ЛАБОРАТОРНАЯ РАБОТА №8. ГРАФИЧЕСКИЙ ИНТЕРФЕЙС ПОЛЬЗОВАТЕЛЯ. ОСНОВЫ ОБРАБОТКИСОБЫТИЙ

Цель и содержание

Цель лабораторной работы: научиться разрабатывать приложения с графическим интерфейсом на языке Java; научиться программировать с использованием технологии WPF.

Задачи лабораторной работы:

– изучить возможности библиотек AWT и Swing;

– изучить архитектуру оконного приложения WPF и Swing;

– научиться проектировать интерфейс приложения с использованием простых элементов управления Swing и WPF;

– научиться использовать контейнеры компоновки в рамках технологий Swing и WPF;

– изучить сходства и различия технологий построения приложений с графическим пользовательским интерфейсом в Java и .NET Framework;

– научиться разрабатывать графические приложения для решения учебных задач.

Теоретическая часть

Основные технологии и библиотеки Java SE.

Объектно-ориентированное программирование как технология претерпевала развитие параллельно с развитием технологий взаимодействия компьютеров с пользователями. Появление графического интерфейса пользователя (ГИП) существенно усложнило разработку прикладного ПО и вывело на первый план именно ООП-парадигму, которая наиболее полно учитывает потребности разработчиков графических приложений.

В большинстве высокоуровневых языков разработки приложений с ГИП реализованы библиотеки для построения интерфейса пользователя. В языке C# используются технологии Windows Forms и Windows Presentation Foundation. В Java существуют библиотеки AWT и Swing.

Стандартная поставка Java SE включает в себя библиотеку классов, обеспечивающих создание графического интерфейса пользователя (ГИП) (Graphical User Interface, GUI). Эта графическая библиотека получила название JFC (Java Foundation Classes). В библиотеке JFC можно выделить шесть основных частей:

1) AWT (Abstract Window Toolkit) – базовая библиотека классов с несколько устарелым набором «тяжелых» (heavyweight) графических компонентов, расположенная в пакете java.awt и его подпакетах.

2) Swing – библиотека «легких» (lightweight) графических компонентов, дополняющая и во многом заменяющая библиотеку AWT. Занимает почти двадцать пакетов с префиксом javax.swing;

3) Java 2D – часть библиотеки AWT, обеспечивающая рисование графики, выбор цвета, вывод изображений и фигурного текста, а также преобразование их перед выводом.

4) DnD (Drag and Drop) – библиотека классов, позволяющих перемещать объекты из одного компонента в другой с помощью буфера обмена (clipboard). Классы этой библиотеки помещены в пакеты java.awt.datatransfer и java.awt.dnd;

5) Input Method Framework – классы для создания новых методов ввода/вывода. Они занимают пакеты java.awt.im и java.awt.im.spi;

6) Accessibility – библиотека классов для взаимодействия с нестандартными устройствами ввода/вывода: клавиатурой Брайля, световым пером и др. Она расположена в пакете javax.accessibility.

Интерфейс программирования графики (Java Graphics API) представлен двумя технологиями в Java: AWT и Swing.

Возможности технологии AWT разнообразны – AWT состоит из 12 пакетов. В большинстве приложений используются пакеты java.awt и java.awt.event. Это наиболее часто используемые пакеты.

Пакета java.awt содержит:

1) Классы, представляющие компоненты ГИП (GUI Component Classes). Эти классы представляют такие элементы как кнопки (Button), текстовые поля (TextField), надписи (Label) и пр.

2) Классы, представляющие контейнеры ГИП (GUI Container Classes). Представляют такие элементы как панель (Panel), фрейм (Frame), диалог (Dialog) и пр.

3) Менеджеры расположения элементов (Layout managers). Это элементы FlowLayout, BorderLayout и GridLayout.

4) Вспомогательные классы и графические классы общего назначения (Custom graphics classes). Например классы для представления цвета (Color), шрифтов (Font).

Пакет java.awt.event, как видно из названия, содержит классы для работы с системой событий и содержит:

1) Классы событий (Event classes): ActionEvent, MouseEvent, KeyEvent и WindowEvent.

2) Интерфейсы слушателей событий (Event Listener Interfaces): ActionListener, MouseListener, KeyListener и WindowListener.

3) Классы адаптеров слушателей событий (Event Listener Adapter classes): MouseAdapter, KeyAdapter и WindowAdapter.

AWT обеспечивает интерфейс для разработки графических приложений, независимый от платформы и устройства. Эти приложения запускаются на всех ОС.

В AWT выделяют два типа GUI элементов: компоненты (Component) и контейнеры (Container). На рис. 1 представлена диаграмма классов AWT, на которой разделены эти две группы элементов.

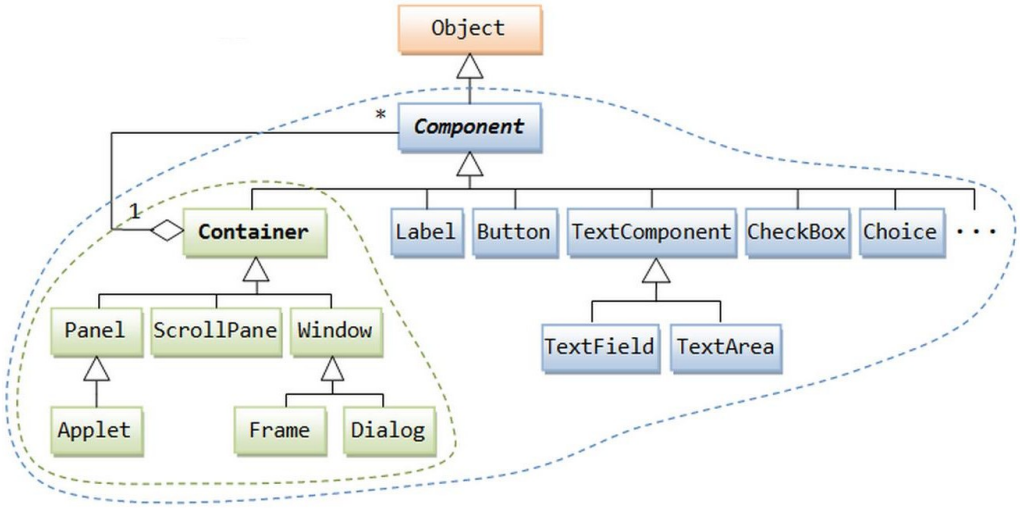

Рисунок 1 – Диаграмма классов AWT.

Понимание принципов функционирования библиотеки AWT необходимо для программирования на Java с использованием любой другой библиотеки ГИП, включая Swing. Но детальное описание AWT выходит за рамки данного курса. В данной лабораторной работе акцент сделан на разработке с использованием Swing.

Технология разработки ГИП Swing основана на AWT. На рис. 2 представлена диаграмма классов, отражающая преемственность двух технологий.

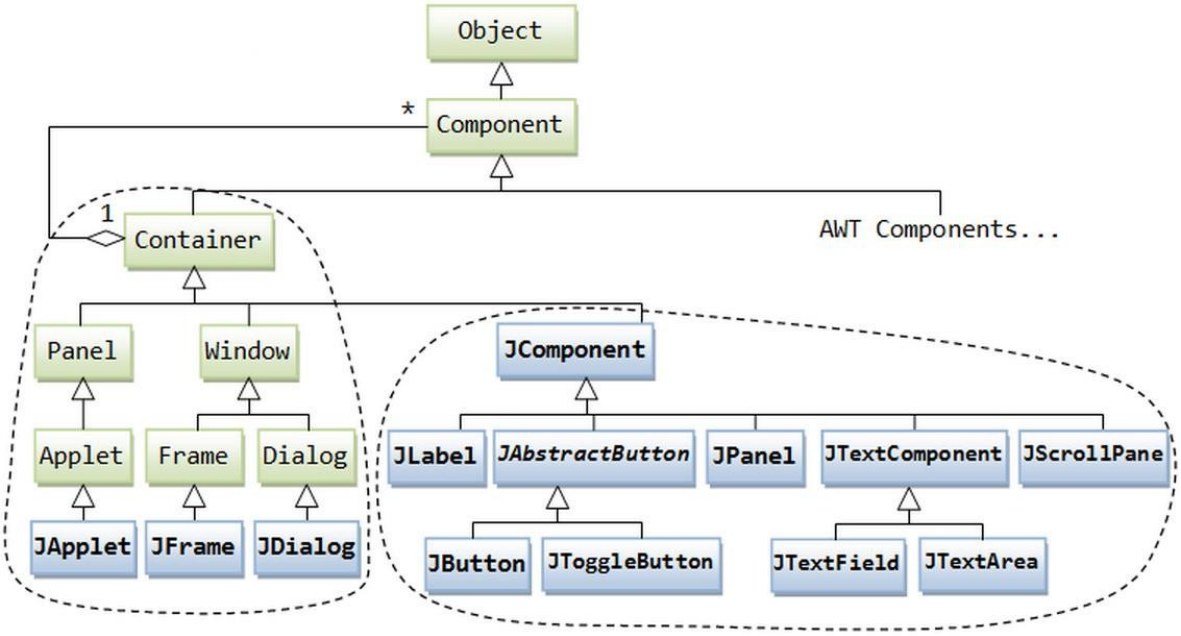

Рисунок 2 – Связь AWT и Swing.

Особенности библиотеки Swing:

1) Swing написан на «чистом» Java (pure Java).

2) Компоненты Swing являются «легковесными». Тогда как компоненты AWT – «тяжеловесные». Это означает, что компоненты Swing вообще не нуждаются в поддержке операционной системы и поэтому гораздо более стабильны и быстры.

3) Компоненты Swing поддерживают подключаемые вид и поведение (pluggable look-and-feel). Такое поведение компонентов реализовано на основе архитектурного подхода MVC.

4) В компоненты Swing встроена поддержка всплывающих подсказок, рамок, клавиатурных действий, средств для пользователей с ограниченными возможностями.

5) Компоненты Swing – это компоненты JavaBeans (модель визуального программирования, основанная на компонентах).

6) Приложения Swing используют классы AWT для обработки событий (пакет java.awt.event). Swing добавляет новые классы в пакет javax.swing.event, но они используются редко.

7) Приложения Swing используют классы AWT для управления размещением компонентов (FlowLayout и BorderLayout пакета java.awt). При этом добавлены новые менеджеры размещения, такие как Springs, Struts, и BoxLayout (пакет javax.swing).

8) В Swing реализована двойная буферизация и оптимизирован механизм прорисовки компонентов.

9) В Swing реализована поддержка много документального интерфейса (JLayeredPane и JInternalFrame).

10) В Swing существенно расширен список поддерживаемых компонентов.

На рис. 3 представлена иерархия классов Swing. На рисунке представлены наиболее часто используемые компоненты.

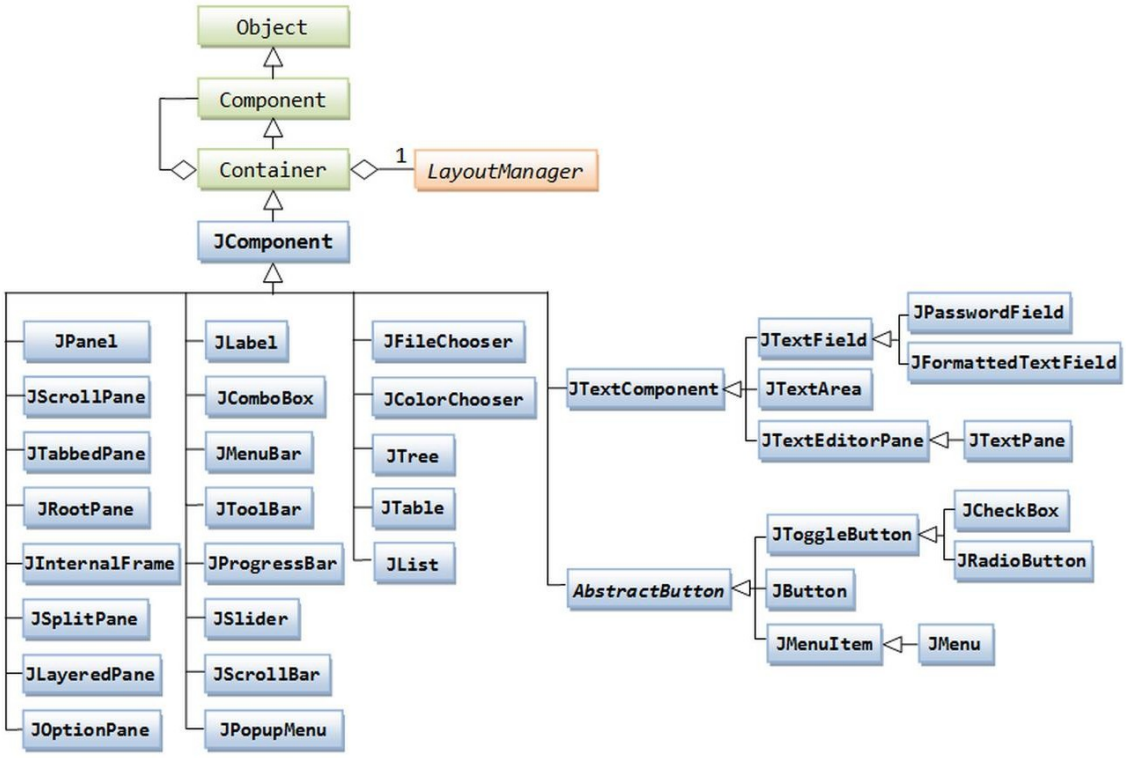

Рисунок 3 – Диаграмма классов Swing.

Описание и принципы работы с каждым из представленных компонентов представлены в документации Java. Перед рассмотрением отдельных компонентов пользовательского интерфейса, необходимо освоить работу с главным элементом графического пользовательского интерфейса – окном.

Контейнеры высшего уровня Swing.

Контейнеры высшего уровня (top level containers) представляют собой окна операционной системы, в которых программист может размещать элементы пользовательского интерфейса. К контейнерам высшего уровня относятся окна JFrame и JWindow, диалоговое окно JDialog, а также апплет JApplet (не является окном, но предназначен для построения интерфейса в браузере).

Для добавления элементов в окно (например, добавление кнопки на JFrame) логично было бы использовать метод add(), но контейнеры высшего уровня не позволяют добавлять компоненты напрямую. Для выполнения подобных задач применяется специальная панель JRootPane, которая расположена внутри контейнера высшего уровня. Эта панель называется корневой панелью (root panel). Следует понимать, что корневая панель не является отдельным компонентом – она представляет собой набор слоев (панелей), каждый из которых отвечает за определенный функционал. JRootPane содержит следующие слои (рис. 4):

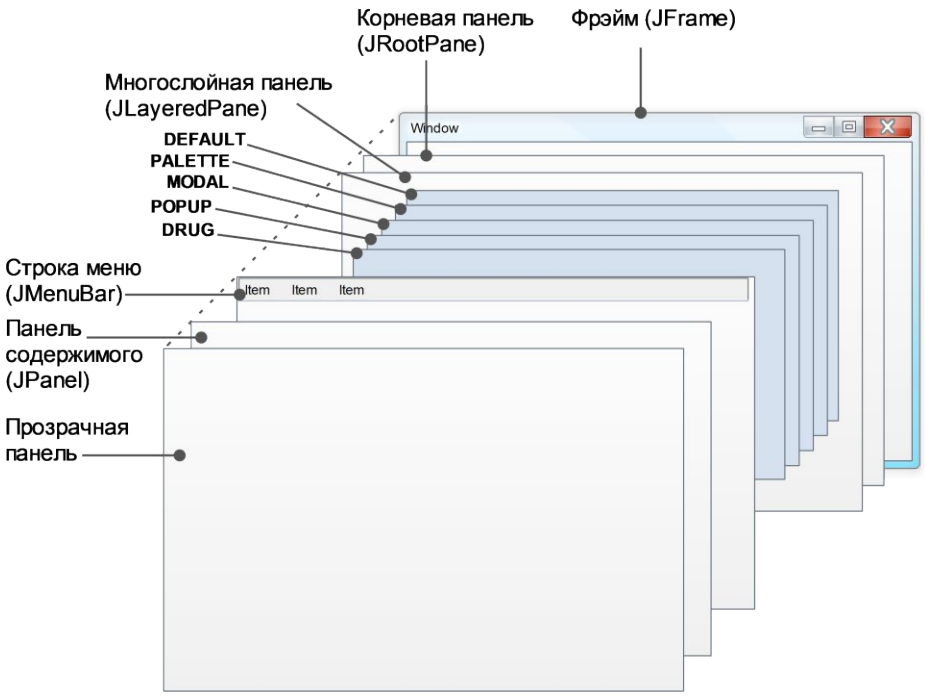

Рисунок 4 – Структура JFrame.

1. Многослойная панель JLayeredPane. Представляет собой совокупность слоев, в которых располагаются вызуальные компоненты Swing. Каждый слой имеет свой идентификатор и программист может создавать произвольное количество слоев и располагать в них свои элементы. Существуют стандартные (предопределенные слои), номера которых заданы идентификаторами, представленными в таблице 2.

Таблица 2 – Слои JLayeredPanel.

Идентификатор слоя	Описание слоя
DEFAULT_LAYER	Этот слой используется для всех обычных компонентов, которые программист добавляет в контейнер. В нем же располагаются внутренние окна многодокументных приложений.
PALETTE_LAYER	Слой предназначен для размещения так называемых палитр, или окон с набором инструментов, которые обычно перекрывают остальные элементы интерфейса. Создавать такие окна позволяет панель JDesktopPane, которая размещает их как раз в этом слое.
MODAL_LAYER	Судя по названию, разработчики планировали использовать этот слой для размещения легковесных модальных диалоговых окон. Однако такие диалоговые окна пока не реализованы. Этот слой в Swing в настоящее время не используется.
POPUP_LAYER	Наиболее часто используемый слой, служащий для размещения всплывающих меню и подсказок.
DRAG_LAYER	Самый верхний в стопке спой. Предназначен для обслуживания операций перетаскивания (drag and drop).

2. Строка меню (menu bar). Строка меню также размещается в многослойной панели, в том же специальном слое FRAME_CONTENT_LAYER, и занимает небольшое пространство в верхней части окна, в длину равное размеру окна, ширина используемого пространства при этом зависит от самой строки меню и содержащихся в ней элементов. Корневая панель следит за тем, чтобы панель содержимого и строка меню не

перекрывались. Если строка меню в контейнере высшего уровня не требуется, корневая панель использует все пространство окна для размещения панели содержимого.

3. Панель содержимого (content pane) –часть корневой панели, служащая для размещения компонентов пользовательского интерфейса программы. Она занимает большую часть пространства многослойной панели (за исключением того места, что занимает строка меню). Чтобы панель содержимого не закрывала добавляемые впоследствии в окно компоненты, многослойная панель размещает ее в специальном очень низком слое с названием. FRAME_CONTENT_LAYER, с номером -30000. Именно в панель содержимого следует добавлять все компоненты вашего пользовательского интерфейса, это основное отличие контейнеров высшего уровня Swing от их аналогов из AWT. Попытка добавить компоненты напрямую в контейнер (JFrame) высшего уровня Swing приведет к ошибке, так что программисту Java следует запомнить важный вызов:

getContentPane().add(Компонент);

Панель содержимого – это экземпляр панели JPanel.

5. Прозрачная панель (glass pane). Прозрачная панель используется в приложениях достаточно редко, поэтому по умолчанию корневая панель делает ее невидимой. Данный слой может использоваться для организации автоматизированного тестирования интерфейса приложения, а также для реализации различных анимационных эффектов.

Подробные описания каждого слоя заслуживают отдельного обсуждения и выходит за рамки данной лабораторной работы.

Классы Swing для представления окон.

Окно с рамкой JFrame унаследовано от класса JWindow (в рамках данного курса не рассматривается) и представляет собой наиболее часто используемое в приложениях окно. Окна JFrame обладают рамкой (которая позволяет пользователям легко изменять их размер), заголовком с названием приложения

(хотя можно оставлять этот заголовок пустым), иногда системным меню (позволяющим проводить манипуляции с этим окном) и кнопками для управления окном (чаще всего для его закрытия и свертывания). Именно класс JFrame применяется в подавляющем большинстве приложений для размещения компонентов пользовательского интерфейса.

Рассмотрим пример исходного кода, демонстрирующий создание окна:

```
1  package lesson8.Theory;
2
3  import javax.swing.JFrame;
4
5  public class JFrameExample extends JFrame{
6
7      public JFrameExample(){
8          // задание заголовка
9          super("Пример JFrame");
10         // что делать, если окно закрывается
11         setDefaultCloseOperation(EXIT_ON_CLOSE);
12         // иконка для окна
13         setIconImage(getToolkit().getImage("1.png"));
14         // установление размера
15         setSize(300, 100);
16         // показ окна
17         setVisible(true);
18     }
19
20     public static void main(String[] args) {
21         new JFrameExample();
22     }
23
24 }
```

Рисунок 5 – Исходный код приложения с ГИП.

На рис. 6 показан результат работы программы.

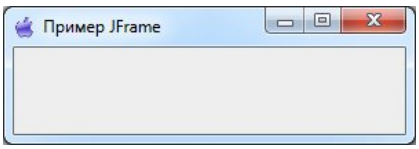

Рисунок .6 – Окно приложения, демонстрирующее базовый функционал JFrame.

В таблице 1 представлены методы класса JFrame, которые необходимо освоить в данной лабораторной работе.

Таблица 1 – Методы класса JFrame.

Метод	Описание
JFrame()	Конструктор. Создает окно, которое является

	невидимым.
JFrame(String title)	Конструктор. Создает окно с заданным заголовком. По-умолчанию окно является невидимым.
setTitle	Метод устанавливает заголовок окна.
setSize(int width, int height)	Устанавливает ширину и высоту окна.
setDefaultCloseOperation(int operation)	Устанавливает действие, которое выполнится при закрытии окна. Запоминать численные значения параметра operation не требуется – для них существуют символьные определения в классе JFrame: DO_NOTHING_ON_CLOSE – не производит никаких действий; HIDE_ON_CLOSE – просто скрывает окно; DISPOSE_ON_CLOSE – скрывает и удаляет окно из памяти, приложение продолжает выполняться; EXIT_ON_CLOSE – после закрытия окна полностью завершает приложение.
setLocationRelativeTo(Component c)	Устанавливает относительную позицию окна (относительно компонента c). Если передано значение null, то окно выравнивается по центру экрана.
void setVisible(boolean b)	Метод показывает (параметр true) или скрывает (false) форму на экране.
getTitle()	Возвращает строку – заголовок формы.

Программисту доступен еще один класс для создания фрейма верхнего уровня – JDialog. Диалоговые окна довольно часто используются в приложениях для получения информации, не имеющей прямого отношения к основному окну, например, для установки параметров, вывода важной вспомогательной информации, получения дополнительной информации от пользователя.

Диалоговые окна могут быть модальными (modal) – они запрещают всем остальным окнам приложения получать информацию от пользователя, пока тот не закончит работу с модальным диалоговым окном. Модальные диалоговые окна полезны в тех ситуациях, когда информация, получаемая от пользователя, коренным образом меняет работу приложения и поэтому до ее получения

продолжение работы невозможно. Внешний вид диалоговых окон повторяет окна с рамкой JFrame, но обычно у них меньше элементов управления (чаще всего, имеется только кнопка закрытия окна) и отсутствует системное меню. Как правило, диалоговые окна всегда располагаются поверх основного окна приложения, напоминая о том, что необходимо завершить ввод информации.

Принципы работы с диалоговыми окнами JDialog схожи с основными операциями класса JFrame. При работе с диалоговыми окнами следует помнить, что они бывают модальными и немодальными. Рассмотрим пример:

```
1  package lesson8.Theory;
2  import java.awt.Dialog;
3  import javax.swing.JDialog;
4
5  public class JDialogExample extends JDialog {
6
7      public JDialogExample(Dialog parent, String caption, boolean isModal){
8
9          super(parent, caption, isModal);
10         setDefaultCloseOperation(DISPOSE_ON_CLOSE);
11         setSize(200, 100);
12         setVisible(true);
13     }
14
15     public static void main(String[] args) {
16         JDialog diag1 = new JDialogExample(null, "Немодальный диалог", false);
17         new JDialogExample(diag1, "Модальный диалог", true);
18     }
19 }
```

Рисунок 7 – Исходный код создания диалоговых окон.

Таким образом, очевидно, что работа с классами JDialog и JFrame отличается не значительно.

Расположение элементов управления в приложениях Java. Менеджеры расположения Swing.

Поддержка менеджеров расположения встроена в базовый класс всех контейнеров java.awt.Container. Задать и получить расположение элементов можно методами setLayout() и getLayout(). В библиотеки существуют стандартные менеджеры расположения (семь различных классов), но если

программист хочет расположить компоненты особенным образом, ему необходимо реализовать интерфейс LayoutManager.

Стандартные менеджеры расположения: FlowLayout, GridLayout, BorderLayout, GridBagLayout, CardLayout, SpringLayout, BoxLayout. Рассмотрим наиболее простые менеджеры.

Полярное расположение BorderLayout. Позволяет создать простую структуру окна. При этом окно разбивается на четыре полярных области и одну область, заполняющую центр окна (рис. 8).

Рисунок 8 – Схема расположения областей менеджера BorderLayout.

Добавление компонента в контейнер производится методом add() с дополнительным параметром, указывающим на место расположения:

1. Значение BorderLayout.NORTH или строка «North» – компонент располагается вдоль верхней (северной) границы окна и растягивается на всю его ширину (панель инструментов).

2. Значение BorderLayout.SOUTH или строка «South» – компонент располагается вдоль нижней (южной) границы и растягивается на всю ширину окна (строка состояния).

3. Значение BorderLayout.WEST или строка «West» – компонент располагается вдоль левой (западной) границы окна и растягивается на всю его высоту, однако при этом учитываются размеры северных и южных компонентов (они имеют приоритет).

4. Значение BorderLayout.EAST или строка «East» – компонент располагается вдоль правой (восточной) границы окна.

5. Значение BorderLayout.CENTER или строка «Center» – компонент помещается в центр окна, занимая максимально возможное пространство.

Пример использования расположения BorderLayout представлен на рис. 9.

```
7  public class BorderLayoutExample extends JFrame {
8
9      public static void main(String[] args) {
10         new BorderLayoutExample();
11     }
12
13     public BorderLayoutExample(){
14         super("Пример BorderLayout");
15         setSize(400, 300);
16         setDefaultCloseOperation(EXIT_ON_CLOSE);
17         // Получаем панель содержимого
18         Container con = getContentPane();
19         // con.setLayout(new BorderLayout());
20         // Добавляем компоненты
21         con.add(new JButton("Кнопка Север"), "North");
22         con.add(new JButton("Кнопка Юг"), "South");
23         con.add(new JLabel("Надпись Запад"), BorderLayout.WEST);
24         con.add(new JTextField("Текст Восток"), BorderLayout.EAST);
25         con.add(new JEditorPane(JEditorPane.DEFAULT_KEYMAP,
26                 "Текстовый редактор"),
27                 BorderLayout.CENTER);
28         setIconImage(getToolkit().getImage("1.png"));
29         setVisible(true);
30     }
31
32 }
```

Рисунок 9 – Листинг с использованием менеджера BorderLayout.

На представленном фрагменте необходимо обратить внимание на строку 18, в которой получается объект панели содержимого. Именно на этом объекте вызываются методы add() в строках 21-25.

В строке 19 вызов метода setLayout() закомментирован. Этот метод устанавливает конкретный менеджер расположения, и все элементы, добавляемые на панель содержимого, будут расставлены в соответствии с данным менеджером расположения. В данном случае этот метод можно не использовать, так как BorderLayout – это менеджер расположения по умолчанию.

В строках 21-25 используется метод add(), в качестве второго параметра которого передается характер расположения элемента. Расположение можно определить как строку: «NORTH», «West» и т.д. Но можно использовать предопределенные значения: BorderLayout.NORTH, BorderLayout.WEST. Очевидны преимущества второго подхода.

Результат выполнения программы представлен на рис. 10.

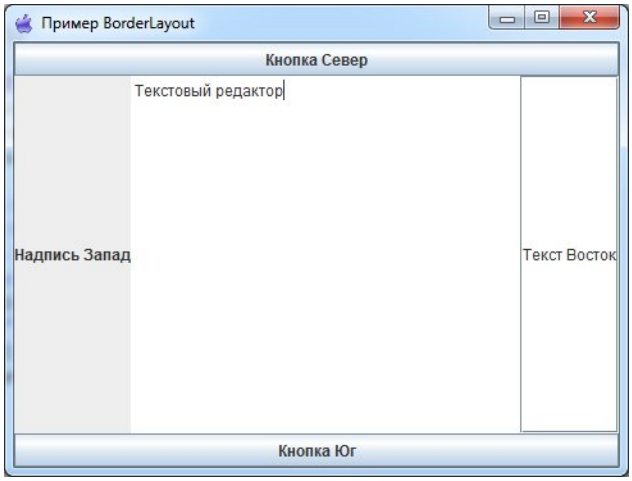

Рисунок 10 – Приложение с использованием BorderLayout.

Последовательное расположение FlowLayout. Менеджер задает последовательное расположение компонентов, «стопкой» (вертикально или горизонтально). Класс FlowLayout имеет три перегруженные версии:

– без параметров (расположение по умолчанию);

– с заданием выравнивания FlowLayout(int align), где переменная align может принимать значения FlowLayout.LEFT, FlowLayout.RIGHT, FlowLayout.CENTER, FlowLayout.LEADING, или FlowLayout.TRAILING.

– с выравниваем и заданием расстояния между элементами FlowLayout(int align, int hdist, int vdist), где hdist и vdist – расстояние между элементами, помещаемыми в контейнер, по горизонтали и вертикали соответственно.

Пример использования FlowLayout показан на рис. 11.

```java
package lesson8.Theory;
import java.awt.Container;
import java.awt.FlowLayout;
import javax.swing.*;

public class FlowLayoutExample extends JFrame {
    public static void main(String[] args) {
        new FlowLayoutExample();
    }

    public FlowLayoutExample(){
        super("Пример FlowLayout");
        setSize(400, 300);
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        // Получаем панель содержимого
        Container con = getContentPane();
        // Устанавливаем менеджер расположения
        con.setLayout(new FlowLayout());
        // Добавляем компоненты
        con.add(new JButton("Кнопка Север"));
        con.add(new JButton("Кнопка Юг"), "South");
        con.add(new JLabel("Надпись Запад"));
        con.add(new JTextField("Текст Восток"));
        con.add(new JEditorPane(JEditorPane.DEFAULT_KEYMAP,
                "Текстовый редактор"));
        setIconImage(getToolkit().getImage("1.png"));
        setVisible(true);
    }
}
```

Рисунок 11 – Расположение элементов с использованием менеджера FlowLayout.

На рис. 12 показан результат работы программы. Следует обратить внимание, что элементы располагаются по горизонтали, а элементы, которым не хватило места в горизонтальном ряду, перемещается на следующую строку.

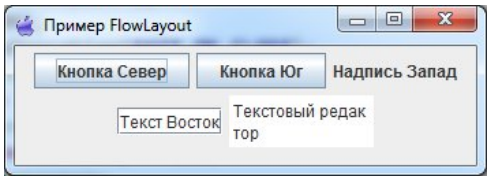

Рисунок 12 – Приложение с использованием менеджера FlowLayout.

Табличное расположение GridLayout. Менеджер расположения разделяет контейнер на ячейки, в которые затем помещаются элементы управления. Все ячейки таблицы имеют одинаковый размер, равный максимальному размеру добавляемого элемента.

Конструкторы менеджера расположения:

– конструктор без параметров GridLayout() – устанавливает один столбец и одну строку;

– конструктор GridLayout(int rows, int cols) с указанием количества строк (rows) и количества столбцов (cols);

– конструктор GridLayout(int rows, int cols, int hdist, int vdist) – задает количество строк (rows) и столбцов (cols), а также горизонтальное (hdist) и вертикальное (vdist) расстояния между элементами.

Если вместо rows или cols передано значение 0 , то предполагается, что возможно неограниченное количество строк или столбцов. Сетка будет увеличиваться динамически. Все параметры, задаваемые конструкторами можно изменить с помощью методов объекта GridLayout.

Пример использования менеджера показан GridLayout на рис. 13.

```
7  public class GridLayoutExample extends JFrame{
8      public GridLayoutExample(){
9          super("Пример GridLayout");
10         setSize(500, 500);
11         setDefaultCloseOperation(EXIT_ON_CLOSE);
12         // Получаем панель содержимого
13         Container con = getContentPane();
14         // Устанавливаем менеджер расположения
15         con.setLayout(new GridLayout(10, 10, 1, 1));
16         // Добавляем компоненты
17         for (int i = 0; i < 10; i++)
18             for (int j = 0; j < 10; j++)
19                 con.add(new JButton(String.valueOf(i)+String.valueOf(j)));
20         setVisible(true);
21     }
22     public static void main(String[] args) {
23         new GridLayoutExample();
24     }
25 }
```

Рисунок 13 – Использование менеджера GridLayout.

Результат работы программы представлен на рис. 14.

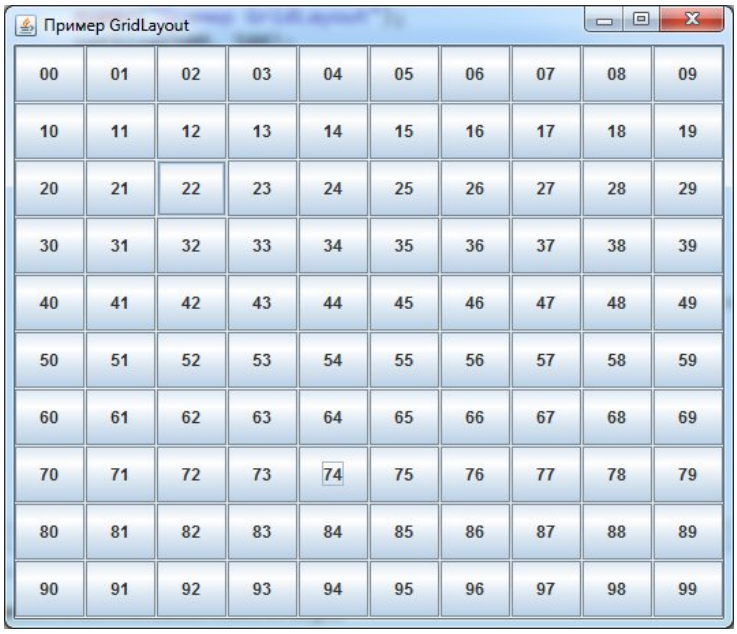

Рисунок 14 – Приложение с использованием менеджера GridLayout.

Рассмотрение прочих менеджеров расположения выходит за рамки данной лабораторной работы. Необходимо отметить, что если программист намерен применить абсолютное позиционирование элементов, то следует использовать вызов setLayout(null).

Блочное расположение BoxLayout. Менеджер расположения, обладающий широкими одновременно возможностями GridBagLayout (сложного менеджера с большим количеством настроек) и BorderLayout (простого менеджера, возможности которого были рассмотрены ранее). Это относительно новый компонент (пакет javax.swing), который рекомендуется использовать для размещения компонентов в современных приложениях. Конструктору класса BoxLayout необходимо указать контейнер, в котором он будет применен, и характер расположения элементов (Y_AXIS – по вертикали; X_AXIS – по горизонтали). Пример использования BoxLayout представлен на рис. 15.

```
 9 public class BoxLayoutExample extends JFrame {
10     public BoxLayoutExample(int axis){
11         super("Пример BoxLayout");
12         setSize(500, 500);
13         setDefaultCloseOperation(EXIT_ON_CLOSE);
14         // Получаем панель содержимого
15         Container container = getContentPane();
16         // Создаем объект BoxLayout
17         BoxLayout manager = new BoxLayout(container, axis);
18         // Устанавливаем менеджер расположения
19         container.setLayout(manager);
20         // Добавляем компоненты
21         URL imgURL = BoxLayoutExample.class.getResource("B-Favorites.png");
22         JButton btn = new JButton("Favorites", new ImageIcon(imgURL));
23         container.add(btn);
24         imgURL = BoxLayoutExample.class.getResource("B-Logoff.png");
25         btn = new JButton("Logoff", new ImageIcon(imgURL));
26         container.add(btn);
27         imgURL = BoxLayoutExample.class.getResource("B-Previous.png");
28         btn = new JButton("Previous", new ImageIcon(imgURL));
29         container.add(btn);
30         imgURL = BoxLayoutExample.class.getResource("B-Play.png");
31         btn = new JButton("Play", new ImageIcon(imgURL));
32         container.add(btn);
33         imgURL = BoxLayoutExample.class.getResource("B-Next.png");
34         btn = new JButton("Next", new ImageIcon(imgURL));
35         container.add(btn);
36         setVisible(true);
37     }
38     public static void main(String[] args) {
39         new BoxLayoutExample(BoxLayout.X_AXIS);
40         new BoxLayoutExample(BoxLayout.Y_AXIS);
41     }
42 }
```

Рисунок 15 – Использование менеджера BoxLayout.

В данном листинге видно, что конструктор BoxLayout отличается от конструкторов менеджеров расположения, рассмотренных ранее. Конструктору в качестве параметра передается контейнер, в котором будет применен данный менеджер.

В результате работы программы создаются две формы с различным расположением элементов (рис. 16).

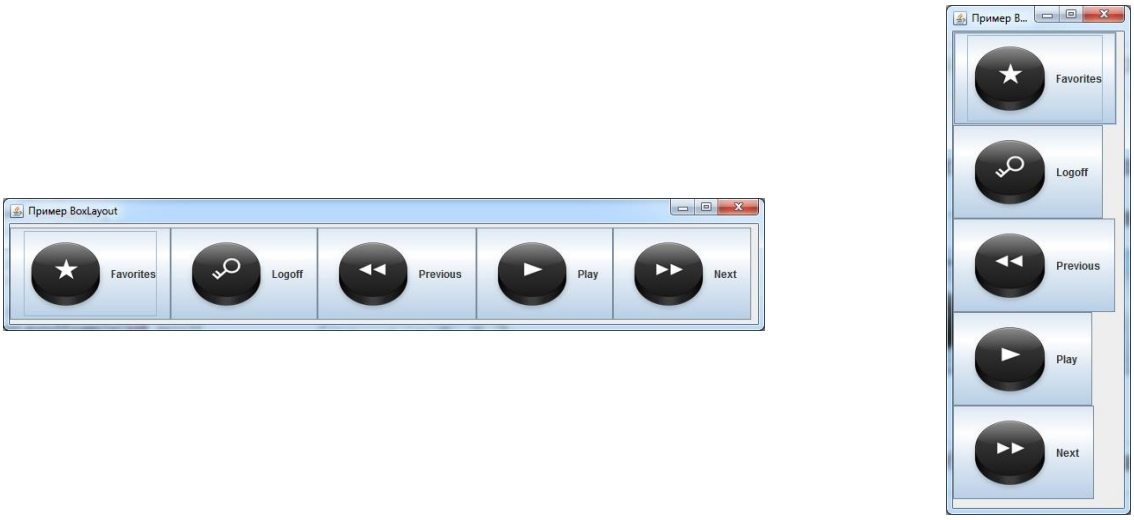

Рисунок 16 – Приложение с использованием менеджера BoxLayout.

При создании промышленных приложений менеджеры обычно комбинируют и вкладывают друг в друга.

Контейнер верхнего уровня WPF

Объект Window – основной элемент традиционных приложений, в нем отображается их основное содержимое. В Windows Presentation Foundation (WPF) за объектом Window скрывается обычное окно Win32. Операционная система не различает окна с WPF- и Win32-содержимым: обрамление рисуется одинаково, на панели задач они неотличимы друг от друга и т. д. Обрамление (Chrome) – это просто другое название области окна, которая обрамляет клиентскую область и в числе прочего содержит кнопки Minimize (Свернуть), Maximize (Развернуть) и Close (Закрыть).

Объект Window – это абстракция окна Win32 (так же, как класс Form в Windows Forms), содержащая ряд простых методов и свойств. Создав объект Window, программист может показать его на экране с помощью метода Show, скрыть – методом Hide() (это то же самое, что присвоить свойству Visibility значение Hidden или Collapsed) и закрыть навсегда – методом Close().

Внешним видом Window можно управлять с помощью таких свойств, как Icon (иконка приложения), Title (интерпретируется как заголовок окна) и

WindowStyle. Для управления положением на экране служат свойства Left и Top. Более осмысленного поведения можно добиться, присваивая свойству WindowStartupLocation значение CenterScreen или CenterOwner. Короче говоря, с помощью установки свойств можно делать почти все, что принято ожидать от окна. Скажем, если установить для Topmost значение true, то окно будет всегда отображаться поверх других, а если присвоить ShowInTaskbar значение false, то значок окна не будет показываться на панели задач.

Всякий раз, как окно становится активным или неактивным (например, из-за того, что пользователь переключается между разными окнами), возникает событие Activated или Deactivated объекта Window.

На рис. 17 представлено определение окна с использованием языка разметки XAML. Данное определение генерируется Visual Studio по умолчанию.

```
<Window x:Class="WpfExample.Window1"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window1" Height="300" Width="300">
    <Grid>

    </Grid>
</Window>
```

Рисунок 17 – Шаблон окна приложения, генерируемого по умолчанию.

Окно может содержать только один дочерний элемент. Для размещения в окне множества дочерних элементов управления в WPF используется подход: сначала в фрейме окна размещается специализированная панель (контейнер компоновки), а затем, в данном контейнере располагаются элементы управления.

Контейнеры компоновки WPF

Для размещения элементов управления в WPF используются специализированные компоненты: контейнеры компоновки. В технологии Windows Forms применяется механизм абсолютного позиционирования

элементов управления: задаются координаты верхнего левого угла элемента (свойства Top, Left) и его размер (свойства Width и Height).

В WPF компоновка определяется используемым контейнером. Окно в WPF должно реализовывать следующие принципы:

1. Элементы (такие как элементы управления) не должны иметь явно установленных размеров. Вместо этого они растут, чтобы уместить свое содержимое. Например, кнопка увеличивается при добавлении в нее текста. Можно ограничить элементы управления приемлемыми размерами, устанавливая максимальное и минимальное их значение.

2. Элементы не указывают свою позицию в экранных координатах. Вместо этого они упорядочиваются своим контейнером на основе размера, порядка и (необязательно) другой информации, специфичной для контейнера компоновки. Для добавления пробелов между элементами служит свойство Margin.

3. Контейнеры компоновки «разделяют» доступное пространство между своими дочерними элементами. Они пытаются обеспечить для каждого элемента его предпочтительный размер (на основе его содержимого), если только позволяет свободное пространство. Они могут также выделять дополнительное пространство одному или более дочерним элементам.

4. Контейнеры компоновки могут быть вложенными. Типичный пользовательский интерфейс начинается с Grid – наиболее развитого контейнера, и содержит другие контейнеры компоновки, которые организуют меньшие группы элементов, такие как текстовые поля с метками, элементы списка, значки в панели инструментов, колонка кнопок и т.д.

Все контейнеры компоновки WPF являются панелями, которые унаследованы от абстрактного класса System.Windows.Controls.Panel (рис. 18). Класс Panel добавляет небольшой набор возможностей всем производным классам.

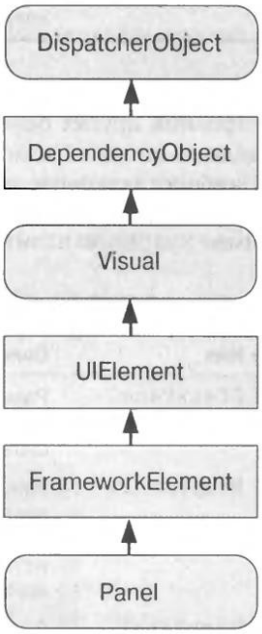

Рисунок 18 – Иерархия классов панелей WPF.

Сам по себе базовый класс Panel – это абстрактный класс, не допускающий создания объектов. WPF предлагает набор производных от Panel классов, которые можно использовать для организации компоновки. Основные контейнеры компоновки перечислены в табл. 2. Как и все элементы управления WPF, а также большинство визуальных элементов, эти классы находятся в пространстве имен System.Windows.Controls.

Таблица 2 – Контейнеры компоновки WPF.

Контейнер	Описание
StackPanel	Размещает элементы в горизонтальном или вертикальном стеке. Этот контейнер компоновки обычно используется в небольших разделах крупного и более сложного окна.
WrapPanel	Размещает элементы в последовательностях строк с переносом. В горизонтальной ориентации WrapPanel располагает элементы в строке слева направо, затем переходит к следующей строке. В вертикальной ориентации WrapPanel располагает элементы сверху вниз, используя дополнительные колонки для дополнения оставшихся элементов.
DockPanel	Выравнивает элементы по краю контейнера.

Grid	Выстраивает элементы в строки и колонки невидимой таблицы. Это один из наиболее гибких и широко используемых контейнеров компоновки.
UniformGrid	Помещает элементы в невидимую таблицу, устанавливая одинаковый размер для всех ячеек. Данный контейнер компоновки используется нечасто.
Canvas	Позволяет элементам позиционироваться абсолютно — по фиксированным координатам. Этот контейнер компоновки более всего похож на традиционный компоновщик Windows Forms, но не предусматривает средств привязки и стыковки. В результате это неподходящий выбор для окон переменного размера.

Контейнер StackPanel

На рис. 19 представлен простой пример компоновки элементов с использованием контейнера StackPanel.

```
1  <Window x:Class="WpfExample.Window31"
2          xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
3          xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
4          Title="Window31" Height="127" Width="300">
5      <StackPanel>
6          <Button>Кнопка 1</Button>
7          <Button>Кнопка 2</Button>
8          <Button>Кнопка 3</Button>
9          <Button>Кнопка 4</Button>
10     </StackPanel>
11 </Window>
```

Рисунок 19 – Использование StackPanel.

Результат подобной компоновки представлен на рис. 20. Без дополнительной информации о порядке, координатах и размерах кнопок с помощью контейнера StackPanel удалось расположить элементы вертикально.

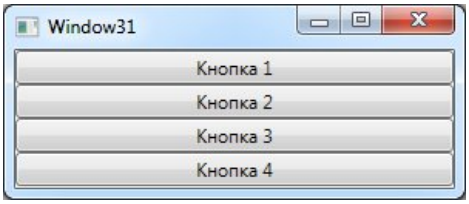

Рисунок 20 – Окно с контейнером StackPanel.

По умолчанию панель StackPanel располагает элементы сверху вниз, устанавливая высоту каждого из них такой, которая необходима для отображения его содержимого.

В данном примере это значит, что размер кнопок устанавливается достаточно большим для спокойного размещения текста внутри них. Все элементы растягиваются на полную ширину StackPanel, которая равна ширине окна. Если вы расширите окно, StackPanel также расширится, и кнопки растянутся, чтобы заполнить ее.

StackPanel может также использоваться для упорядочивания элементов в горизонтальном направлении за счет установки свойства Orientation (рис. 21):

<StackPanel Orientation=llHorizontalll>

```
<Window x:Class="WpfExample.Window31"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window31" Height="127" Width="300">
    <StackPanel Orientation="Horizontal">
        <Button>Кнопка 1</Button>
        <Button>Кнопка 2</Button>
        <Button>Кнопка 3</Button>
        <Button>Кнопка 4</Button>
    </StackPanel>
</Window>
```

Рисунок 21 – Использование StackPanel.

Результат представлен на рис. 22.

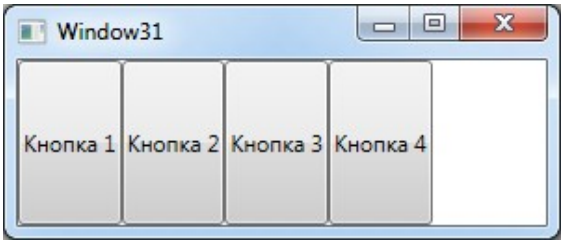

Рисунок 22 – Использование горизонтального размещения StackPanel.

Размещение элементов управления определяется контейнером, но элементы управления могут повлиять на этот процесс, явно задавая свойства. Свойства элементов, которые влияют на размещение в контейнере представлены в таблице 3.

Таблица 3 – Свойства компоновки.

Наименование свойства	Описание
HonzontalAlignment	Определяет позиционирование дочернего элемента внутри контейнера компоновки, когда имеется дополнительное пространство по горизонтали. Доступные значения: Center, Left, Right или Stretch.
VerticalAlignment	Определяет позиционирование дочернего элемента внутри контейнера компоновки, когда имеется дополнительное пространство по вертикали. Доступные значения: Center, Top, Bottom или Stretch.
Margin	Добавляет некоторое пространство вокруг элемента. Свойство Margin – это экземпляр структуры System.Windows.Thickness, с отдельными компонентами для верхней, нижней, левой и правой граней.
MinWidth и MinHeight	Устанавливает минимальные размеры элемента. Если элемент слишком велик, чтобы поместиться в его контейнер компоновки, он будет усечен.
MaxWidth и MaxHeight	Устанавливает максимальные размеры элемента. Если контейнер имеет свободное пространство, элемент не будет увеличен сверх указанных пределов, даже если свойства HonzontalAlignment и VerticalAlignment установлены в Stretch.
Width и Height	Явно устанавливают размеры элемента. Эта установка переопределяет значение Stretch для свойств HorizontalAlignment и VerticalAlignment. Однако данный размер не будет установлен, если выходит за пределы, заданные в MinWidth, MinHeight, MaxWidth и MaxHeight.

Элемент Border

Border не является одной из панелей компоновки, но это удобный элемент, который часто используется.

Класс Border принимает единственную порцию вложенного содержимого (которым часто является панель компоновки) и добавляет фон или рамку вокруг него. Для работы с Border понадобятся свойства, перечисленные в табл. 4.

Таблица 4 – Свойства класса Border.

Наименование свойства	Описание
Background	С помощью объекта Brush устанавливает фон, который появляется под содержимым. Можно использовать сплошной цвет либо что-нибудь более сложное.
BorderBrush и BorderThickness	Устанавливают цвет рамки, который появляется на границе объекта Border, и толщину рамки. Для отображения рамки потребуется установить оба свойства.
CornerRadius	Позволяет скруглить углы рамки. Чем больше значение CornerRadius, тем заметнее эффект.
Padding	Добавляет пространство между рамкой и содержимым внутри нее.

На рис. 23 представлен фрагмент кода XAML, использующий данные свойства.

```
<Window x:Class="WpfExample.Window31"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Window31" Height="127" Width="300">
    <Border Background="GreenYellow" BorderBrush="Red"
            BorderThickness="5" CornerRadius="10" Padding="10">
        <StackPanel Orientation="Vertical">
            <Button>Кнопка 1</Button>
            <Button>Кнопка 2</Button>
            <Button>Кнопка 3</Button>
            <Button>Кнопка 4</Button>
        </StackPanel>
    </Border>

</Window>
```

Рисунок 23 – Использование свойств Border.

На рис. 24 показано результирующее окно.

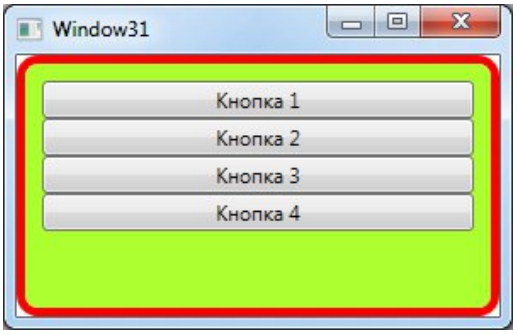

Рисунок 24 – Результат применения Border.

Элементы WrapPanel и DockPanel

Панель WrapPanel располагает элементы управления в доступном пространстве – по одной строке или колонке за раз. По умолчанию свойство WrapPanel.Orientation установлено в Horizontal; элементы управления располагаются слева направо, затем – в следующих строках. Установка значения Vertical для свойства WrapPanel. Orientation приводит к размещению элементов в нескольких колонках.

Подобно StackPanel, панель WrapPanel действительно предназначена для управления мелкими деталями пользовательского интерфейса, а не компоновкой всего окна. Например, WrapPanel можно использовать для удержания вместе кнопок в элементе управления, подобном панели инструментов.

Панель DockPanel обеспечивает более интересный вариант компоновки. Эта панель растягивает элементы управления вдоль одной из внешних границ. Простейший способ представить это – вообразить панель инструментов, которая присутствует в верхней части многих Windows-приложений. Такие панели инструментов пристыковываются к верхней части окна. Как и в случае StackPanel, пристыкованные элементы должны выбрать один аспект компоновки. Например, если вы пристыковать кнопку к верхней части DockPanel, она растянется на всю ширину DockPanel, но получит высоту, которая ей понадобится (на основе своего содержимого и свойства MinHeight). С другой стороны, если пристыковать кнопку к левой стороне контейнера, ее высота будет растянута для заполнения контейнера, но ширина будет установлена по необходимости.

Дочерние элементы DockPanel прикрепляются к одной из сторон панели посредством задания присоединенного свойства Dock. Элементы на самом деле не имеют этого свойства – они приобретают его после помещения в контейнер.

На рис. 25 показан пример использования DockPanel.

```
<Window x:Class="WpfExample.DockPanelExample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="DockPanelExample" Height="300" Width="300">
    <DockPanel>
        <Button Content="кнопка 1" DockPanel.Dock="Top"/>
        <Button Content="кнопка 2" DockPanel.Dock="Bottom"/>
        <Button Content="кнопка 3" DockPanel.Dock="Left"/>
        <Button Content="кнопка 4" DockPanel.Dock="Right"/>
        <Button Content="кнопка 5"/>
    </DockPanel>
</Window>
```

Рисунок 25 – Использование DockPanel.

На рис. 26 представлен результат.

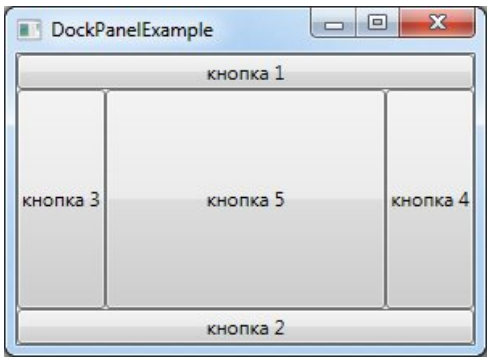

Рисунок 26 – Контейнер DockPanel в окне приложения.

Контейнер Grid

Элемент управления Grid – это наиболее мощный контейнер компоновки в WPF. Большая часть того, что можно достичь с помощью других элементов управления компоновкой, также возможно и в Grid. Контейнер Grid является идеальным инструментом для разбиения окна на меньшие области, которыми можно управлять с помощью других панелей.

При добавлении в Visual Studio нового документа XAML для окна автоматически добавляются дескрипторы Grid в качестве контейнера первого уровня, вложенного внутрь корневого элемента Window.

Хотя Grid задуман как невидимый элемент, можно установить свойство Grid.ShowGridLines в true и получить наглядное представление о нем. Это средство на самом деле не предназначено для украшения окна. В действительности это средство для облегчения отладки, которое предназначено

для того, чтобы наглядно показать, как Grid разделяет пространство на отдельные области. Благодаря ему, появляется возможность точно контролировать то, как Grid выбирает ширину колонок и высоту строк.

Создание компоновки на основе Grid – поэтапный процесс. Сначала выбирается необходимое количество колонок и строк. Затем каждому содержащемуся элементу назначается соответствующая строка и колонка, тем самым помещая его в правильное место.

Колонки и строки создаются путем заполнения объектами коллекции Grid.ColumnDefinitions и Grid.RowDefinitions. Например, если необходимо создать две строки и три колонки, то используются следующие дескрипторы:

```
<Window x:Class="WpfExample.GridExample"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="GridExample" Height="300" Width="300">
    <Grid>
        <Grid Background="GreenYellow">
            <Grid.ColumnDefinitions>
                <ColumnDefinition/>
                <ColumnDefinition/>
                <ColumnDefinition/>
            </Grid.ColumnDefinitions>
            <Grid.RowDefinitions>
                <RowDefinition/>
                <RowDefinition/>
                <RowDefinition/>
            </Grid.RowDefinitions>
            <Button Grid.Column="0" Grid.Row="0" Content="Кнопка (0, 0)"/>
            <Button Grid.Column="1" Grid.Row="1" Content="Кнопка (1, 1)"/>
            <Button Grid.Column="2" Grid.Row="2" Content="Кнопка (2, 2)"/>
            <Button Grid.Column="2" Grid.Row="0" Content="Кнопка (2, 0)"/>
            <Button Grid.Column="0" Grid.Row="2" Content="Кнопка (0, 2)"/>
        </Grid>
    </Grid>
</Window>
```

Рисунок 27 – Контейнер Grid.

Результирующее окно представлено на рис. 28.

Рисунок 28 – Расположение кнопок в контейнере Grid.

Таким образом, в подходах, применяемых в рамках технологий Swing (Java SE) и Windows Presentation Foundation, прослеживается определенное сходство: принципы компоновки элементов; фреймы высшего уровня. На сегодняшний день технология Microsoft превосходит Java SE в плане удобства разработки Windows-приложений, но технология Java SE, в свою очередь, не ограничена одной операционной системой. Также следует учитывать стоимость программных средств. Инструменты Swing имеет перспективы использования в рамках технологии Java EE, то есть библиотека Swing не ограничена только клиентскими приложениями Java SE.

Отделение интерфейсов от кода в технологии WPF привело к появлению дополнительного языка XAML, который также применяется для разработкивеб-приложений (технология Silverlight).

Методика и порядок выполнения работы

В рамках выполнения лабораторной работы необходимо освоить два подхода для программирования приложений с использованием GUI.

Первый подход предполагает проектирование фрейма, а затем вызов окна из функции main(). Второй подход предполагает определение функции main() непосредственно в классе окна (фрейма) как метода данного класса. Такое использование классов экранных форм не реализуется в WPF. Это особенность разработки графических приложений Java SE. Рассмотрим оба подхода.

Реализация первого подхода с функцией main(), осуществляющей создание окна приложения и управление им. Для этого необходимо выполнить следующую последовательность действий (приведен пример реализации в среде Eclipse):

1. Создайте проект приложения Java. После создания проекта и, если необходимо, пакетов, необходимо добавить класс главного окна с именем FrameWithoutMain. Листинг данного класса представлен на рис. 29.

```
3  import java.awt.Container;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6
7  import javax.swing.JFrame;
8  import javax.swing.JButton;
9
10 public class FrameWithoutMain extends JFrame {
11
12     private JButton btn;
13
14     public FrameWithoutMain(){
15         super("Окно без функции main");
16         setSize(400, 200);
17         setDefaultCloseOperation(EXIT_ON_CLOSE);
18         Container cont = getContentPane();
19
20         btn = new JButton("Log");
21         btn.addActionListener(new ActionListener() {
22             @Override
23             public void actionPerformed(ActionEvent e) {
24                 System.out.println("Button <Log> pressed!");
25             }
26         });
27
28         cont.add(btn, BorderLayout.CENTER);
29     }
30 }
```

Рисунок 29 – Определение фрейма.

2. Создайте класс, содержащий функцию main(). На рис. 30 представлен код функции main(), которая определена в классе FrameExample.

```
1  package lesson8.method;
2
3  public class FrameExample {
4
5      public static void main(String[] args) {
6          FrameWithoutMain window = new FrameWithoutMain();
7          window.setVisible(true);
8      }
9
10 }
```

Рисунок 30 – Управление окном приложения.

3. Запустите приложение. Рассмотрите представленные варианты исходного кода.

Рассмотрим второй подход применения класса JFrame, когда функция main() определяется непосредственно в классе окна. Для изучения данного

механизма создайте проект Java и реализуйте в нем класс FrameWithMain (рис. 31).

```
1  package lesson8.method;
2  import java.awt.Container;
3  import java.awt.Dimension;
4  import java.awt.event.ActionEvent;
5  import java.awt.event.ActionListener;
6  import javax.swing.Box;
7  import javax.swing.BoxLayout;
8  import javax.swing.JButton;
9  import javax.swing.JFrame;
10
11 public class FrameWithMain extends JFrame {
12
13     JButton btn;
14
15     public FrameWithMain() {
16         super("Окно с методом main");
17         setSize(400, 200);
18         setDefaultCloseOperation(EXIT_ON_CLOSE);
19         Container cont = getContentPane();
20         BoxLayout box = new BoxLayout(cont, BoxLayout.X_AXIS);
21         cont.setLayout(box);
22
23         btn = new JButton("Log");
24         btn.setPreferredSize(new Dimension(200, 100));
25         btn.addActionListener(new ActionListener() {
26             @Override
27             public void actionPerformed(ActionEvent e) {
28                 System.out.println("Button <Log> pressed!");
29             }
30         });
31
32         cont.add(Box.createHorizontalGlue());
33         cont.add(btn);
34         cont.add(Box.createHorizontalGlue());
35         setLocationRelativeTo(null);
36         setVisible(true);
37     }
38
39     public static void main(String[] args) {
40         new FrameWithMain();
41     }
42 }
```

Рисунок 31 – Класс окна, содержащий функцию main().

Данный класс окна, фактически, отображает сам себя. При этом класс инкапсулирует функцию main(). В коде на рис. 31 обратите внимание на использование контейнера компоновки BoxLayout.

Таким образом, можно создавать окна в среде Eclipse. Некоторые среды разработки обладают графическими инструментами создания окон (NetBeans, MS Visual Studio).

Рассмотрим пример решения учебной задачи с использованием классов библиотеки Swing.

Задание. Реализовать приложение для вычисления значения выражения:

$$Z = \begin{cases} 1 - \frac{1}{2} + \frac{1}{3} - \frac{1}{4} + \frac{1}{5} - ..; \\ \sum_{i=1}^{N} i^2. \end{cases}$$

В данном выражении необходимо предоставить пользователю возможность выбора формулы расчета Z.

Решение. Решение поставленной задачи производится с использованием среды Eclipse.

1. Перед этапом программирования необходимо спроектировать интерфейс приложения. На рис. 8.32 представлен проект экранной формы приложения.

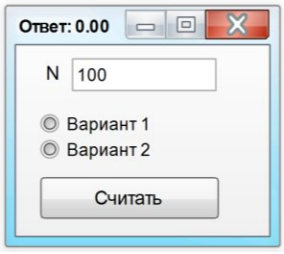

Рисунок 32 – Проект экранной формы.

2. После создания схемы реализуем ее в коде. На рис. 8.33 представлен код класса окна приложения.

3. После реализации внешнего вида главного окна приложения, создадим метод, который будет отвечать за вычисление результата. Данный метод – обработчик нажатия кнопки «Считать». Добавьте соответствующий класс слушателя и наполните метод слушателя кодом (рис. 8.34).

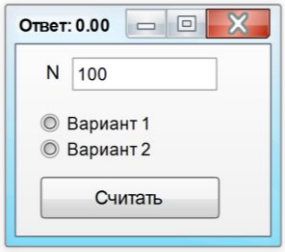

Рисунок 33 – Код главного окна приложения.

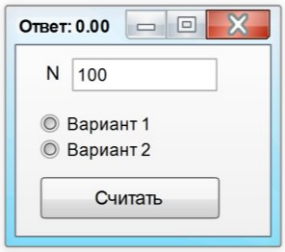

Рисунок 34 – Код метода слушателя, реализующий вычисление выражения.

Индивидуальное задание.

Вариант	Выражение для вычисления
1	$Z=\begin{cases}1-\frac{X}{2}+\frac{Y^2}{6}-\frac{X^3}{24}+\frac{Y^4}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{3+j}{i}.\end{cases}$
2	$Z=\begin{cases}-\frac{1}{2}+\frac{Y}{6}-\frac{X^2}{24}+\frac{Y^3}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{i^2+j}{i^j}.\end{cases}$
3	$Z=\begin{cases}-\frac{1}{1}+\frac{Y}{2}-\frac{X^2}{3}+\frac{Y^3}{4}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{j^2+i}{i^j+j}.\end{cases}$
4	$Z=\begin{cases}-\frac{Y}{2}+\frac{Y^2}{6}-\frac{X^3}{24}+\frac{Y^4}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{a\cdot i+b\cdot j}{c\cdot i^j}.\end{cases}$

5	$Z=\begin{cases}-\frac{p^2}{2}+\frac{p^3}{6}-\frac{p^4}{24}+\frac{p^5}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{i^2+b\cdot j}{c\cdot i^3}.\end{cases}$
6	$Z=\begin{cases}-\frac{X}{2}+\frac{p\cdot X^2}{6}-\frac{p^2\cdot X^3}{24}+\frac{p^3\cdot X^4}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{i^2+j^2}{i^3+j^3}.\end{cases}$
7	$Z=\begin{cases}-\frac{X}{2}+\frac{X^2}{6}-\frac{X^3}{24}+\frac{X^4}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{i^2}{i^3+j^3}.\end{cases}$
8	$Z=\begin{cases}-\frac{1}{2}+\frac{X}{6}-\frac{X^2}{24}+\frac{X^3}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{a\cdot i}{i^3+j^3}.\end{cases}$
9	$Z=\begin{cases}-\frac{f}{2}+\frac{X\cdot f^2}{6}-\frac{X^2\cdot f^3}{24}+\frac{X^3\cdot f^4}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{a\cdot(i+j)}{i\cdot(i+2)}.\end{cases}$
10	$Z=\begin{cases}1-\frac{X}{2}+\frac{Y^2}{6}-\frac{X^3}{24}+\frac{Y^4}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{3+j}{i}.\end{cases}$
11	$Z=\begin{cases}-\frac{1}{2}+\frac{Y}{6}-\frac{X^2}{24}+\frac{Y^3}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{i^2+j}{a^j}.\end{cases}$
12	$Z=\begin{cases}-\frac{1}{1}+\frac{Y}{2}-\frac{X^2}{3}+\frac{Y^3}{4}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{j^2+i}{i^j+j}.\end{cases}$

13	$Z=\begin{cases}-\frac{Y}{2}+\frac{Y^2}{6}-\frac{X^3}{24}+\frac{Y^4}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{a\cdot i+b\cdot j}{c\cdot i^j}.\end{cases}$
14	$Z=\begin{cases}-\frac{p^2}{2}+\frac{p^3}{6}-\frac{p^4}{24}+\frac{p^5}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{i^2+b\cdot j}{c^i\cdot i^3}.\end{cases}$
15	$Z=\begin{cases}-\frac{X}{2}+\frac{p\cdot X^2}{6}-\frac{p^2\cdot X^3}{24}+\frac{p^3\cdot X^4}{120}-..;.\\ \sum_{i=1}^{N}\sum_{j=1}^{R}\frac{i^2+j^2}{i^3+j^3}.\end{cases}$