Написание приложения с графическим интерфейсом

Основы работы с библиотекой Swing. Основное окно приложения, элементы управления и их компоновка. Написание приложения, сборка проекта

[Создание формы](#_tyjcwt)

[Обработка событий](#_vphgzphqfkfo)

[Обработка кликов по кнопке](#_u3gytzw1y935)

[Обработка нажатия кнопки Enter в текстовом поле](#_2k8sdcktzw5t)

[Отслеживание кликов мыши по панели о определение координат клика](#_rephijgb5ac4)

[Поддержка графики](#_2s8eyo1)

[Домашнее задание](#_3rdcrjn)

[Дополнительные материалы](#_35nkun2)

Используемая литература

# 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) {  MyWindow myWindow = 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 формы; setBounds() – устанавливает координаты формы и ее размер в пикселях; setVisible(true) – показывает полученную форму на экране, желательно вызывать этот метод после настроек формы, иначе при запуске некоторые элементы могут быть отображены с искажениями.

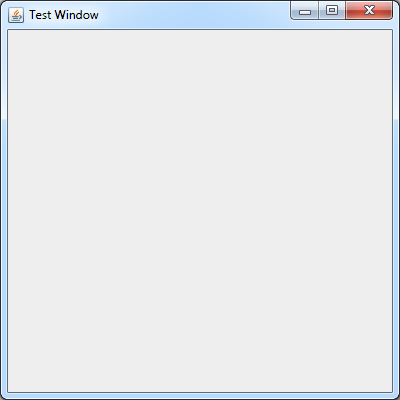


Рисунок 1 – Пример пустого окна Swing

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);  } } |

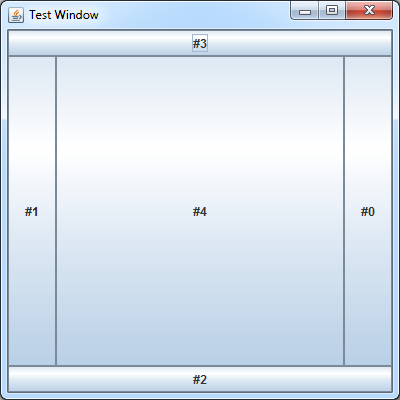


Рисунок 2 – Форма с 5 кнопками

За расстановку элементов на форме отвечают компоновщики элементов, здесь 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);  } } |

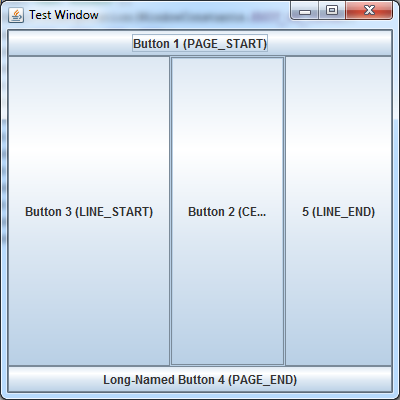


Рисунок 3 – Пример BorderLayout

**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);  } } |

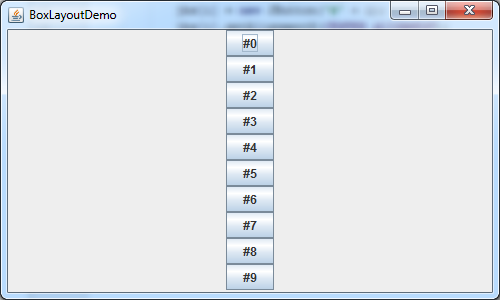
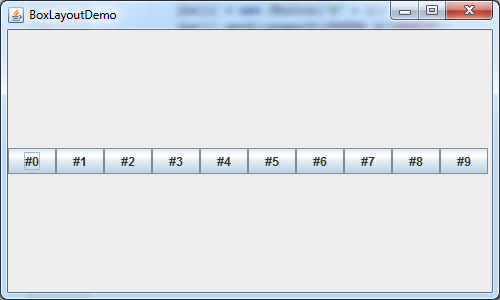
 

Рисунок 4 – Пример BoxLayout

**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);  } } |

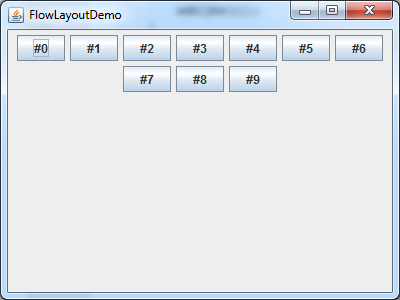
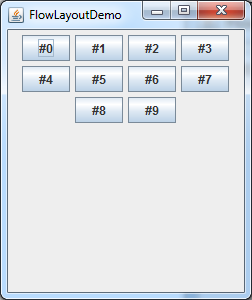
 

Рисунок 5 – Пример FlowLayout

**GridLayout –** элементы управления выравниваются по таблице заданного размера.

|  |
| --- |
| 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);  } } |

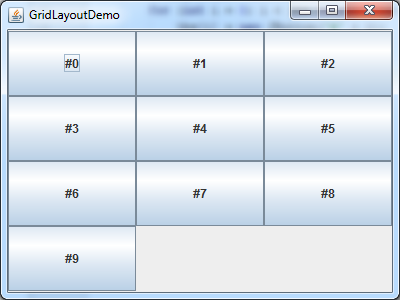


Рисунок 6 – Пример GridLayout

Возможен сценарий ручной расстановки элементов путем указания их абсолютных координат. Для этого необходимо указать setLayout(null) и для каждого элемента указать его координаты и размеры с помощью метода setBounds().

**Базовые элементы управления**: JFrame – окно; JButton – кнопка; JLabel – надпись; JTextField – однострочное текстовое поле; JTextArea – многострочное текстовое поле; JScrollPane – контейнер для пролистывания контента; JMenuBar – верхнее меню программы; JTabel – таблица; JRadioButton – RadioButton; JCheckBox – CheckBox.

# 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 взаимодействии пользователя с этой программой. Существует несколько типов событий, включая генерируемые мышью, клавиатурой, различными элементами управления ГПИ. 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 произойдёт событие нажатия этой кнопки, выполнится метод actionPerformed().

## Обработка нажатия кнопки Enter в текстовом поле

|  |
| --- |
| public class MyWindow extends JFrame {  public MyWindow() {  setBounds(500, 500, 400, 300);  setTitle("Demo");  setDefaultCloseOperation(WindowConstants.*EXIT\_ON\_CLOSE*);  JTextField field = new JTextField();  add(field);  field.addActionListener(new ActionListener() {  @Override  public void actionPerformed(ActionEvent e) {  System.*out*.println("Your message: " + field.getText());  }  });  setVisible(true);  } } |

При работе с текстовым полем ActionListener отлавливает нажатие кнопки Enter, 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 о определение координат клика

|  |
| --- |
| public class MyWindow extends JFrame {  public MyWindow() {  setBounds(500, 500, 400, 300);  setTitle("Demo");  setDefaultCloseOperation(WindowConstants.*EXIT\_ON\_CLOSE*);  JPanel pan = new JPanel();  add(pan);  pan.addMouseListener(new MouseAdapter() {  @Override  public void mouseReleased(MouseEvent e) {  System.*out*.println("MousePos: " + e.getX() + " " + e.getY());  }  });  setVisible(true);  } } |

# Поддержка графики

Существует множество методов для работы с графикой на элементах управления. Вывод графики осуществляется в определенном графическом контексте, инкапсулируемом в классе Graphics. Доступ к графическому контексту элемента можно получить двумя способами.

* Путём передачи в качестве аргумента методу, например, paint() или update().
* Методом getGraphics() из класса Component .

Среди прочего в классе Graphics определяется ряд методов для рисования различных графических объектов, в том числе линий, прямоугольников и дуг. В одних случаях графические объекты рисуются только по контуру, в других – дополнительно заполняются цветом. Графические объекты рисуются и заливаются текущим выбранным цветом, которым по умолчанию является черный (для смены цвета необходимо вызвать метод setColor()). Если рисуемый графический объект выходит за пределы окна, он автоматически усекается. Начало отсчета находится в верхнем левом углу окна и имеет координаты (0, 0), указываемые в пикселях.

**Рисование линий.** Линии рисуются методом drawLine(), общая форма которого приведена ниже.

*void drawLine (int началоХ, int началоY, int конецХ, int конецY);*

Метод drawLine() рисует линию текущим цветом от точки с координатами началоX, началоY к точке с координатами конецХ, конецY.

**Рисование прямоугольников.** Методы drawRect() и fillRect() рисуют контурный и заполняемый прямоугольники соответственно. Ниже приведены их общие формы. При рисовании прямоугольников необходимо указать координаты их верхнего левого угла, ширину и высоту.

**Рисование эллипсов и окружностей.** Для рисования эллипса служит метод drawOval(), а для его заливки – метод fillOvаl(). Эллипс рисуется внутри ограничивающего прямоугольника, для которого задаются координаты верхнего левого угла и его размеры (ширина и высота).

**Работа с цветом**

Работа с цветом осуществляется через Color, в котором определено несколько констант (вроде Color.blасk) для описания наиболее употребительных используемых цветов, и конструкторы для создания своих цветов.

*Color (int R, int G, int B)*

*Color (float R, float G, float B)*

Первый конструктор данного класса принимает три аргумента, задающие цвет в определённом сочетании красной, зелёной и синей составляющих в пределах от 0 до 255. Второй работает по аналогии только в формате с плавающей точкой в пределах от 0.0f до 1.0f.

|  |
| --- |
| Color lightRedInt = new Color(255, 127, 127); Color lightRedFloat = new Color(1.0f, 0.5f, 0.5f); |

# Домашнее задание

1. Доработать проект. Уточнение на странице занятия.

# 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.

# Используемая литература

1. Брюс Эккель Философия Java // 4-е изд.: Пер. с англ. – СПб.: Питер, 2016. – 1168 с.
2. Г. Шилдт Java 8. Полное руководство // 9-е изд.: Пер. с англ. – М.: Вильямс, 2015. – 1376 с.
3. Г. Шилдт Java 8: Руководство для начинающих. // 6-е изд.: Пер. с англ. – М.: Вильямс, 2015. – 720 с.