



## Урок 4

# Продвинутые вопросы создания графического интерфейса

Создание собственных элементов управления. Работа с графикой.  
Обработка событий

## Оглавление

Создание формы	2
Обработка событий	8
Домашнее задание	9
Дополнительные материалы	9

# Создание формы

Для создания простого окна достаточно создать класс, унаследовать его от `JFrame`, и создать объект этого класса в методе `main()`, как показано ниже:

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setTitle("Test Window");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setBounds(300, 300, 400, 400);
        setVisible(true);
    }
}

public class MainClass {
    public static void main(String[] args) {
        new MyWindow();
    }
}
```

В конструкторе сразу же задаются параметры окна: `setTitle()` – устанавливает заголовок; `setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE)` – сообщает системе о необходимости завершить работу программы при закрытии окна; `setBounds()` – задает координаты и размер формы в пикселях; `setVisible(true)` – показывает форму на экране, желательно вызывать этот метод после всех настроек, иначе при запуске некоторые элементы могут быть отображены некорректно.

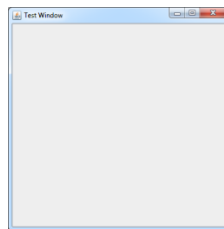


Рисунок 1 – Пустое окно Swing

Попробуем добавить на форму несколько управляющих элементов, например, 5 кнопок `JButton`. Как правило, в библиотеке Swing названия классов, отвечающих за элементы графического интерфейса, начинаются с буквы `J`.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setTitle("Test Window");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setBounds(300, 300, 400, 400);
        JButton[] jbs = new JButton[5];
        for (int i = 0; i < 5; i++) {
            jbs[i] = new JButton("#" + i);
        }
        setLayout(new BorderLayout()); // выбор компоновщика элементов
        add(jbs[0], BorderLayout.EAST); // добавление кнопки на форму
        add(jbs[1], BorderLayout.WEST);
        add(jbs[2], BorderLayout.SOUTH);
        add(jbs[3], BorderLayout.NORTH);
        add(jbs[4], BorderLayout.CENTER);
        setVisible(true);
    }
}
```

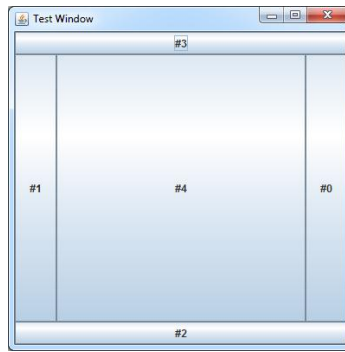


Рисунок 2 – Форма с несколькими кнопками

За расстановку элементов на форме отвечают компоновщики элементов, в данном случае мы использовали BorderLayout. После создания кнопок, их необходимо добавить/расположить на форме, для этого используется метод add(элемент\_интерфейса, местонахождение).

### Наиболее используемые компоновщики элементов:

**BorderLayout** – располагает элементы «по сторонам света» (запад, восток, север, юг и центр). Элемент имеющий расположение CENTER занимает БОльшую часть окна, то есть при растяжении формы, сторонние элементы не будут менять размер, а центральный будет растягиваться чтобы занять всю имеющуюся площадь.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setTitle("Test Window");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        setBounds(300, 300, 400, 400);
        JButton button = new JButton("Button 1 (PAGE_START)");
        add(button, BorderLayout.PAGE_START);
        button = new JButton("Button 2 (CENTER)");
        button.setPreferredSize(new Dimension(200, 100));
        add(button, BorderLayout.CENTER);
        button = new JButton("Button 3 (LINE_START)");
        add(button, BorderLayout.LINE_START);
        button = new JButton("Long-Named Button 4 (PAGE_END)");
        add(button, BorderLayout.PAGE_END);
        button = new JButton("5 (LINE_END)");
        add(button, BorderLayout.LINE_END);
        setVisible(true);
    }
}
```

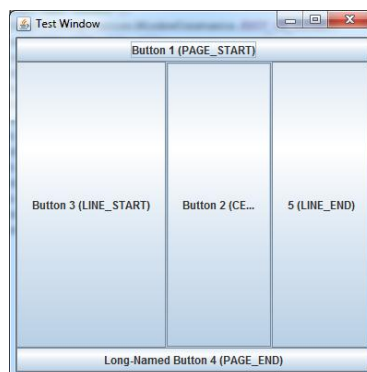


Рисунок 3 – BorderLayout

**BoxLayout** – располагает элементы в строку или столбец, в зависимости от используемой константы: `BoxLayout.Y_AXIS` для расположения элементов в столбец, `BoxLayout.X_AXIS` - в строку.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setBounds(500,500,500,300);
        setTitle("BoxLayoutDemo");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JButton[] jbs = new JButton[10];
        setLayout(new BoxLayout(getContentPane(), BoxLayout.Y_AXIS)); // одну из
// строк надо закомментировать
        setLayout(new BoxLayout(getContentPane(), BoxLayout.X_AXIS)); // одну из
// строк надо закомментировать
        for (int i = 0; i < jbs.length; i++) {
            jbs[i] = new JButton("#" + i);
            jbs[i].setAlignmentX(CENTER_ALIGNMENT);
            add(jbs[i]);
        }
        setVisible(true);
    }
}
```

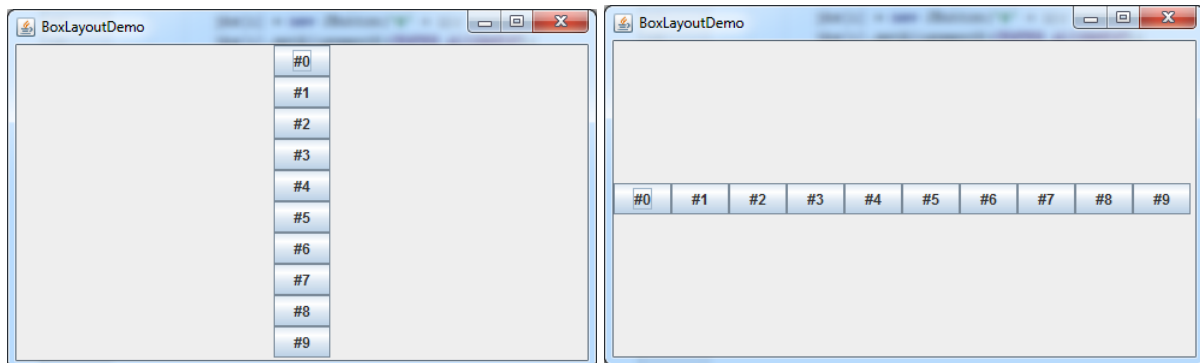


Рисунок 4 – BoxLayout

**FlowLayout** – располагает элементы в одну строку, когда ширины строки становится недостаточно, переносит новые элементы на следующую.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setBounds(500, 500, 400, 300);
        setTitle("FlowLayoutDemo");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JButton[] jbs = new JButton[10];
        setLayout(new FlowLayout());
        for (int i = 0; i < jbs.length; i++) {
            jbs[i] = new JButton("#" + i);
            add(jbs[i]);
        }
        setVisible(true);
    }
}
```

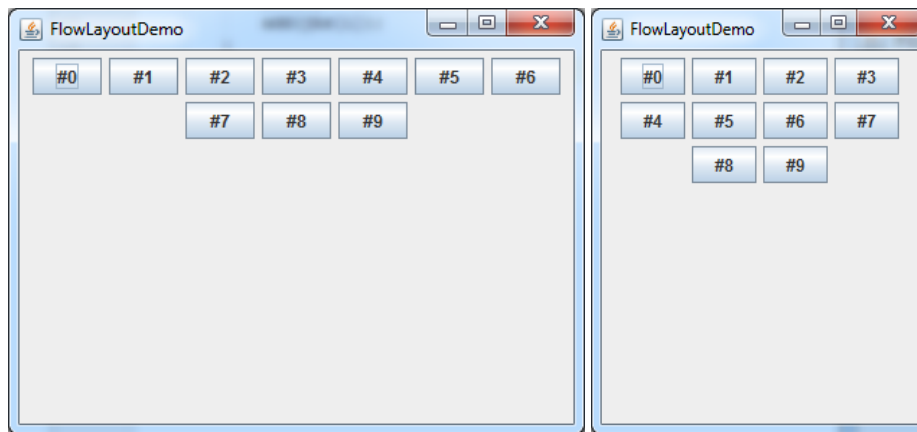


Рисунок 5 – FlowLayout

**GridLayout** – элементы управления выравниваются по таблице заданного размера. Все строки имеют одинаковую высоту, все столбцы – длину. Элемент управления может занимать только одну ячейку таблицы.

```
public class MyWindow extends JFrame {
    public MyWindow() {
        setBounds(500, 500, 400, 300);
        setTitle("GridLayoutDemo");
        setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
        JButton[] jbs = new JButton[10];
        setLayout(new GridLayout(4, 3));
        for (int i = 0; i < jbs.length; i++) {
            jbs[i] = new JButton("#" + i);
            add(jbs[i]);
        }
        setVisible(true);
    }
}
```

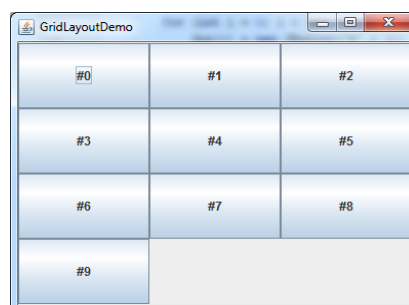


Рисунок 6 – GridLayout

**Отключение компоновщика элементов** – возможен вариант ручной расстановки элементов путем указания их абсолютных координат и размеров с помощью метода `setBounds()`, для чего необходимо выключить компоновщик элементов через `setLayout(null)`.

## Пример использования базовых элементов интерфейса:

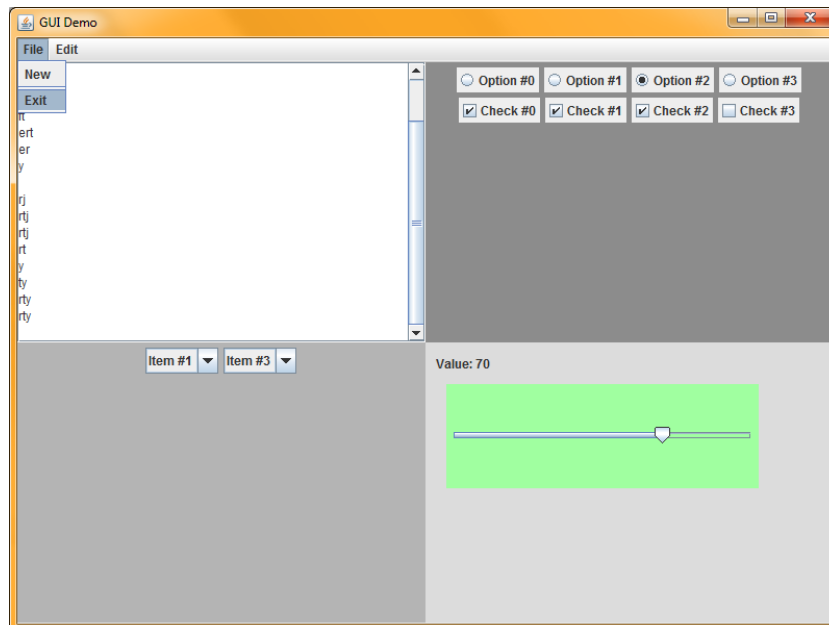


Рисунок 7 – Использование различных элементов интерфейса

В первом блоке кода настраиваем параметры окна и создаем четыре панели для размещения элементов, при этом задаем им отличный друг от друга цвет. Располагаем эти панели на форме в виде таблицы (2, 2), с помощью GridLayout.

```
setBounds(500, 200, 800, 600);
setTitle("GUI Demo");
setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
setLayout(new GridLayout(2, 2));
JPanel[] jp = new JPanel[4];
for (int i = 0; i < 4; i++) {
    jp[i] = new JPanel();
    add(jp[i]);
    jp[i].setBackground(new Color(100 + i * 40, 100 + i * 40, 100 + i * 40));
}
```

На первой панели расположено многострочное текстовое поле, которое находится внутри элемента JScrollPane, что позволяет пролистывать контент этого поля.

```
jp[0].setLayout(new BorderLayout());
JTextArea jta = new JTextArea();
JScrollPane jsp = new JScrollPane(jta);
jp[0].add(jsp);
```

Во второй панели содержится 2 типа элементов: JCheckBox и JRadioButton. JCheckBox предназначен для вкл/выкл каких-либо опций (флажков), при этом одновременно может быть включено несколько JCheckBox. JRadioButton предоставляет выбор только одного пункта из набора, т.е. может быть выбрана только одна опция. Для корректной работы связанных RadioButton их необходимо заносить в ButtonGroup.

```
jp[1].setLayout(new FlowLayout());
JRadioButton[] jrb = new JRadioButton[4];
ButtonGroup bgr = new ButtonGroup();
for (int i = 0; i < jrb.length; i++) {
```

```

        jrb[i] = new JRadioButton("Option #" + i);
        bgr.add(jrb[i]);
        jp[1].add(jrb[i]);
    }
    JCheckBox[] jcb = new JCheckBox[4];
    for (int i = 0; i < jcb.length; i++) {
        jcb[i] = new JCheckBox("Check #" + i);
        jp[1].add(jcb[i]);
    }
}

```

На третьей панели расположена пара элементов типа JComboBox, которые представляют собой выпадающие списки. ActionListener для JComboBox проверяет событие выбора пользователем одного из пунктов.

```

jp[2].setLayout(new FlowLayout());
String[] comboStr = {"Item #1", "Item #2", "Item #3", "Item #4"};
JComboBox<String> jcombo1 = new JComboBox<String>(comboStr);
JComboBox<String> jcombo2 = new JComboBox<String>(comboStr);
jp[2].add(jcombo1);
jp[2].add(jcombo2);
jcombo1.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println(jcombo1.getSelectedItem().toString());
    }
});

```

Четвертая панель представляет собой пример расстановки элементов с использованием абсолютных координат. На ней расположено обычное не редактируемое текстовое поле, которое показывает значение, выбранное на JSlider.

```

jp[3].setLayout(null);
JSlider js = new JSlider();
JLabel jlab = new JLabel("Value: 50");
js.setMaximum(100);
js.setMinimum(0);
js.setValue(50);
jp[3].add(jlab);
jp[3].add(js);
js.addChangeListener(new ChangeListener() {
    @Override
    public void stateChanged(ChangeEvent e) {
        jlab.setText("Value: " + js.getValue());
    }
});
jlab.setBounds(10, 10, 100, 20);
js.setBounds(20, 40, 300, 100);
js.setBackground(new Color(160, 255, 160));

```

Последним пунктом идет создание верхнего меню окна. Для этого создаются элементы JMenuBar -> JMenu -> JMenuItem. Как видно из кода ниже, для обработки нажатия на один из подпунктов меню, достаточно «повесить» на него ActionListener.

```

JMenuBar mainMenu = new JMenuBar();
JMenu mFile = new JMenu("File");
JMenu mEdit = new JMenu("Edit");
JMenuItem miFileNew = new JMenuItem("New");
JMenuItem miFileExit = new JMenuItem("Exit");

```



```

setJMenuBar(mainMenu);

mainMenu.add(mFile);
mainMenu.add(mEdit);

mFile.add(miFileNew);
mFile.addSeparator(); // разделительная линия в меню
mFile.add(miFileExit);

miFileExit.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.exit(0);
    }
});

addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        System.out.println("BYE");
    }
});

```

## Обработка событий

Обработка событий является неотъемлемой частью разработки прикладных программ с графическим пользовательским интерфейсом (ГПИ). Любая прикладная программа с ГПИ, выполняется под управлением событий, большинство которых направлено на взаимодействие с пользователем. Существует несколько типов событий, включая генерируемые мышью, клавиатурой, различными элементами интерфейса. Рассмотрим некоторые варианты обработки событий.

### Обработка кликов по кнопке

```

JButton button = new JButton("Button");
add(button);
button.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button pressed...");
    }
});

```

Для отслеживания кликов по кнопке необходимо добавить ActionListener, как показано выше. Как только произойдет событие нажатия этой кнопки, выполнится метод actionPerformed().

### Обработка нажатия кнопки Enter в текстовом поле

```

JTextField field = new JTextField();
add(field);
field.addActionListener(new ActionListener() {
    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println("Your message: " + field.getText());
    }
});

```

При работе с текстовым полем (TextField) ActionListener отлавливает нажатие кнопки Enter, конечно только в случае если поле находится в фокусе. Поэтому нет необходимости отслеживать именно нажатие кнопки Enter, например через addKeyListener(...), с указанием кода этой клавиши.

#### Отслеживание кликов мыши по панели и определение координат клика:

```
JPanel panel = new JPanel();
add(panel);
pan.addMouseListener(new MouseAdapter() {
    @Override
    public void mouseReleased(MouseEvent e) {
        System.out.println("MousePos: " + e.getX() + " " + e.getY());
    }
});
```

В приведенном примере, отслеживание отжатия кнопки мыши над объектом типа JPanel. В объект e типа MouseEvent попадает вся информация о произошедшем событии, в том числе: координатах курсора в системе координат панели, кнопке которая была нажата (левая или правая), количестве кликов (одинарный, двойной) и т.д.

#### Отслеживание момента закрытия окна:

```
addWindowListener(new WindowAdapter() {
    @Override
    public void windowClosing(WindowEvent e) {
        System.out.println("Bye");
    }
});
```

Чтобы выполнить какое-либо действие при закрытии окна, необходимо к объекту типа JFrame добавить WindowListener. В примере выше ссылка на объект отсутствует, так как метод прописан в конструкторе класса, наследуемого от JFrame.

## Домашнее задание

Создать окно для клиентской части чата: большое текстовое поле для отображения переписки в центре окна. Однострочное текстовое поле для ввода сообщений и кнопка для отсылки сообщений на нижней панели. Сообщение должно отсылаться либо по нажатию кнопки на форме, либо по нажатию кнопки Enter. При «отсылке» сообщение перекидывается из нижнего поля в центральное.

## Дополнительные материалы

- 1 Кей С. Хорстманн, Гари Корнелл Java. Библиотека профессионала. Том 1. Основы // Пер. с англ. - М.: Вильямс, 2014. - 864 с.
- 2 Брюс Эккель Философия Java // 4-е изд.: Пер. с англ. — СПб.: Питер, 2016. — 1168 с.
- 3 Г. Шилдт Java 8. Полное руководство // 9-е изд.: Пер. с англ. - М.: Вильямс, 2015. - 1376 с.
- 4 Г. Шилдт Java 8: Руководство для начинающих. // 6-е изд.: Пер. с англ. - М.: Вильямс, 2015. - 720 с.