| 8 | Формула №1:<br>$F(x,y,z)=(\ln(1+x)^2+\cos \pi z^3)^{\sin y}+(e^{x^2}+\cos e^z+\sqrt{1/y})^{1/x}$<br>Формула №2: |

| | |
|---|---|
| | $F(x,y,z)=y\cdot\frac{x^2}{\lg(z^y)+\cos^2(\sqrt[3]{x})}$ |
| 9 | Формула №1:<br>$F(x,y,z)=\sqrt{(\sin y+y^2+e^{\cos y})^2+(\ln z^2+\sin \pi x^2)^3}$<br>Формула №2:<br>$F(x,y,z)=x\cdot\frac{\cos^3(y^2)}{\sqrt[x]{z}}$ |
| 10 | Формула №1:<br>$F(x,y,z)=\frac{(\sin \pi y^2+\ln y^2)}{\sin \pi z^2+\sin x+\ln z^2+x^2+e^{\cos zx}}$<br>Формула №2:<br>$F(x,y,z)=\frac{\sin^2(z^y)}{\sqrt{1+x^3}}$ |
| 11 | Формула №1:<br>$F(x,y,z)=\frac{1/\sqrt{x}+\cos e^y+\cos z^2}{\sqrt[3]{\ln(1+z)^2+\sqrt{e^{\cos y}+\sin^2 \pi x}}}$<br>Формула №2:<br>$F(x,y,z)=\frac{\operatorname{arctg}(\sqrt[x]{z})}{y^2+z\cdot\sin(\ln x)}$ |
| 12 | Формула №1:<br>$F(x,y,z)=(\cos e^x+\ln(1+y)^2+\sqrt{e^{\cos x}+\sin^2 \pi z}+\sqrt{1/x}+\cos y^2)^{\sin z}$<br>Формула №2:<br>$F(x,y,z)=\frac{1+\sqrt{z\cdot x}}{\sqrt[y]{1+x^3}}$ |

### 4.3 Вариант С

а) Реализовать (по вариантам) вычисление значений двух функций трёх переменных. Выбор функции осуществляется с помощью радио-кнопок.

б) Добавить в окно область для просмотра внешнего вида активной формулы. При переключении активной формулы изображение в области должно изменяться. Внешний вид формул брать из графических файлов в формате BMP, полученных из слепка экрана документа Word.

в) Добавить в приложение внутреннюю память из трёх переменных `mem1`, `mem2`, `mem3` типа `Double`. Кнопки «*MC*» и «*M+*» работают с выбранной переменной и предоставляют возможность суммирования с накоплением результата. Нажатие на «*MC*» очищает значение активной переменной, нажатие на «*M+*» приводит к суммированию результата текущего

вычисления с текущим значением активной переменной и показе суммы в текстовом поле вывода результата.

г) Для переключения активной переменной добавить в приложение область с тремя радио-кнопками: «Переменная 1», «Переменная 2», «Переменная 3». Щелчок на радио-кнопке активирует соответствующую переменную, последующие нажатия на кнопки «*MC*» и «*M+*» изменяют только её содержимое, значения других переменных остаются неизменными.

д) Добавить в окно отдельную панель, показывающую текущие значения ячеек памяти.

| № п/п | Формула 1 |
|---|---|
| 1 | Формула №1: $F(x,y,z)=\sin(\ln y+\sin \pi y^2)\sqrt[4]{x^2+\sin z+e^{\cos z}}$ Формула №2: $F(x,y,z)=(\cos e^x+\ln(1+y)^2+\sqrt{e^{\cos x}+\sin^2 \pi z}+\sqrt{1/x}+\cos y^2)^{\sin z}$ |
| 2 | Формула №1: $F(x,y,z)=(\ln(1+x)^2+\cos \pi z^3)^{\sin y}+(e^{x^2}+\cos e^z+\sqrt{1/y})^{1/x}$ Формула №2: $F(x,y,z)=\sqrt[4]{\cos \pi x^3+\ln(1+y)^2}\,(e^{z^2}+\sqrt{1/x}+\cos e^y)$ |
| 3 | Формула №1: $F(x,y,z)=\dfrac{(\sin \pi y^2+\ln y^2)}{\sin \pi z^2+\sin x+\ln z^2+x^2+e^{\cos zx}}$ Формула №1: $F(x,y,z)=\dfrac{\sqrt[4]{\cos e^y+e^{y^2}+\sqrt{1/x}}}{(\cos \pi z^3+\ln(1+z)^2)^{\sin y}}$ |

# Приложение 1. Исходный код приложения

```
package bsu.rfe.java.group7.lab2.Ivanov.varB4;

// Импортируются классы, используемые в приложении
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Image;
import java.awt.Toolkit;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.BorderFactory;
import javax.swing.Box;
import javax.swing.ButtonGroup;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JRadioButton;
import javax.swing.JTextField;

@SuppressWarnings("serial")
// Главный класс приложения, он же класс фрейма
public class MainFrame extends JFrame {

	// Размеры окна приложения в виде констант
	private static final int WIDTH = 400;
	private static final int HEIGHT = 320;

	// Текстовые поля для считывания значений переменных,
	// как компоненты, совместно используемые в различных методах
	private JTextField textFieldX;
	private JTextField textFieldY;

	// Текстовое поле для отображения результата,
	// как компонент, совместно используемый в различных методах
	private JTextField textFieldResult;

	// Группа радио-кнопок для обеспечения уникальности выделения в группе
	private ButtonGroup radioButtons = new ButtonGroup();

	// Контейнер для отображения радио-кнопок
	private Box hboxFormulaType = Box.createHorizontalBox();

	private int formulaId = 1;

	// Формула №1 для рассчёта
	public Double calculate1(Double x, Double y) {
		return x*x + y*y;
	}

	// Формула №2 для рассчёта
	public Double calculate2(Double x, Double y) {
		return x*x*x + 1/y;
	}

	// Вспомогательный метод для добавления кнопок на панель
	private void addRadioButton(String buttonName, final int formulaId) {
		JRadioButton button = new JRadioButton(buttonName);
```

```
            button.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent ev) {
                        MainFrame.this.formulaId = formulaId;
                  }
            });
            radioButtons.add(button);
            hboxFormulaType.add(button);
      }

      // Конструктор класса
      public MainFrame() {
            super("Вычисление формулы");
            setSize(WIDTH, HEIGHT);
            Toolkit kit = Toolkit.getDefaultToolkit();
            // Отцентрировать окно приложения на экране
            setLocation((kit.getScreenSize().width - WIDTH)/2,
(kit.getScreenSize().height - HEIGHT)/2);

            hboxFormulaType.add(Box.createHorizontalGlue());
            addRadioButton("Формула 1", 1);
            addRadioButton("Формула 2", 2);
            radioButtons.setSelected(
                  radioButtons.getElements().nextElement().getModel(), true);
            hboxFormulaType.add(Box.createHorizontalGlue());
            hboxFormulaType.setBorder(
                  BorderFactory.createLineBorder(Color.YELLOW));
            // Создать область с полями ввода для X и Y
            JLabel labelForX = new JLabel("X:");
            textFieldX = new JTextField("0", 10);
            textFieldX.setMaximumSize(textFieldX.getPreferredSize());
            JLabel labelForY = new JLabel("Y:");
            textFieldY = new JTextField("0", 10);
            textFieldY.setMaximumSize(textFieldY.getPreferredSize());
            Box hboxVariables = Box.createHorizontalBox();
            hboxVariables.setBorder(
                  BorderFactory.createLineBorder(Color.RED));
            hboxVariables.add(Box.createHorizontalGlue());
            hboxVariables.add(labelForX);
            hboxVariables.add(Box.createHorizontalStrut(10));
            hboxVariables.add(textFieldX);
            hboxVariables.add(Box.createHorizontalStrut(100));
            hboxVariables.add(labelForY);
            hboxVariables.add(Box.createHorizontalStrut(10));
            hboxVariables.add(textFieldY);
            hboxVariables.add(Box.createHorizontalGlue());
            // Создать область для вывода результата
            JLabel labelForResult = new JLabel("Результат:");
            //labelResult = new JLabel("0");
            textFieldResult = new JTextField("0", 10);
            textFieldResult.setMaximumSize(
                  textFieldResult.getPreferredSize());
            Box hboxResult = Box.createHorizontalBox();
            hboxResult.add(Box.createHorizontalGlue());
            hboxResult.add(labelForResult);
            hboxResult.add(Box.createHorizontalStrut(10));
            hboxResult.add(textFieldResult);
            hboxResult.add(Box.createHorizontalGlue());
            hboxResult.setBorder(BorderFactory.createLineBorder(Color.BLUE));
            // Создать область для кнопок
            JButton buttonCalc = new JButton("Вычислить");
            buttonCalc.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent ev) {
                        try {
                          Double x = Double.parseDouble(textFieldX.getText());
```

```
                        Double y = Double.parseDouble(textFieldY.getText());
                        Double result;
                        if (formulaId==1)
                            result = calculate1(x, y);
                        else
                            result = calculate2(x, y);
                        textFieldResult.setText(result.toString());
                    } catch (NumberFormatException ex) {
                        JOptionPane.showMessageDialog(MainFrame.this,
"Ошибка в формате записи числа с плавающей точкой", "Ошибочный формат числа",
JOptionPane.WARNING_MESSAGE);
                    }
                }
            });
            JButton buttonReset = new JButton("Очистить поля");
            buttonReset.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent ev) {
                    textFieldX.setText("0");
                    textFieldY.setText("0");
                    textFieldResult.setText("0");

                }
            });
            Box hboxButtons = Box.createHorizontalBox();
            hboxButtons.add(Box.createHorizontalGlue());
            hboxButtons.add(buttonCalc);
            hboxButtons.add(Box.createHorizontalStrut(30));
            hboxButtons.add(buttonReset);
            hboxButtons.add(Box.createHorizontalGlue());
            hboxButtons.setBorder(
                    BorderFactory.createLineBorder(Color.GREEN));
            // Связать области воедино в компоновке BoxLayout
            Box contentBox = Box.createVerticalBox();
            contentBox.add(Box.createVerticalGlue());
            contentBox.add(hboxFormulaType);
            contentBox.add(hboxVariables);
            contentBox.add(hboxResult);
            contentBox.add(hboxButtons);
            contentBox.add(Box.createVerticalGlue());
            getContentPane().add(contentBox, BorderLayout.CENTER);
        }

        // Главный метод класса
        public static void main(String[] args) {
            MainFrame frame = new MainFrame();
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setVisible(true);
        }

}
```