

Министерство образования и науки РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«Воронежский государственный архитектурно-строительный университет»

РАЗРАБОТКА ГРАФИЧЕСКОГО ИНТЕРФЕЙСА ПОЛЬЗОВАТЕЛЯ

Методические указания

*к выполнению лабораторных работ по дисциплине «Технологии
программирования» и «Программная инженерия» для студентов бакалавриата
направления 09.03.02 «Информационные системы и технологии»*

Воронеж 2016

УДК 004.424
ББК 32.973.26-018

Составитель О.В. Минакова

Разработка графического интерфейса пользователя: метод. указания к выполнению лабораторных работ для студ. бакалавриата направления 09.03.02 «Информационные системы и технологии» / Воронежский ГАСУ ; сост.: О.В. Минакова – Воронеж, 2016. – 44 с.

Приводятся основные положения разработки графического интерфейса на базе библиотеки Swing и языка Java. Даются рекомендации и последовательность решения задач по разработке программных приложений с использованием IDE Eclipse и Java SDK.

Предназначены для выполнения лабораторных работ по дисциплине «Технология программирования» студентов бакалавриата направления 09.03.02 «Информационные системы и технологии», а также дисциплине «Программная инженерия» студентов бакалавриата направления 09.03.03 «Прикладная информатика» всех форм обучения.

Ил. 22. Табл. 1. Библиогр. 6 назв.

УДК 004.424
ББК 32.973.26-018

*Печатается по решению учебно-методического совета
Воронежского ГАСУ*

Рецензент – О.Е. Елфимова, доцент кафедры информатики и графики Воронежского государственного архитектурно-строительного университета

ВВЕДЕНИЕ

Простой, удобный и элегантный интерфейс пользователя стал неотъемлемой частью всякого успешного программного продукта. Многооконные системы позволяют выполнять одновременно несколько работ, с помощью мыши легко выбирать и управлять задачами, меню кратко и компактно описывает все возможности выбора, значки представляют важные понятия, рисунки актуализируют информацию, и т. д. GUI (Graphical User Interface – Графический Интерфейс Пользователя) служит обобщением такого стиля взаимодействия. Связанные с ним термины – WYSIWYG (What You See Is What You Get – Что Видите, То и Имеете), WIMP (Windows, Icons, Menus, Pointing device – Окна, Значки, Меню, Указатели) и фраза "прямое манипулирование" – характеризуют приложения, у пользователей которых создается впечатление, что они работают непосредственно с объектами, изображенными на экране. Эти мощные и понятные любому пользователю средства на сегодня являются стандартом для разработки любых информационных систем взаимодействующих с человеком.

Программное приложение, дружественное для пользователя, не должно быть сложным для разработчика. Для удовлетворения нужд разработчиков создано множество библиотек повторно используемых классов, которые поддерживают абстракции данных, описываемых сущностями – окно, меню, контекст, событие, состояние. Компонент Swing – это часть Java Fundamental Class содержит набор классов, предназначенных для создания GUI, удовлетворяющих требованиям конкретных приложений и сред, в которых они выполняются. Для удобства их использования создан специальный инструментарий – конструкторы приложений, которые не только позволяют создавать интерфейс приложения, но и реализовывать различные механизмы взаимодействия как с пользователем, так и с моделью обработки данных

Изложение теоретического материала лабораторного цикла ставит целью показать как хорошая абстракция данных и повторное использование позволяет легко и просто разрабатывать, модифицировать и совершенствовать программные приложения. Выполнение представленных лабораторных работ должно способствовать формированию навыков разработки интерактивных приложений в инструментальной среде Eclipse, а также развитию навыков объектно-ориентированного программирования на языке Java.

Методические указания предназначены для работы в Eclipse IDE с SDK JavaSE.

1. ЗНАКОМСТВО С ECLIPSE

Цель работы: Получение практических навыков работы с инструментарием разработчика Eclipse с SDK JavaSE путем написания простой программы

1.1 Теоретические сведения

Консольное приложение принимает входные и отправляет выходные символьные данные на консоль (стандартный поток ввода/вывода), которая также называется командной строкой. Консольное приложение можно также использовать для демонстрации функциональных возможностей, которые позже будут реализованы в приложениях с любым платформоориентированным графическим интерфейсом, а также для быстрой отладки программных моделей.

Взаимодействие с консолью в Java осуществляется с помощью стандартных потоков (объекта класса System) System.in и System.out.

1. Для вывода информации на консоль следует обратиться к системе, указав направление вывода, и вызвать функцию печати с любым значением аргумента, например строка «Hello»:

```
System.out.println("Hello");
```

2. Для ввода данных с консоли можно использовать методы класса Scanner, которые собирают и интерпретируют введенные пользователем символы. Так метод nextLine() позволяет считывать введенный набор символов до нажатия <Enter> и возвращает объект типа String

Объявление объекта	<i>Scanner userInput;</i>
Создание объекта	<i>userInput = new Scanner(System.in);</i>
Вызов метода	<i>userInputScanner.nextLine();</i>
Объявление объекта и присвоение ему значения	<i>String userInput = userInput.nextLine();</i>

Вся информация вводимая и выводимая на консоль представляет собой множество символом и задача интерпретации ее возлагается на программиста. Варианты преобразования строки arg в число value основываются на использовании статических объектов классов-оболочек Integer, Float, Double, Long в зависимости от используемой переменной:

```
int value1 = Integer.parseInt(arg); // возвращает int  
int value2 = Integer.valueOf(arg); // возвращает Integer  
int value3 = Integer.decode(arg); // возвращает Integer
```

Обратное преобразование из типизированного значения (в частности int) в строку можно выполнить следующими способами:

```
String arg1 = Integer.toString(value1);  
String arg2 = String.valueOf(value1);
```

1.2 Задания

1. Разработайте программу тренировки навыков устного счета.

Программа должна генерировать выражение на сложение двух целых чисел, получать от пользователя его ответ и сравнивать его с правильным результатом.

2. Организуйте вывод/ввод значений до трех неверных ответов.

3. Реализуйте возможность выбора одного из трех уровней игры:

А) нулевой уровень – однозначные числа;

Б) первый – двухзначные;

В) второй – трехзначные.

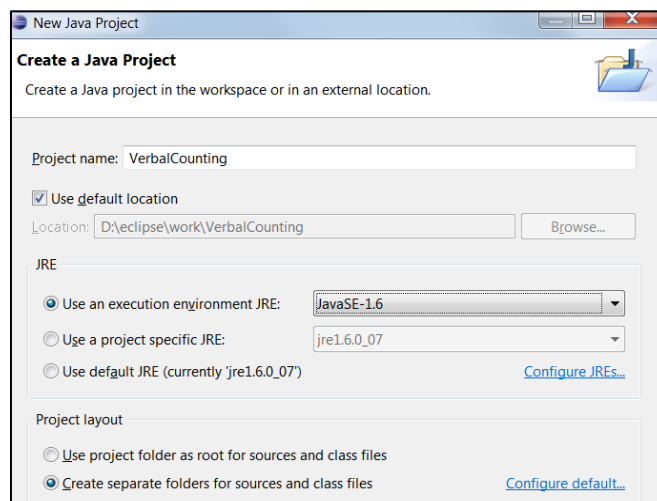
4. Усложните сценарий игры, введя счет (например, 15:6 в пользу компьютера) и временное ограничение (1, 3, 5 минут в зависимости от уровня).

1.3. Указания по созданию программы в среде Eclipse

1. Чтобы начать создание первого Java-проекта, выберите **File->New->Project...** В левом списке появившегося мастера выберите **Java Project**, затем нажмите кнопку **Next**.

2. Введите название проекта и снова нажмите **Next**. Располагаться проект будет в директории, установленной как **Workspace** при настройке Eclipse.

Другие установки этого окна определяют, что будет использоваться JRE, установленная по умолчанию в операционной системе, и исходные файлы будут помещены в отдельную папку "src".



Последний шаг мастера **JavaSetting** позволяет указать место для хранения исходных и скомпилированных файлов, а также задать любые подпроекты и библиотеки, которые могут понадобиться для работы и компоновки текущего проекта.

4. Щелкните на кнопке **Finish** и Eclipse создаст новый проект.

Новый внешний вид носит название Перспективы Java. Перспектива, в терминах Eclipse, это сохраняемый внешний вид окна, включающий любое число редакторов (editors) и представлений (views). В Eclipse по умолчанию включено несколько перспектив (Java, Debug, Resource и так далее), которые можно настраивать, а также создавать новые. Перспективы управляются с помощью элементов меню **Window** или панели инструментов, которая обычно располагается вдоль левой границы окна Eclipse.

Для создания папок, в которых будет содержаться исходный код, в представлении Навигатора (Package Explorer) раскройте дерево папок и выберите

src, а затем пункт меню **File->New->Folder**. В открывшемся диалоге убедитесь, что выбрана нужная папка и введите **com** в поле **Folder Name**.

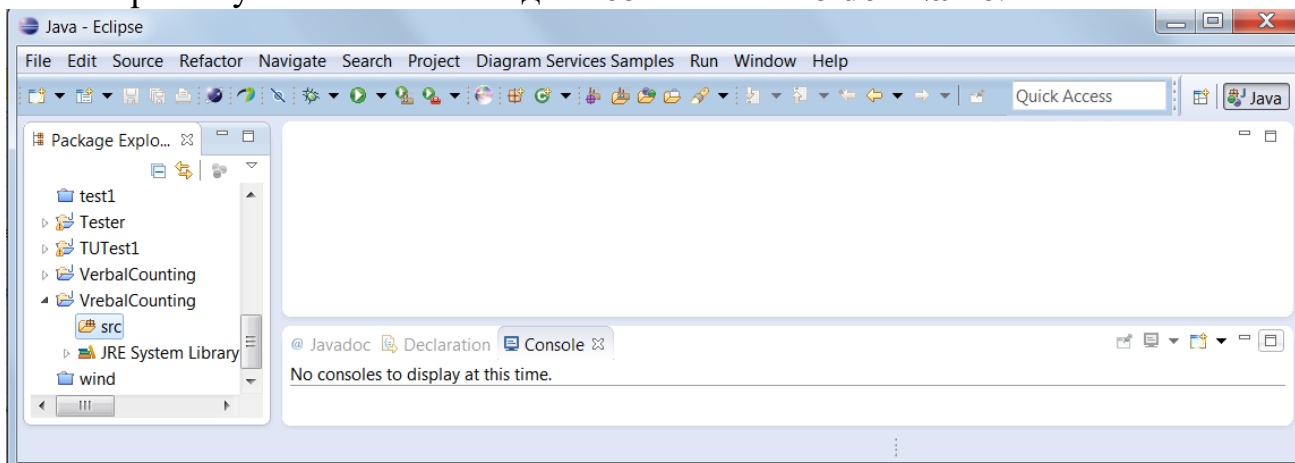


Рис.1. Перспектива Java среды Eclipse

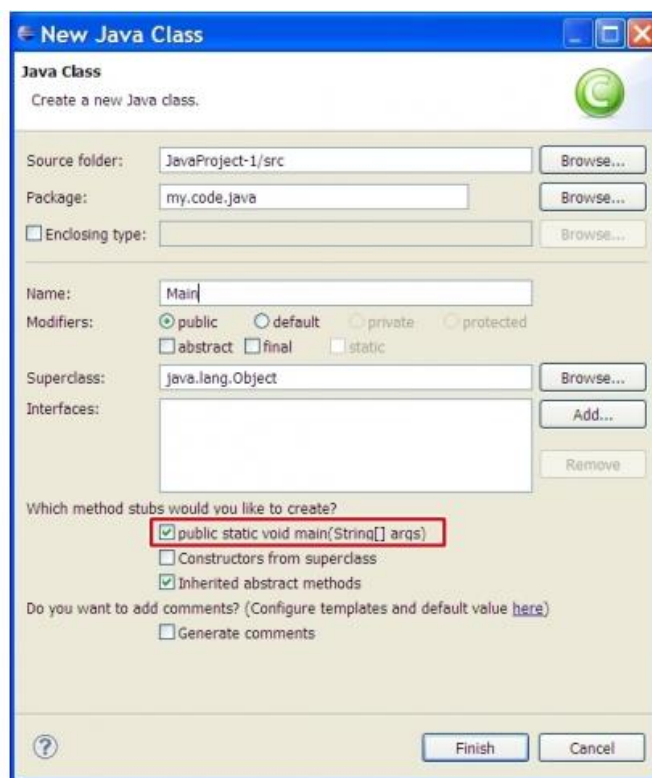
Для создания большого числа папок следует использовать Windows Explorer и другие средства ОС. Воспользуйтесь одним из этих методов и создайте подпапки **gui** и **model** в папке **com**, после этого выберите любую папку в представлении Навигатора, а затем пункт меню **File->Refresh**. Eclipse просмотрит файловую систему и приведет проект в соответствие с внесенными изменениями.

5. Создайте главный класс программы, он должен быть **public** - общедоступный и содержать функцию **main** - точку входа в программу. Для этого щелчком правой кнопкой мыши на пакете выбрать **New -> Class**. То же можно проделать, нажав кнопку **New Java Class** на панели инструментов.

В диалоговом окне создания класса задается любое имя, согласно правилам именования классов в Java.

Отметьте опцию "public static void main(String[] args)", чтобы IDE создало одноименную функцию.

Для завершения – **Finish**.



6. В появившемся окне редактора, по метке **TODO?** введите код программы. Пользуйтесь автодополнением – после начала ввода кода следует нажать <Ctrl + Пробел>. Для сохранения изменений – <Ctrl + S>.

Важно помнить, что меню Eclipse отображается по-разному в зависимости от текущей перспективы. Создать нужное представление можно из

пункта меню *Window->Show View*, найти ранее созданное – *Window->OpenPerspective*. Чтобы закрепить сделанное изменение, используйте команду меню *Window->Save Perspective As...*

1.4. Указания по запуску и отладке программы в среде Eclipse

1. Из меню **Run** выберите элемент **Run...** В появившемся диалоге из списка Конфигураций Запуска (**Launch Configurations**) выберите "Java Application", затем нажмите на кнопку **New**.

3. Введите *Calculator* в качестве имени (**Name**) новой конфигурации.

4. Нажмите кнопку **Search...** и выберите *Calculator* в качестве главного класса (**Main Class**), введите в текстовом поле *com.delirious.calculator.Calculator*.

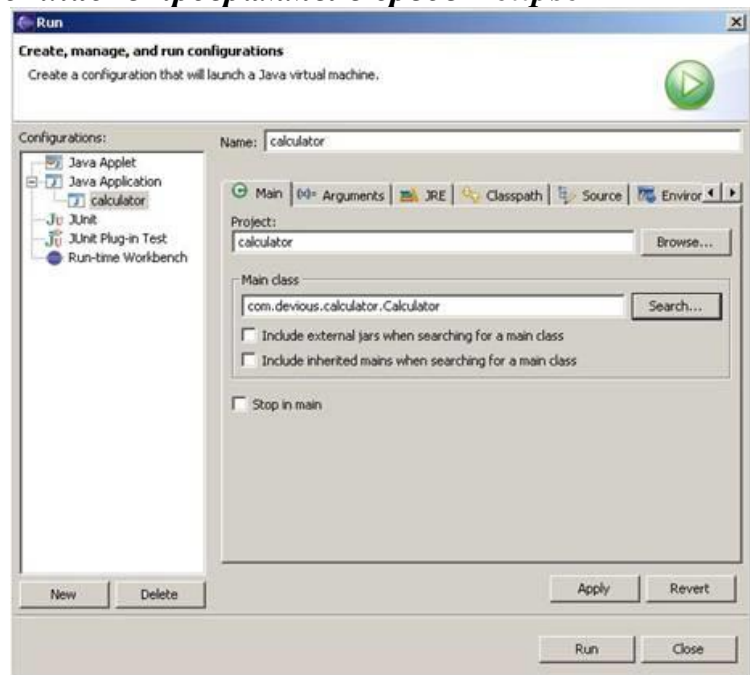


Рис. 2. Создание конфигурации проекта

5. Щелкните кнопку **Run** для сохранения конфигурации и запуска приложения.

После создания конфигурации запуска приложения, его многократно запускать многократно с использованием элемента меню **Run->Run History** или кнопки **Run** на панели инструментов.

6. Для запуска приложения в отладчике выберите в меню **Run->Debug** и Eclipse автоматически перейдет к перспективе Отладки (Debug) представленной на рис. 3.

Для установки точек останова выполните двойной щелчок на поле напротив интересующего оператора в коде и запустите выполнение (<F8>). Для мониторинга состояний объектов используется вкладка **Variables**, которая содержит список переменных, задействованных в текущей точке прерывания. Точки прерывания можно отключать и включать в процессе отладки, устанавливая или удаляя маркер напротив нужной точки. Кроме проверки значений переменных в точках останова возможно пошаговое выполнение программы, заход и обход строк кода. Также можно дополнительно установить условия останова на точке прерывания, наведя указатель мыши на нужную точку и выбрав в раскрывающемся контекстном меню **Breakpoint Properties**. В открывшемся окне свойств выбранной точки прерывания установите галочку на **Enable Condition** (Включить Условие) и в поле ввода введите необходимое условие, при котором будет происходить останов на данной точке.

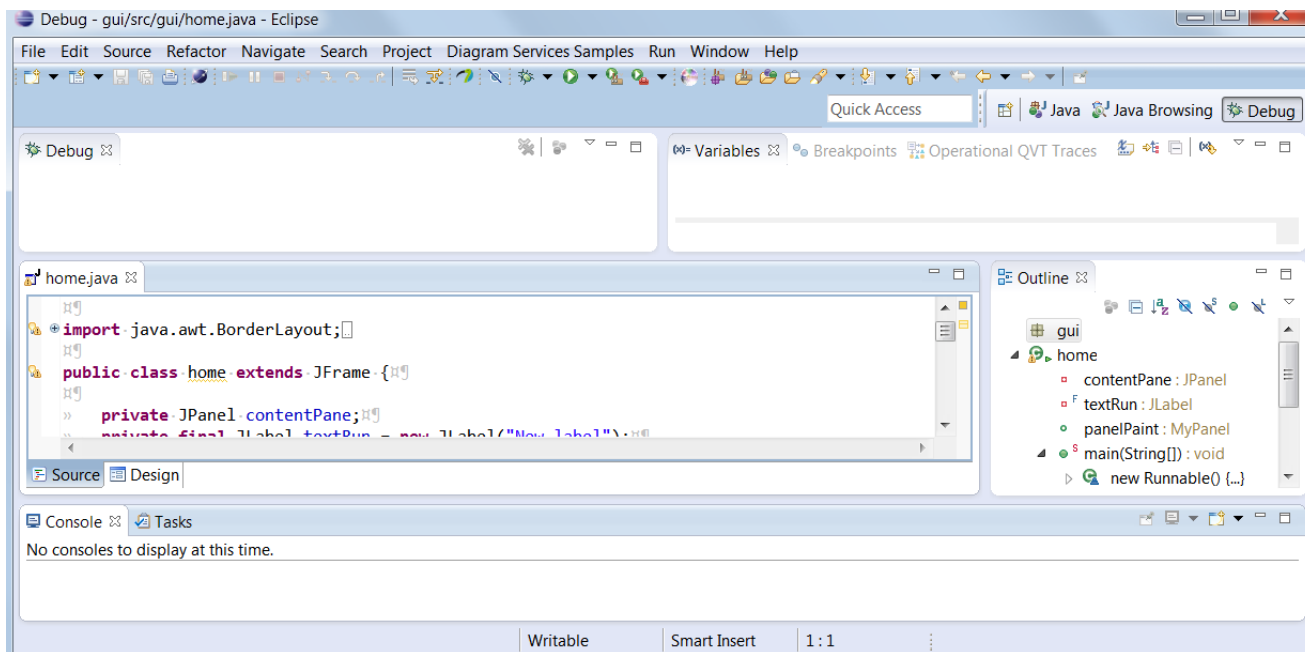


Рис. 3. Перспектива Отладки Eclipse

2. РИСОВАНИЕ ГРАФИЧЕСКИХ ПРИМИТИВОВ В ОКНЕ

Цель работы: Получение практических навыков работы с графикой.

2.1. Теоретическая справка

При отображении пользовательского интерфейса используется подход – «отрисуй себя сам». У каждой графической компоненты есть метод `paint (Graphics)`, который вызывается для перерисовки. Метод этот определен в классе `java.awt.Component`, а все элементы GUI наследуются от него, поэтому он может быть переопределен везде, где необходимо. Используя этот графический контекст, компонент сам выполняет свою отрисовку.

Но рисование осуществляется не на самом компоненте, а на его графическом контексте – объекте класса `Graphics`. Он отражает особенности реализации графической подсистемы любой ОС и содержит информацию необходимую для процесса рисования – участок экрана на котором расположен компонент, его размеры, цвет, толщина линий и т.п. Использование графического контекста позволяет единообразно обращаться ко всем графическим устройствам.

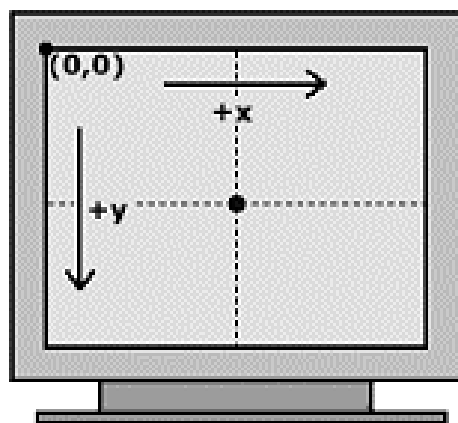


Рис. 4 – Направление вывода

Единицей измерения координат в графическом контексте является логически условная единица – пиксел (`pixel`). Начало координат $(0,0)$ находится в левом верхнем углу экрана, при этом ось $0x$ направлена традиционно (слева

направо), а ось Oy – сверху вниз (рис. 4).

Рисование любой растровой фигуры (по точкам) сводится к следующим шагам.

1. Получение графического контекста.
2. Задание параметров отрисовки.
3. Рисование фигуры.

Для корректного отображения необходимо знать текущий размер компонента, для этого используют метод `getWidth()` и `getHeight()`.

Установить цвет для рисования можно методом `setColor(Color a)`, где статический объект класса `Color` задает цвет.

Основные методы рисования класса `Graphics`:

`drawLine(x1, y1, x2, y2)` – отображает линию толщиной в один пиксел, проходящую из точки (x_1, y_1) в (x_2, y_2) ;

`drawRect(int x, int y, int width, int height)` – отображает прямоугольник, чей левый верхний угол находится в точке (x, y) , а ширина и высота равняются `width` и `height` соответственно;

`drawOval(int x, int y, int width, int height)` – рисует овал, вписанный в прямоугольник, задаваемый указанными параметрами. Очевидно, что если прямоугольник имеет равные стороны (т.е. является квадратом), овал становится окружностью;

`drawString(String text, int x, int y)` – выводит на экран текст, задаваемый первым параметром и где точка (x, y) задает позицию самого левого символа.

`drawPolygon(int[], int[], int)` и `fillPolygon(int[], int[], int)` – рисует контур многоугольника (ломаную линию), задаваемого двумя массивами, содержащими `x` и `y` координаты вершин, третий параметр метода - число пар координат.

Для отрисовки закрашенных фигур используют аналогичные методы – `fillRect`/`fillOval`/`fillPolygon`.

2.2. Задания и указания к их выполнению

1. Создайте оконное приложение.

Каждая GUI-программа запускается в окне и по ходу работы может открывать несколько дополнительных окон.

В библиотеке `Swing` описан класс `JFrame`, представляющий собой окно с рамкой и строкой заголовка (с кнопками «Свернуть», «Во весь экран» и «Закрыть»). Оно может изменять размеры и перемещаться по экрану.

Конструктор `JFrame()` без параметров создает пустое окно. Конструктор `JFrame(String title)` создает пустое окно с заголовком **title**.

Для написания простейшей программы, выводящей на экран пустое окно, требуются следующие методы:

`setSize(int width, int height)` – устанавливает размеры окна. Размеры окна включают не только «рабочую» область, но и границы и строку заголовка.

`setDefaultCloseOperation(int operation)` – позволяет указать действие, которое необходимо выполнить, когда пользователь закрывает окно нажатием

на крестик. Для того, чтобы при закрытии окна программа прекращала работу, следует в качестве параметра `operation` передать константу `EXIT_ON_CLOSE`, описанную в классе `JFrame`.

`setVisible(boolean visible)` – когда окно создается, оно по умолчанию невидимо, для отображения окна на экране, метод вызывается с параметром `true`.

Пример программы, которая создает окно, выводит его на экран и завершает работу после того, как пользователь закрывает окно

```
import javax.swing.*;
public class MyClass {
    public static void main (String [] args) {
        JFrame myWindow = new JFrame("Пробное окно");
        myWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        myWindow.setSize(400, 300);
        myWindow.setVisible(true);}
}
```

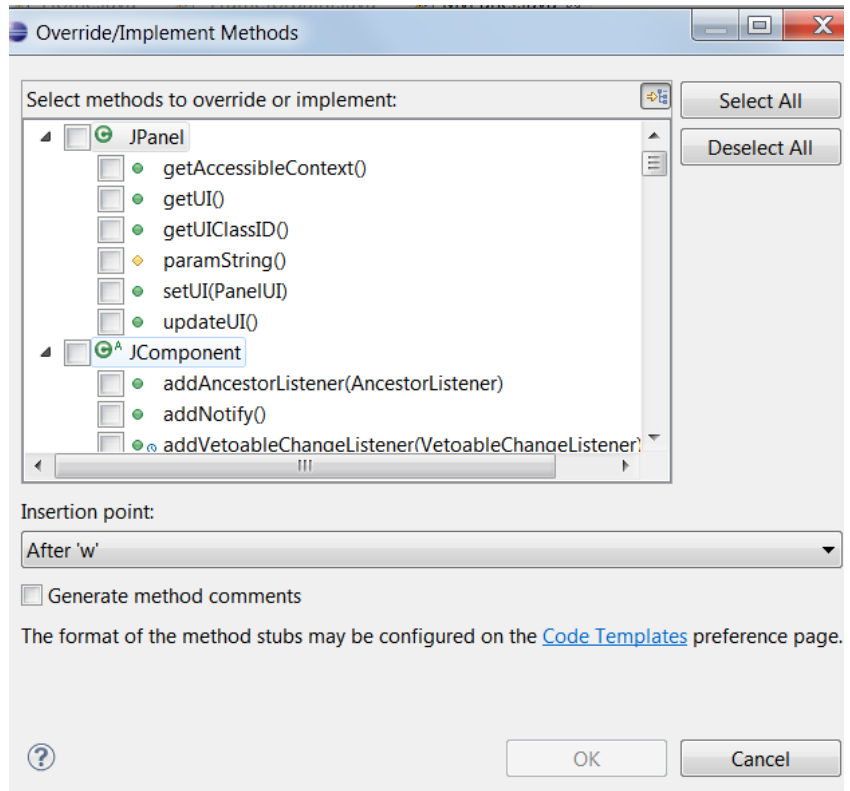
Обратите внимание, для работы с большинством классов библиотеки `Swing` понадобится импортировать пакет `java.swing.*`.

2. Добавьте к классу, создающему окно метод `paint()` и разместите в методе `paint()` следующие графические примитивы:

- 1) контур овала красным цветом;
- 2) закрашенный синим цветом прямоугольник;
- 3) текст «Мне все удалось!!!», размещенный в центре рабочей области.

Для переопределения методов класса следует открыть контекстное меню правой кнопкой мыши и выбрать **Source -> Override/Implement Methods**. В открывшемся окне развернуть список доступных для переопределения методов класса `JFrame`. После чего выбрать необходимые методы из представленного списка.

Для того чтобы изменить имя переменной/класса, следует двойным щелчком левой клавиши



мышью выделить имя, требующее изменения, после чего правой кнопкой вызвать список, выбрать в нем *Source => Refactor => Rename*, ввести новое имя и нажать Enter для фиксации изменений в приложении.

Пример переопределения метода `paint()` текущего окна

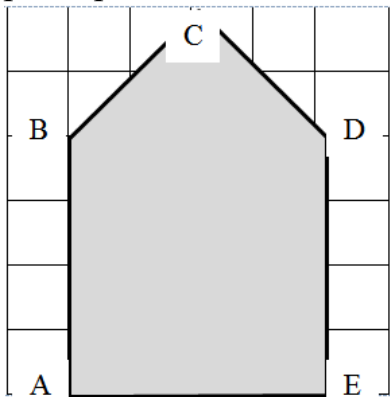
```
JFrame myWindow = new JFrame("Пробное окно") {
    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(Color.BLUE);
        g.fillRect(100, 100, 100, 200);
    }
};
```

3. Создайте тематическое изображение, состоящее не менее чем из 7 геометрических фигур. Пример на рисунке 5.



Рис. 5 Пример изображения

4. Сделайте так, чтобы размеры изображения были пропорциональны размерам окна и изменялись вместе с окном.



Считается, что рисунок должен занимать $2/3$ окна. Поэтому удобно разбить область окна на шесть клеток по горизонтали ($w/6$) и шесть по вертикали ($h/6$), где w – ширина окна по оси x , а h – высота по y . В таком случае, координаты точки $A(w/6, h)$, $B(w/6, 2h/6)$, $C(w/2, 0)$, $D(5w/6, 2h/6)$, $E(5w/6, h)$. Аналогично, можно рассчитать и связать с реальными размерами окна любую фигуру.

Эти точки можно объединить в сложную фигуру, используя объект `Polygon`:

```
Polygon A = new Polygon(); //объявление и создание полигона A
```

И последовательно добавляя каждую точку:

```
A.addPoint(w/6, h); // добавление частей полигона через заданные координаты//
A.addPoint(w/6, 2h/6);
A.addPoint(w/2, 0);
A.addPoint(5w/6, 2h/6);
A.addPoint(5w/6, h);
A.addPoint(w/6, h);
```

В классе `Graphics` предусмотрены методы для рисования как контура, так и закрашенной фигуры:

```

g.setColor(Color.yellow); // задание цвета фигуры
g.fillPolygon(A); //рисование закрашенного фигуры
g.setColor(Color.red); // задание цвета фигуры
g.drawPolygon(A); //рисование контура фигуры

```

Перейдите от представления на фиксированной сетке к относительным координатам, где $5w/6=0,833 w$. Класс для перехода от абсолютных к относительным координатам представлен в приложении А.

2.3. Примеры построения изображений

Пример 1. Вывод 10 диагональных линий

```

public void paint( Graphics g ) {
// задаем цвет для прорисовки
g.setColor(Color.green);
// определяем длину и ширину текущей области отрисовки
width = getSize().width;
height = getSize().height;
// Рисуем 10 диагональных линий в цикле
for ( int i = 0; i < 10; ++i ) {
g.drawLine( width, height, i * width / col, 0 );
//устанавливаем шрифт и выводим текст
g.setFont(new Font("Courier", Font.PLAIN, 12));
g.drawString("Текст внутри окна панели", 10, 50);
}

```

Пример 2. Вывод текста в центр окна

```

void DrawTextCenter(Graphics g, String str){
FontMetrics fm = g.getFontMetrics();
int w = fm.stringWidth(str);
int h = fm.getAscent();
width = g.getSize().width;
height = g.getSize().height;
g.drawString("Swing", width/2-(w/2), width/2+(h/4));
}

```

Пример 3. Отображение рисунка из файла, находящегося в одной папке с классом приложения Example, в котором используется окно myWindow.

```

JFrame myWindow = new JFrame("Пробное окно"){
    Image im;
    boolean loaded = false;
    public void paint(Graphics g) {
        super.paint(g);
        if(!loaded){ try {

            im=ImageIO.read(Example.class.getResource("1.jpg"));
            loaded = true;
        } catch (IOException e){
            e.printStackTrace();
        } }
        g.setColor(Color.BLUE);
        g.drawImage(im, 0, 0,null);}}

```

3. СОЗДАНИЕ ПРОСТОГО GUI-ПРИЛОЖЕНИЯ С КНОПКОЙ

Цель работы: Получение практических навыков работы с конструктором приложений WindowsBuilder.

3.1 Теоретическая справка

Все элементы управления в Swing разделяются на контейнеры и компоненты. Компонент - это базовый элемент графического пользовательского интерфейса, например, кнопка, надпись или текстовое поле. Контейнер отличается от компонента тем, что может содержать в себе другие элементы. Контейнеры являются организующими звеньями и позволяют группировать графические элементы в пространстве графического пользовательского интерфейса. Примером контейнера является окно (рис. 6), поскольку оно может содержать такие компоненты, как кнопки, текстовые метки и т. п.

Каждый визуальный компонент характеризуется:

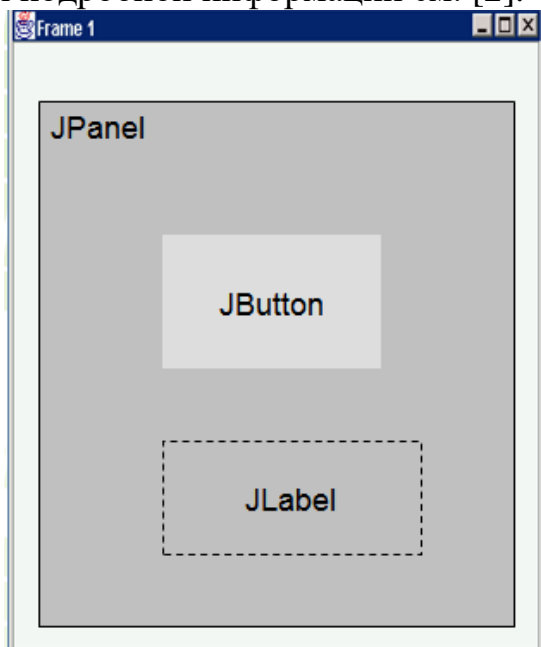
- способом отображения на экране (представление);
- реакцией на действия пользователя (контроллер);
- информацией, связанной с ним (модель).

Для отображения компонентов можно установить множество разных атрибутов, таких как цвет, шрифт и множество других.

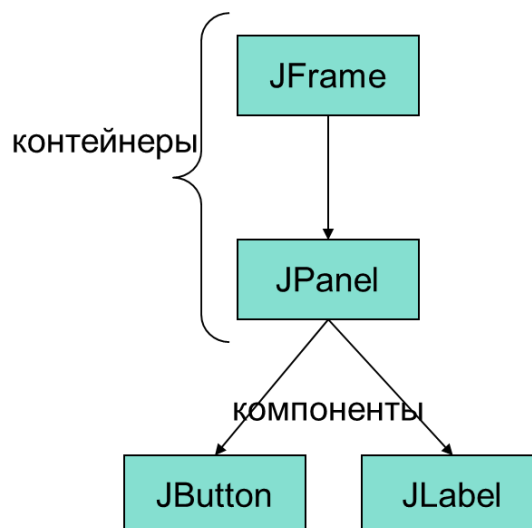
Например, для установки цвета фона и переднего плана используются следующие методы:

```
setBackground(Color); // установка цвета фона компонента  
setForeground(Color); // установка цвета на переднем плане
```

а для размещения надписи на метке (JLabel) – `setText(String)`. Для получения подробной информации см. [2].



Расположение компонентов



Структурная организация

Рис. 6. Внешняя и внутренняя организация окна

Компоненты генерируют события в соответствии с воздействиями поль-

зователя на графический интерфейс. Механизм управления событиями основан на непрерывном «прослушивании» событий и передачи сообщения о их наступлении зарегистрированным компонентам. Источник события оповещает слушателя путем вызова специального полиморфного метода интерфейса слушателя (`actionPerformed`) и передачи ему объекта события (`ActionEvent`). Обработчик события представляется собой объект специального класса, и все слушатели определенного события должны реализовывать соответствующий интерфейс.

Для реализации управления событиями необходимы следующие шаги:

1. создание источника	<code>Button b = new Button("New");</code>
2. регистрация слушателя у источника	<code>b.addActionListener(this);</code>
3. реализация интерфейса слушателя	<pre>public void actionPerformed(ActionEvent e) { System.out.println("Button "+b+" was pressed"); }</pre>

Слушатель может использовать неименованные (анонимные) классы для прослушивания события. Это упрощает код слушателя и позволяет реализовать в одном классе много слушателей однотипных событий:

```
b.addActionListener(new ActionListener() {
    // реализация интерфейса слушателя в анонимном классе
    public void actionPerformed(ActionEvent e) {
        System.out.println("Button "+e.getActionCommand()+
            " was pressed");
    }
});
```

Если в программе необходимо создать множество элементов управления, настроить их внешний вид, разместить в нужных местах окна для этого используется специальный инструментарий – дизайнер форм, например `WindowsBuilder`.

3.2 Рекомендации по использованию `WindowsBuilder` для создания оконного приложения в среде `Eclipse`

1. Для начала нужно создать новый проект в `Eclipse`. Для этого нажмите на вкладку `File`, далее `New => JavaProject`.

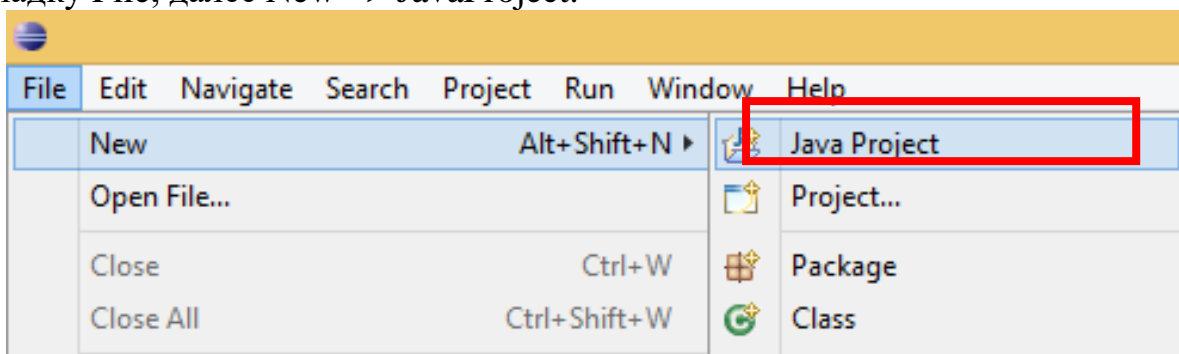


Рис. 7. Создание нового проекта

2. В открывшемся окне укажите имя проекта, например, «GUI_Test»:

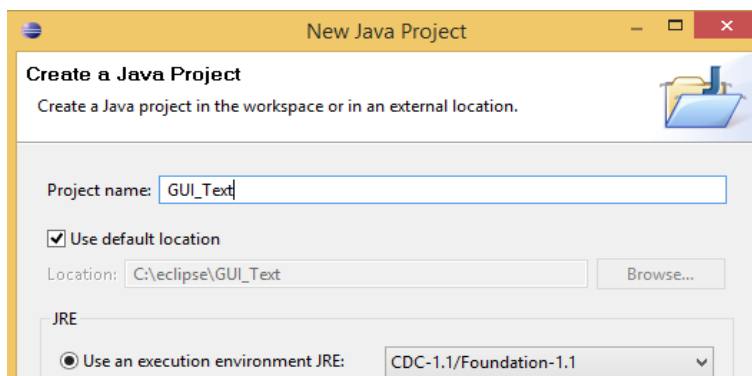


Рис. 8. Создание проекта

3. В Package Explorer выполните щелчок по только что созданному проекту GUI_Test правой кнопкой мыши и выберите **New** → **Other** (можно использовать сочетание клавиш <Ctrl+N>).

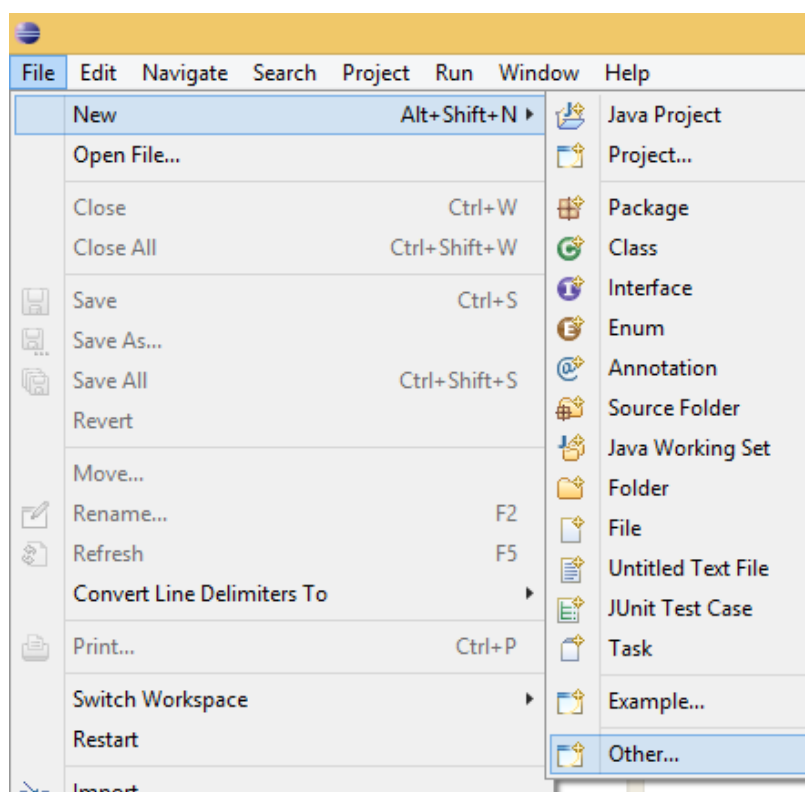


Рис. 9. Выбор программных компонентов

В открывшемся окне следует выбрать **Window Builder** → **Swing Designer** → **Application Window** и нажать **Next**.

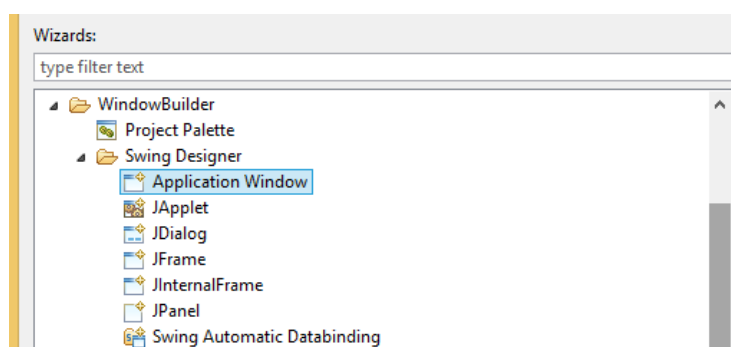


Рис. 10. Создание оконного приложения

4. В появившемся окне нужно ввести имя создаваемого приложения, например, «MyFirst_GUI» и нажать кнопку **Finish**.

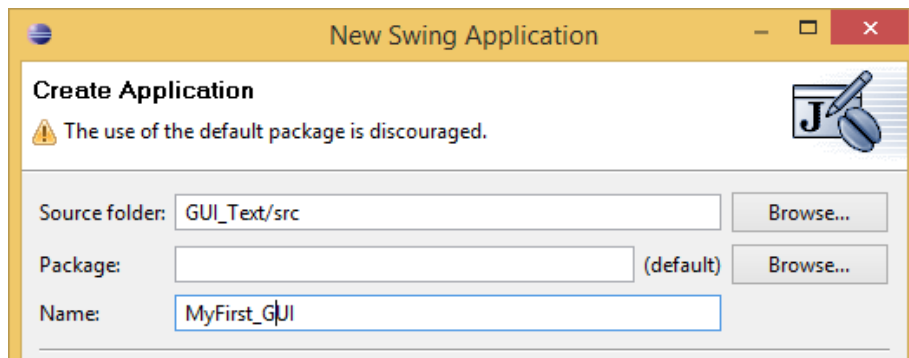


Рис. 11. Именованное приложение

Если все сделано правильно, то открывшееся окно будет выглядеть так:

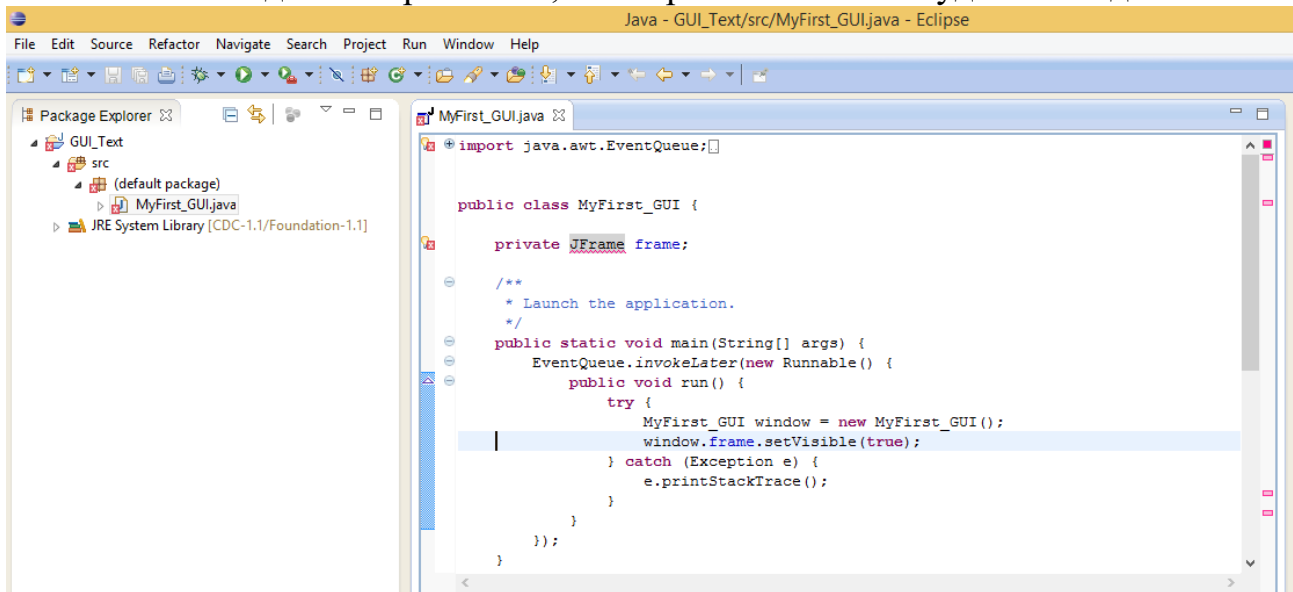


Рис. 12. Внешний вид Eclipse после создания GUI приложения

Нажмите кнопку **Run**, чтобы запустить приложение на выполнение и увидеть каркас приложения.

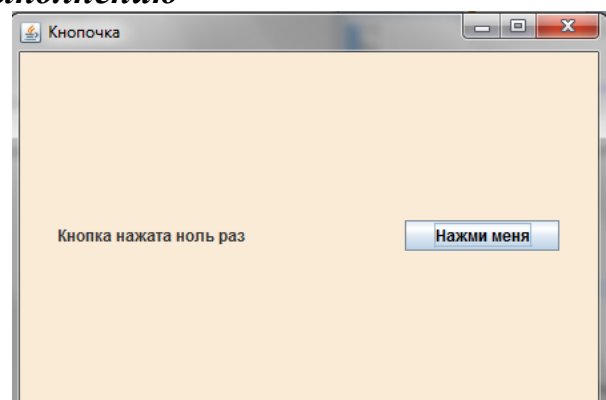
3.3 Задание и рекомендации по его выполнению

Разработать приложение с одной кнопкой. Каждое нажатие на кнопку будет увеличивать счетчик нажатий на 1 и его значение должно выводиться в текстовой форме в этом же окне.

Реализуйте выполнение дополнительного условия, например, каждый четный раз кнопка будет закрашиваться красным цветом, а нечетный оставаться серой.

Приложение должно содержать контейнер – панель, где размещаются компоненты – кнопка (JButton) и текстовая метка (JLabel).

JLabel – самый простой компонент, поскольку не предполагает обработку



действий пользователя, а лишь отражает текст и/или изображение. Конструктор JLabel (String str, Icon id_icon, int align) создает метку, содержащую строку str и/или графическое изображение id_icon. Для изменения содержимого следует вызвать метод setText (String). Метод setVisible(true) делает ее невидимой, подробнее см [2].

Кнопка может генерировать событие в ответ на щелчок мыши, поэтому следует создать обработчик этого события и зарегистрировать его у кнопки с помощью метода addActionListener(), которому в качестве параметра передать сам обработчик. Внутри выбранного метода обработчика необходимо реализовать счетчик нажатий (например, переменная counter, значение которой увеличивается на 1, при каждом вызове метода обработки нажатия мыши) и изменения содержимого текстовой метки.

3.4 Этапы выполнения работы с использованием Дизайнера форм

1. Для включения Дизайнера форм в нижней части окна редактора кода выберите вкладку **Design**.

После недолгой загрузки на экране появится редактор, состоящий из набора окон, включающего в себя область визуального редактирования, палитру компонентов **Palette**, представление **Structure**, отображающее иерархию используемых компонентов и представление **Properties**, отображающее свойства выбранного компонента (рис. 13).

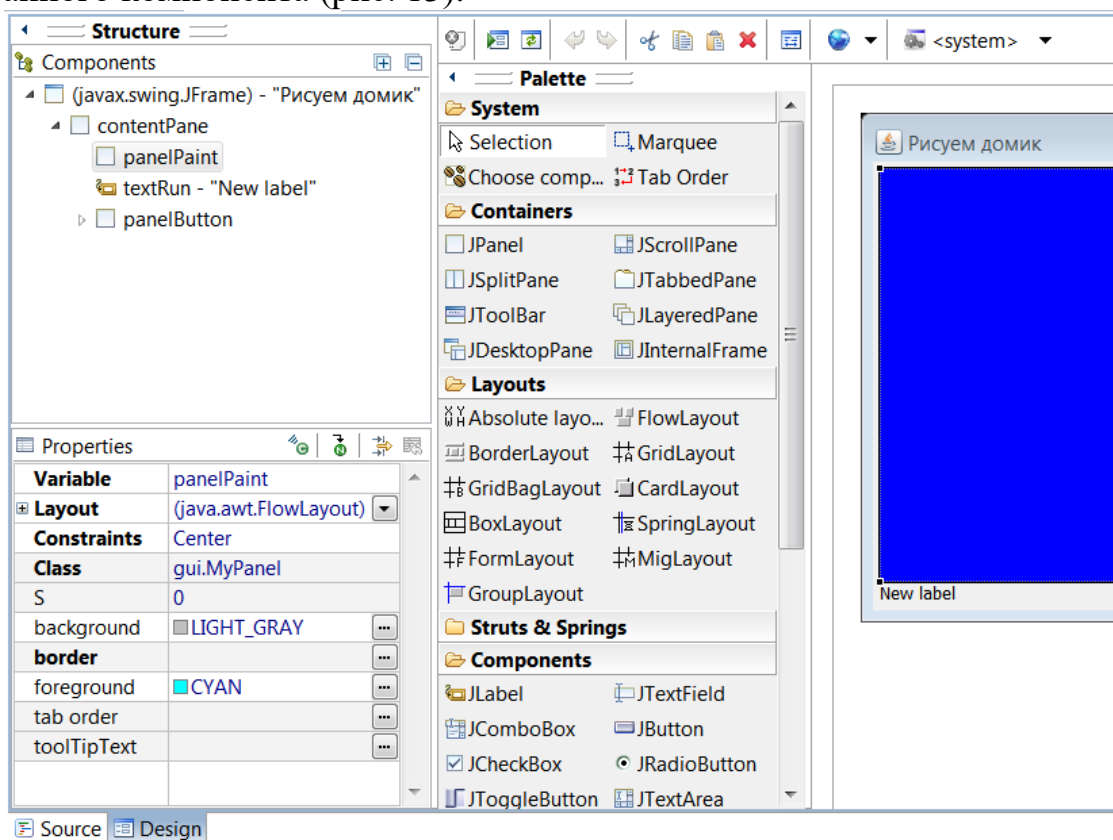
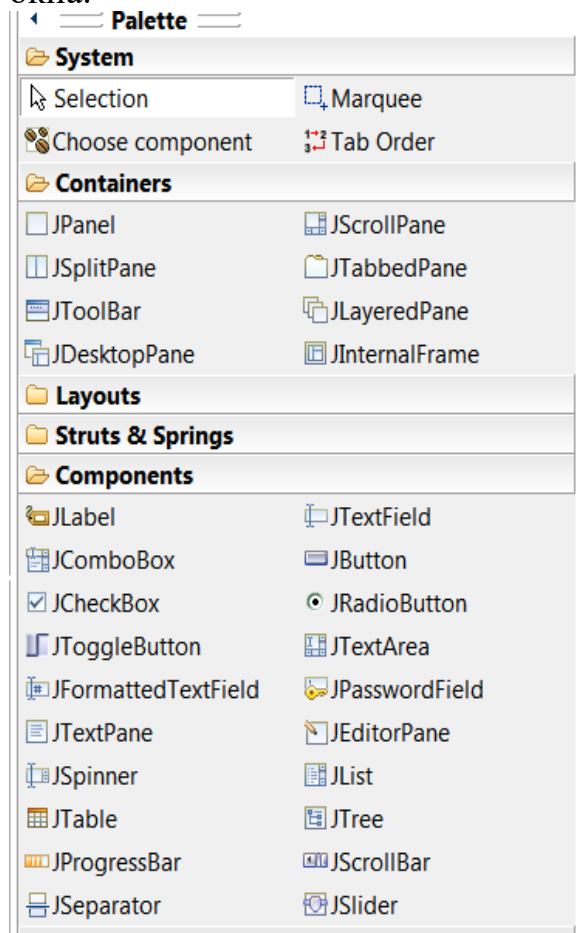


Рис. 13 – Визуальный графический редактор GUI-интерфейса конструктора приложений WindowsBuilder

Область визуального редактирования представляет собой холст дизайнера, в котором будет проектироваться GUI-интерфейс.

Окно Structure описывает содержимое области редактирования в виде иерархии использованных компонентов. При проектировании вносить элементы управления можно как на холст, так и в контейнеры, отображаемые в этом окне.



Палитра компонентов **Palette** содержит следующие разделы:

System – инструменты выбора элементов: Selection (выбор с помощью курсора), Choose Component (выбор элемента с помощью мастера Open type), Marquee (выбор группы элементов), TabOrder (определение порядка выбора элементов пользователем с помощью кнопки Tab).

Containers – контейнеры или элементы управления верхнего уровня (в них можно помещать другие элементы управления).

Layouts – менеджеры компоновки, содержат различные схемы взаимного размещения компонентов в окне.

Struts&Springs – инструменты выбора визуальных эффектов отображения элементов управления.

Components – инструменты выбора элементов управления.

Представление **Properties** содержит инструменты настройки свойств выбранного компонента. В зависимости от выбранного на холсте или каталоге Structure элемента управления набор свойств динамически меняется.

2. Для начала заполнения формы компонентами следует установить схему компоновки. Для первого приложения следует выбрать **Absolute Layout** в **Palette**→**Layouts**, что позволит размещать элементы GUI в произвольных местах нашего окна.

3. Первой разместите на холсте Дизайнера кнопку, нажатия которой будут считаться в приложении. Кнопка находится в **Palette**→**Components** и называется **JButton**. Следует выбрать JButton в Палитре компонентов Palette, затем курсор переместить вправо на внешний вид редактируемого окна на холсте Дизайнера. Чтобы зафиксировать кнопку на форме, нужно нажать и отпустить левую кнопку мыши в нужной позиции на редактируемом окне.

После добавления элемента в окно он появится в списке Components (на рис. 14, отмечен добавленный элемент button).

Сразу после добавления кнопки можно задать текст, который в ней будет отображаться. Все дальнейшие настройки свойств следует осуществлять в соответствующих полях представления Properties.

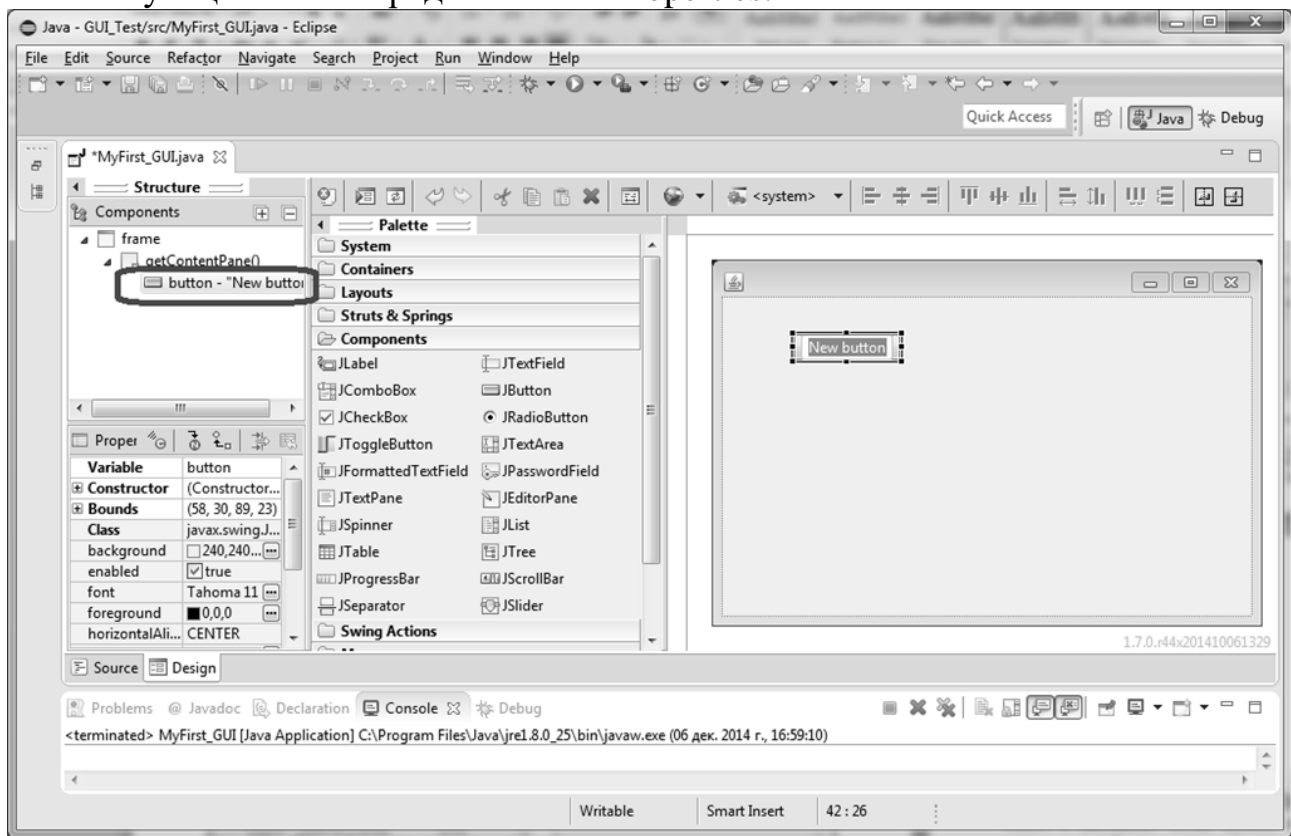


Рис.14. Размещение кнопки на форме

4. Следующей следует разместить текстовую метку JLabel для вывода сообщения о нажатии. Нужно выбрать JLabel в Палитре (**Palette**→**Components**), затем курсор переместить вправо на внешний вид редактируемого окна и выбрать место для добавления метки, кликнуть левой кнопкой мыши – метка фиксируется. Потом надо ввести текст метки «Кнопка нажата 0 раз», и изменить размер метки так, чтобы текст помещался на ней.

5. Нажмите правой кнопкой мыши по разработанному окну и в появившемся контекстном меню выберите **Test**→**Preview** для тестового предпросмотра получившегося окна.

6. Текст сообщения в метке должен меняться при каждом нажатии кнопки. Чтобы это реализовать, необходимо добавить обработчик нажатия кнопки. Для этого следует дважды кликнуть левой кнопкой мыши по кнопке. После этого в код будет добавлен пустой обработчик, и Eclipse автоматически перейдет в режим редактирования кода. При этом курсор будет поставлен в начало заголовка метода-обработчика. К кнопке button автоматически добавляется «слушатель» действий – это анонимный класс создаваемый как `new ActionListener() { }` с единственным методом для обработки события – нажатия мыши.

Далее следует объявить в основном классе приложения переменную счетчик и написать код обработчика:

```

public void actionPerformed(ActionEvent e) {
    // увеличиваем счетчик нажатий
    counter++;
    // формируем строку для вывода в метку
    String str = "Кнопку нажали раз: " + counter;
    // выводим новый текст в метку
    label.setText(str);
}
});

```

7. Запустите и протестируйте работу приложения.

4. СОЗДАНИЕ ПРИЛОЖЕНИЯ С ПАНЕЛЬЮ ДЛЯ РИСОВАНИЯ

Цель работы: Получение практических навыков работы с элементами управления GUI.

4.1 Теоретическая справка

Рисование и отображение графики возможно на контекст отображения. Контекст отображения, контейнер, компонент, обработчик, как и любой другой объект должен находиться в зоне видимости для обращения к нему.

Глобальные переменные	<code>public class Home extends JFrame{</code>
Параметры метода	<code>public HousePanel panel_4;</code>
Локальные переменные (метода)	<code>void initialize(int a) {</code>
	<code>JPanel panel = new JPanel();</code>
	<code>panel.setBackground(new Color(255, 228, 196));</code>
	<code>contentPane.add(panel, BorderLayout.NORTH);</code>
	<code>JButton button = new JButton("крѣша");</code>
	<code>button.addMouseListener(new MouseAdapter() {</code>
	<code>public void mouseClicked(MouseEvent arg0)</code>
	<code>{</code>
Параметры обработчика событий	<code>int select;</code>
	<code>panel_4.paint_knob();</code>
	<code>}</code>
	<code>}</code>
	<code>}</code>
	<code>}</code>

Рис. 15. Области видимости переменных

Получить доступ к графическому контексту GUI-компонента можно тремя способами.

1) Вызвать его с помощью метода `getGraphics`:

```
Graphics gpDraw;
```

```
gpDraw = panelDraw.getGraphics ();
```

2) Передать его автоматически через метод `paint`, которому контекст отображения, связанный с элементом его вызвавшим, передается при вызове:

```
void paint (Graphics g) {
```

```

g.setColor(Color.white);
g.fillPolygon(p);
g.setColor(Color.BLACK);
g.drawPolygon(p);
}

```

3) Использовать текущий контекст в зоне видимости компонента:

```

Dimension dimApp = this.size();
this.drawRect(1, 1, dimApp.width-1, dimApp.height-1);

```

Получив контекст отображения, можно рисовать, используя любые методы класса Graphics.

Любой элемент управления используется также как и любой другой программный объект, но обязательно требуется его помещение в контейнер верхнего уровня:

Объявление (panel=null)	MyPanel panel;
Создание	panel=MyPanel («Домик»);
Доступ к переменным	panel.select=1;
Вызов метода	panel.SetColor(Color.RED);
Добавление в контейнер	contentPane.add(panel);

Вызвать отрисовку GUI-компонента можно с помощью метода repaint не передавая ссылку на Graphics и он сам вызовет метод update(), который очищает компонент, заливает его фоновым цветом, устанавливает текущий цвет компонента и вызывает метод paint с необходимым контекстом отображения. Но метод repaint() заставляет систему лишь запланировать перерисовку компонента (не выполнить сразу, а поставить в очередь на обработку). Обычно подразумевается, что обновление должно произойти как можно скорее, но можно указать временной промежуток, в рамках которого его следует сделать. Такой подход применяют для управления частотой смены кадров.

Если метод paint() был переопределен, то его вызов необходимо самостоятельно делегировать в родительский класс: **super.paint(g);**

Это необходимо потому, что метод paint делегирует отрисовку трем защищенным методам. paintComponent выполняет отрисовку непосредственно на контексте отображения самого компонента, paintBorder рисует его рамку, paintChildren – дочерние элементы.

4.2. Задание и указание по его выполнению

Создайте приложение, позволяющее пользователю управлять рисованием в окне. Используйте созданный в предшествующей работе рисунок из семи графических примитивов, добавив к нему три кнопки – «Стереть», «Нарисовать», «Раскрасить».

Чтобы нарисовать изображение в окне приложения, нужно поместить метод paint() в один из визуальных компонентов. В Java для этого потребуется создать свой собственный компонент на основе одного из стандартных, например JPanel. Для этого в Eclipse создайте новый класс (обычный Java класс, не

Application Window), например MyPanel (рис. 16).

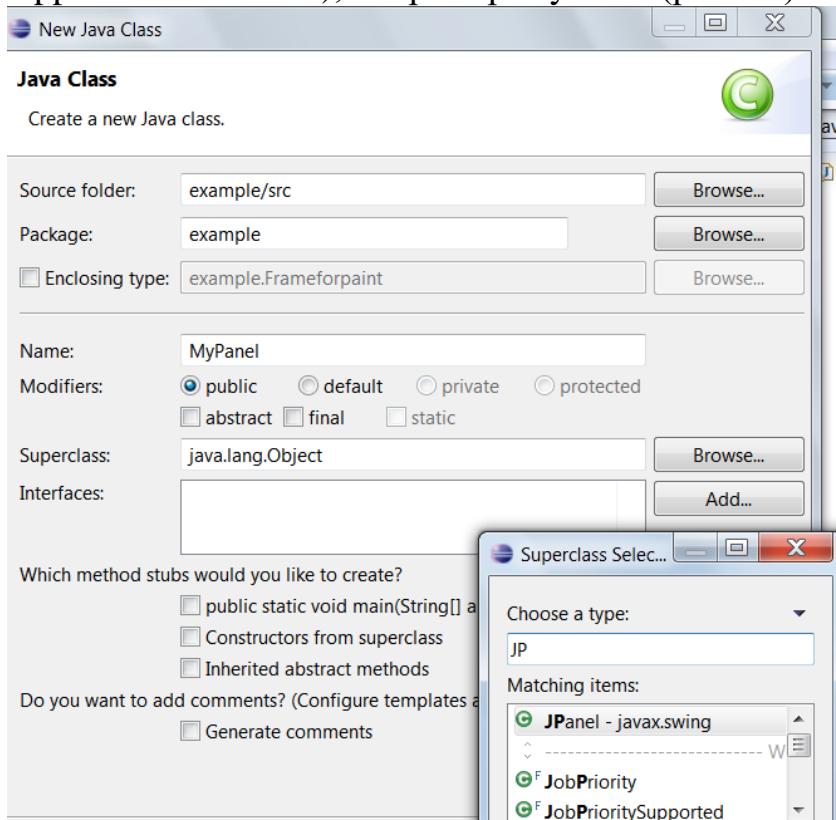


Рис. 16. – Создание Java класса

4.3 Этапы выполнения работы

1. Создайте новое окно приложения. Его можно создать и в старом проекте – GUI_Test. Нажатием правой кнопкой мыши по проекту выберите New => Other => Application Window. В открывшемся диалоге задайте имя класса окна, например Home.

2. В окно приложения Home добавьте BorderLayout и контейнер JPanel в центр (Center).

Чтобы панель была более заметна на окне приложения, задайте ей стиль границы BevelBorder, изменив свойство border в представлении Properties.

3. Перейдите в режим редактирования кода и найдите код метода initialize(), создающий эту панель:

```
JPanel panel = new JPanel();
```

Чтобы заменить стандартную панель JPanel на созданную ранее MyPanel, измените класс на MyPanel:

```
JPanel panel = new MyPanel();
```

В графе Superclass нажмите на кнопку «Browse...».

В строке Choose a type пишем JPanel и выбираем JPanel – javax.swing.

Нажмите OK и Finish. Галочку возле «public static void main» не ставить, чтобы оставить абстрактный метод, установленный по умолчанию.

В результате создается класс MyPanel и его код готов для редактирования.

К созданному классу следует добавить метод paint() для прорисовки желаемого изображения на панели.

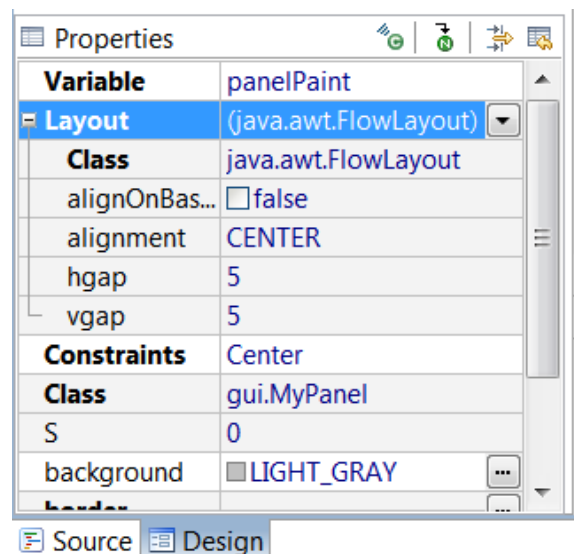


Рис. 17. Свойства разработанной панели

Для обращения к ней из различных точек программы рекомендуется объявить ее вне функции initialize() в самом классе окна:

```
JMyPanel panelPaint;
```

И в методе initialize() соответствующее объявление убрать, оставив лишь создание объекта:

```
panelPaint= new MyPanel ();
```

4. Запустите программу на выполнение и убедитесь, что она работает.

5. Вернитесь в Дизайнер и добавьте еще одну панель справа или слева от центра, и поместите на нее 3 кнопки «Стереть», «Нарисовать», «Раскрасить».

6. Напишите три одинаковых обработчика с разной функцией установки режима отрисовки панели.

```
public void actionPerformed(ActionEvent e) {  
    panelPaint.Clear(); // для «Стереть»  
/*Border() для «Нарисовать», Full() для «Раскрасить»*/  
    panelPaint.repaint(); }
```

7. В разработанном ранее классе реализуйте эти методы для задания переменной select, которая будет определять режим отображения в панели.

8. Добавьте отрисовку этих режимов в paint() и протестируйте работу программы.

9. Протестируйте отрисовку отдельных компонентов установкой переменной select в обработчиках кнопок перед вызовом repaint(). Метод paint можно построить следующим образом:

```
super.paintComponents(g);  
switch (select){  
    case 1: paintGrass(g); break;  
    case 2: paintWall(g); break;  
  
    case 7: paintWindow(g); break;  
    }
```

5. РАЗМЕЩЕНИЕ ЭЛЕМЕНТОВ УПРАВЛЕНИЯ В ОКНЕ

Цель работы: Получение практических навыков работы с менеджерами размещения компонентов.

5.1 Теоретическая справка

Напрямую в окне элементы управления не размещаются. Для этого служит панель содержимого, занимающая все пространство окна. Обратиться к этой панели можно методом **getContentPane()** класса **JFrame**. С помощью метода **add(Component component)** можно добавить на нее любой элемент управления.

В классе Container определено около двух десятков методов для управления набором компонентов, содержащихся в контейнере. Они похожи на методы класса-коллекции и являются коллекцией особого рода — визуальной. Кроме хранения элементов контейнер занимается их пространственным расположением и прорисовкой. У каждой панели есть, так называемый, менеджер размеще-

ния, который определяет стратегию взаимного расположения элементов, добавляемых на панель. Его можно изменить методом `setLayout(LayoutManager manager)`.

Менеджер последовательного размещения **FlowLayout** размещает добавляемые на панель компоненты строго по очереди, строка за строкой, в зависимости от размеров панели. Как только очередной элемент не помещается в текущей строке, он переносится на следующую.

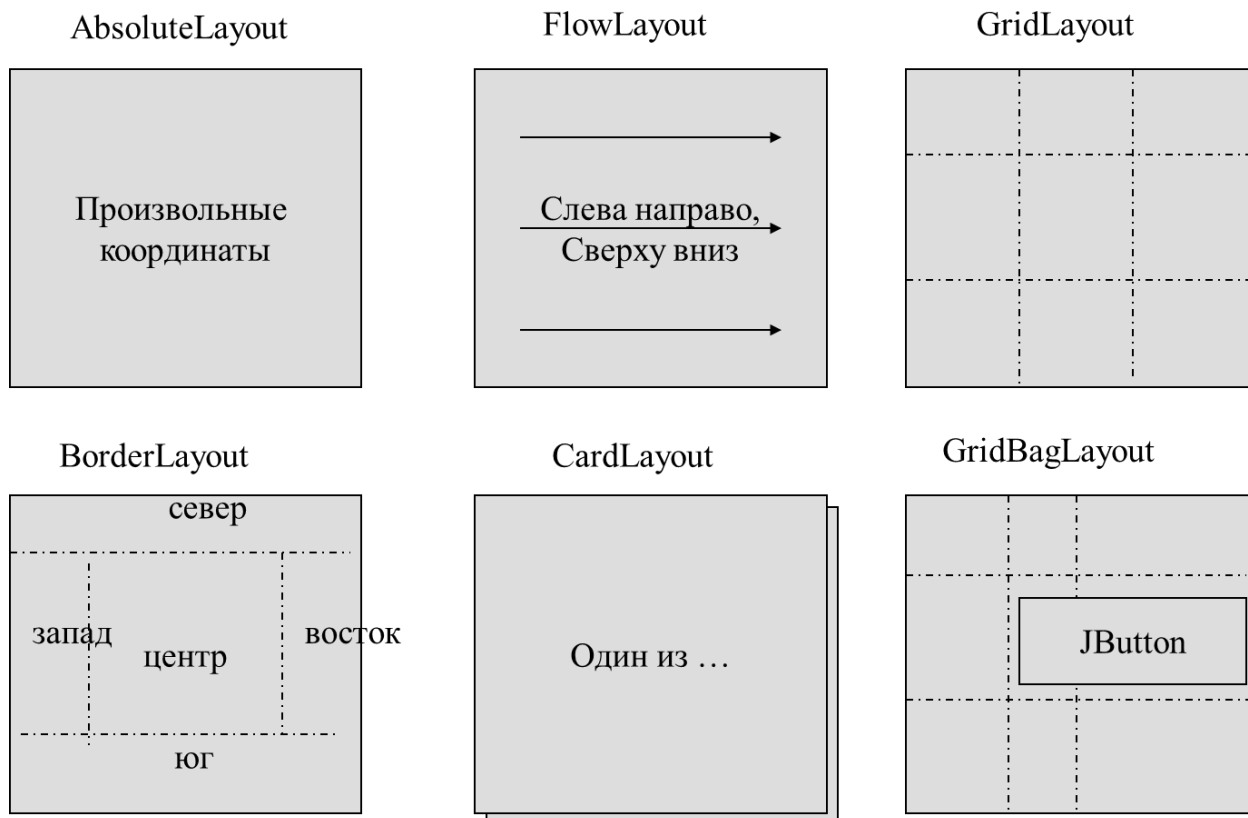


Рис. 18 – Наиболее распространенные менеджеры размещения

Менеджер размещения **BorderLayout** разделяет панель на пять областей: центральную, верхнюю, нижнюю, правую и левую (константы, определенные в классе `BorderLayout`: `NORTH`, `SOUTH`, `EAST`, `WEST` и `CENTER`). В каждую из этих областей можно добавить ровно по одному компоненту, причем компонент будет занимать всю отведенную для него область. Компоненты, добавленные в верхнюю и нижнюю области, будут растянуты по ширине, добавленные в правую и левую — по высоте, а компонент, добавленный в центр, будет растянут так, чтобы полностью заполнить оставшееся пространство панели.

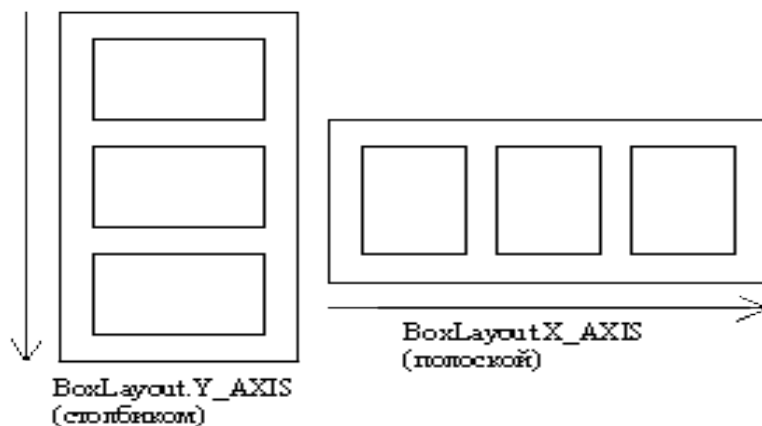
При добавлении элемента на панель с менеджером размещения `BorderLayout`, необходимо дополнительно указывать в методе `add()`, какая из областей имеется в виду.

Менеджер табличного размещения **GridLayout** разбивает панель на ячейки одинаковой ширины и высоты (таким образом окно становится похожим на таблицу). Каждый элемент, добавляемый на панель с таким расположением, целиком занимает одну ячейку. Ячейки заполняются элементами по очереди,

начиная с левой верхней. Этот менеджер, в отличие от рассмотренных ранее, создается конструктором с параметрами (четыре целых числа). Необходимо указать количество столбцов, строк и расстояние между ячейками по горизонтали и по вертикали.

Менеджер блочного размещения **BoxLayout** размещает элементы на панели в строку или в столбец.

Элементы, добавленные на панель с блочным размещением, выстраиваются один за другим в строку или в столбец (определяется заданием **X_AXIS** и **Y_AXIS**). Расстояние между элементами по умолчанию нулевое.



Однако, вместо компонента можно добавить невидимую «распорку» (**Strut**), единственная задача которой — раздвигать соседние элементы, обеспечивая между ними заданное расстояние. Кроме того, на такую панель можно добавить еще один специальный элемент — своеобразную «пружину» (**Glue**). Если размер панели будет больше, чем необходимо для оптимального размещения всех элементов, те из них, которые способны растягиваться, будут стараться заполнить дополнительное пространство собой. Если же разместить среди элементов одну или несколько «пружин» (**Glue**), дополнительное свободное пространство будет распределяться и в эти промежутки между элементами.

Принцип работы менеджера **GridBagLayout** прост: контейнер разбивается на сетку, и над каждой ячейкой сетки организуется свой контроль. Компоненты могут занимать любое количество ячеек сетки, ячейки могут оставаться пустыми и т.п. **GridBagLayout** располагает компоненты на форме с помощью следующих полей.

поле	описание
gridx, gridy	номер столбца и номер строки для ячейки, в которую будет помещен компонент (верхняя левая ячейка =0,0)
gridwidth, gridheight	определяют количество ячеек, занимаемых добавляемым компонентом (две смежные ячейки в одной строке gridwidth=2 и gridheight =1)
fill	распределение компонента по ячейкам (NONE – компонент не изменяет своих размеров, BOTH – изменяется высота и ширина, так чтобы компонент занимал все отведенное для него пространство, HORIZONTAL – компонент растягивается по горизонтали, VERTICAL – по вертикали)
anchor	выравнивание компонента внутри отведенного для него пространства

поле	описание
weightx, weighty	изменение размеров компонента (чтобы размеры компонента изменялись по горизонтали или вертикали значения должны лежать от 0.0 до 1.0)
ipadx, ipady	задание количества пикселей по горизонтали и вертикали на которые необходимо увеличить
insets	отступы компонента от краев выделенной ему области (сверху, слева, снизу, справа)

Обратите внимание, менеджер расположения обязан расположить добавляемые в контейнер компоненты в некотором порядке, зависящем от реализованного в нем алгоритма, и придать им некоторый размер, при этом он обычно учитывает определенные свойства компонентов, что и определяет конечный результат отображения.

5.2 Практические задания

1. Создайте кнопки «Да, нравится», «Скорее да, чем нет», «Не знаю», «Скорее нет, чем да», «Нет, не нравится», и текстовую метку «Нравится ли тебе учиться?» и разместите их с помощью менеджера последовательного размещения FlowLayout. Поварьируйте размерами созданного окна. Измените схему размещения: по левому краю, по правому краю, по центру.

2. Создайте 5 кнопок с надписями «Верх», «Низ», «Правая сторона», «Левая сторона», «Центр» и разместите их с помощью менеджера граничного размещения BorderLayout. Эффект будет хорошо наблюдаться, если изменять размеры окна.

3. Создайте 15 одинаковых кнопок с цифрами от 1 до 15 и разместите их с помощью менеджера табличного размещения GridLayout как для игры в «Пятнашки».

4. Создайте 5 кнопок "плохо", "посредственно", "удовлетворительно", "хорошо", "отлично" и расположите их с помощью BoxLayout с параметром Y_AXIS и установите одинаковый размер распорок (strut), а потом X_AXIS, включите "пружинки" (glue) и увеличьте размер окна.

5. С помощью GridBagLayout создайте следующую форму. Взаимное расположение компонентов при изменении размеров окна не должно меняться.

Вход в систему	
Имя:	<input type="text"/>
Пароль:	<input type="text"/>
	<input type="button" value="ОК"/> <input type="button" value="Отмена"/>

6. Создайте полноценный интерфейс для раскрашивания картинki, созданной в ходе предшествующих работ.

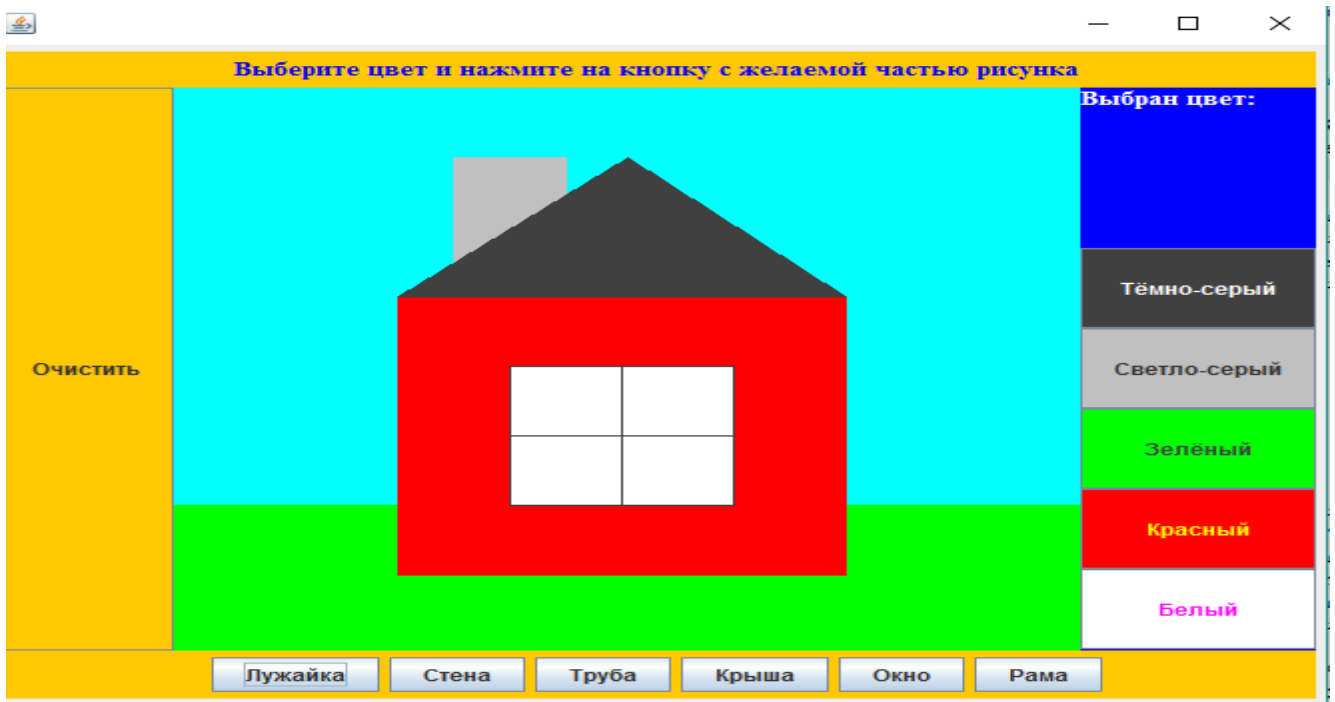


Рис. 19. Интерфейс GUI-приложения «Раскраска»

5.3 Рекомендации по выполнению заданий

При выполнении практических заданий учтите, что область визуального редактирования имеет контекстное меню, позволяющее:

Cut	Ctrl+X	вырезать
Copy	Ctrl+C	копировать
Paste	Ctrl+V	вставлять
Delete	Delete	удалять компоненты
Test/Preview...		осуществить предварительный просмотр приложения.
Refresh	F5	обновить область
Add event handler		добавить слушателя событий в компонент.
Set layout		устанавливать компоновку дочерних компонентов.
Select		создать метод, возвращающий экземпляр компонента.
Expose component...		превратить в другой компонент.
Morph		создать класс-фабрику для выбранного компонента
Factory		
Rename...		переименовать компонент

Рис. 20. Контекстное меню холста

Настраивать параметры менеджеров размещения можно также в окне

Свойств, изменяя и задавая соответствующие значения (рис. 21).

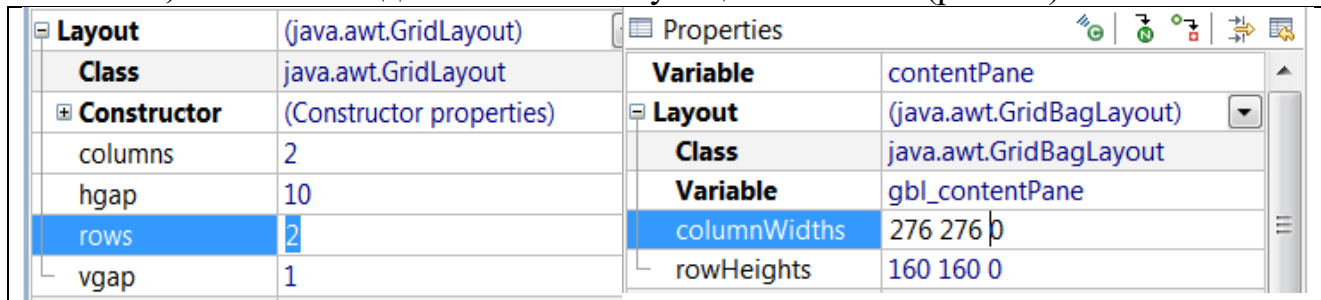



Рис. 21. Развернутая вкладка менеджера размещений на панели свойств

Существует два способа добавить обработчик событий с использованием WindowBuilder. Первый способ с использованием панели свойств Properties. Сначала выберите компонент либо в дереве иерархии компонентов **Structure** или на холсте Design View. Щелкните кнопку **Show Events**  для получения списка в Properties. Расширьте событие: либо щелкните дважды, либо нажмите Enter на обработчике, который необходимо создать.

Второй способ добавления обработчика событий состоит в использовании контекстного меню (рис. 20) и выборе Add event handler.

Чтобы удалить существующий обработчик событий в WindowBuilder, выберите компонент на холсте или в окне **Structure**. В окне свойств переключитесь на вкладку **Show Events**, щелкните на метод события, который намерены удалить и нажмите **Delete**.

6. ОБРАБОТКА СОБЫТИЙ МЫШИ

Цель работы: Получение практических навыков обработки событий

6.1 Теоретическая справка

Обработка событий (event handling) – это механизм, позволяющий двум или более объектам системы обмениваться информацией об изменении состояния. В системах, управляемых событиями, один объект всегда выступает в качестве генератора событий, создавая объект – события, предназначенные для обозначения определенных изменений в состоянии этого объекта. Затем это событие передается одному или нескольким получателям, которые выполняют необходимые действия.

Модель делегирования событий Swing работает следующим образом (рис. 22). Событие возникает при воздействии на компонент (манипулирование мышью, ввод с клавиатуры, перемещение и изменение размеров окна). Источником события (Source) может быть любой объект, наследуемый от класса Component. При возникновении события автоматически создается объект соответствующего класса вида xxxxxxEvent, который не производит никаких действий, а только хранит сведения о событии.

Для того чтобы обработка события произошла, необходим обработчик. Обработчики событий — это интерфейсы, в которых содержатся определения методов, вызываемых источниками событий при возникновении определенных

обстоятельств: щелчок на кнопке, перемещение мыши и т.п. В приложении Б представлены интерфейсы широко используемых обработчиков событий. Поэтому, чтобы выполнить обработку события нужно реализовать соответствующий этому событию интерфейс.

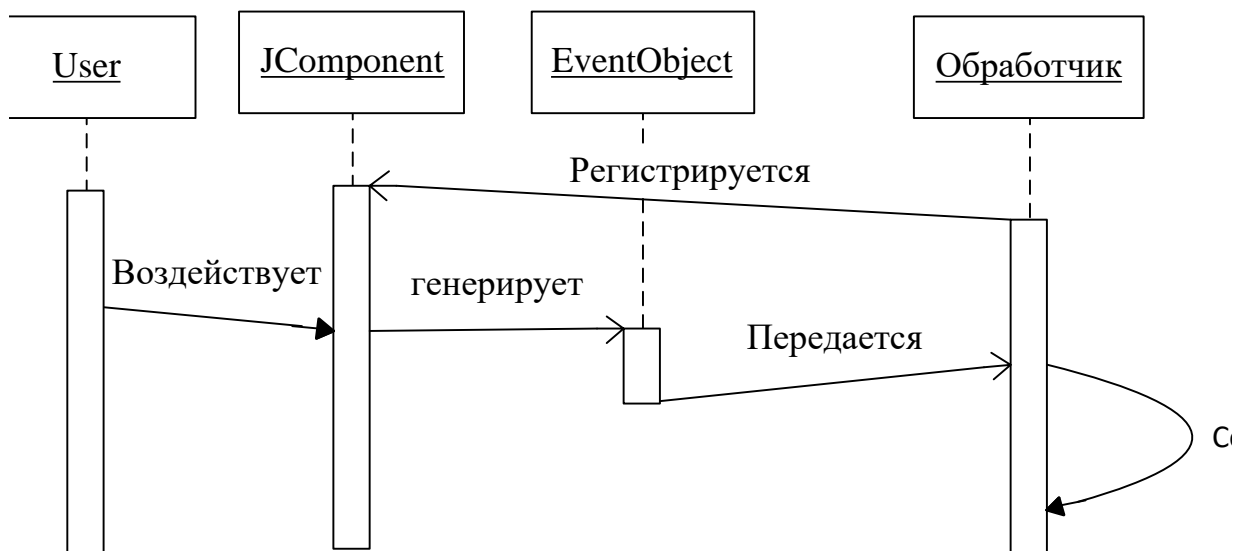


Рис. 22. Последовательность обработки событий

Ключевым в этом механизме является необходимость регистрация обработчика у компонента, что обеспечивает оповещение о произошедшем событии слушателя (Listener). Для управления слушателями служат методы:

- addXXXListener(XXXListener listener) – добавить слушателя;
- removeXXXListener(XXXListener listener) – убрать слушателя.

Для использования модели делегирования событий необходимо:

1. Создать слушателя, т. е. реализовать соответствующий интерфейс в слушателе, чтобы он мог принимать события данного типа.
2. Добавить слушателя к компоненту, т.е. зарегистрировать слушателя как получателя уведомлений о событиях.
3. Реагировать на события.

Основной путь создания слушателей – это реализация соответствующего интерфейса в классе:

```
class a implements MouseListener {
```

Но большинство интерфейсов (см. прил. В) описывают несколько методов, чтобы не реализовывать их все, используют классы-адаптеры. Адаптер – абстрактный класс, содержащий пустые реализации методов. От такого класса можно наследоваться и переопределить только те методы, которые нужны для приложения.

В некоторых случаях создание слушателя и его регистрацию в компоненте-источнике можно сделать быстро, например, с помощью вложенного класса:

```
cp.add(jb);  
cp.add(jtf);  
jb.addActionListener(new TextMove());
```

```

    jf.setVisible(true);
}
class TextMove implements ActionListener
{public void actionPerformed(ActionEvent obj)
  { jtf.setText("Button pressed"); }
}

```

Или с помощью анонимного обработчика:

```

jb.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent obj)
    { jtf.setText("Button pressed"); }
} );

```

6.2 Задания и рекомендации по их выполнению

1. Реализуйте и исследуйте программу, которая сообщает о действии, выполняемом мышкой на компоненте.

В приложении с кнопкой (см. работа 3) панели, метке и другим имеющимся элементам управления задать имя. Это можно сделать в окне свойств каждого компонента в Дизайнере или прямо в коде:

```
myButton.setName("mybutton");
```

Также добавьте внизу текстовую метку eventLabel для вывода сообщений о действиях мыши

Реализуйте интерфейс MouseListener и изучите, какие виды событий могут происходить.

```

class MyClick implements MouseListener{
  JComponent comp;
  public void mouseClicked(MouseEvent e) {
    comp = (JComponent) e.getSource();
    String text = "<html><b>" + comp.getName()
      + " mouseReleased() <br>" + comp.getName()
      + " mouseClicked() </b><html>";
    titul.myLabel.setText(text);
    System.out.println("щелчок");
  }
  //появление курсора мыши в компоненте
  public void mouseEntered(MouseEvent e) {
    comp = (JComponent) e.getSource();
    titul.myLabel.setText(comp.getName() + "
mouseEntered() ");
  }
  //выход курсора мыши из компонента
  public void mouseExited(MouseEvent e) {
    comp = (JComponent) e.getSource();
    titul.myLabel.setText(comp.getName() + " mouseExited() ");
  }
  //нажатие кнопки мыши
  public void mousePressed(MouseEvent e) {
    comp = (JComponent) e.getSource();
    titul.myLabel.setText(comp.getName() + "
mouse-Pressed() ");
  }
}

```

```

    }
    //отпускание кнопки мыши
    public void mouseReleased(MouseEvent e) {
        comp = (JComponent) e.getSource();
        titul.myLabel.setText(comp.getName() + " mouseRe-
leased()");
    }
}

```

Обратите внимание, что слушатели должны быть добавлены к объекту, для работы, например к панели:

```
mypanel.addMouseListener(new MyClick());;
```

Сначала подключите обработчик к кнопке, потом текстовой метке, затем панели-контейнеру. Изучите особенности работы программы.

2. Доработайте код задачи 1 так, чтобы сообщалось на каком компоненте окна в данный момент находится мышь.

Метод `mouseenter` получает управление, когда курсор мыши в процессе перемещения по экрану попадает в область элемента управления:

```
public boolean mouseEnter(Event evt, int x, int y);
```

Этот метод можно использовать для активизации элемента управления, на который указывает курсор мыши.

Метод `mouseExit` вызывается при покидании курсором элемента управления:

```
public boolean mouseExit(Event evt, int x, int y);
```

Если пользователь убрал курсор из элемента управления, активизированного методом `mouseenter`, то метод `mouseExit` может переключить этот элемент управления в пассивное состояние.

Если при обработке событий важна информация о месте, компоненте и других особенностях произошедшего события, то следует использовать получаемый при возникновении события объект класса `Event` (таблица).

Обычно экземпляр класса `Event` создается исполняющей системой Java, для доступа к свойствам, следует использовать методы:

`int getClickCount()` – возвращает число щелчков мышью, связанных с этим событием;

`Point getLocationOnScreen()` – возвращает абсолютную позицию `x`, `y` на экране, где произошло события;

`Point getPoint()` – возвращает `x`, `y` позиции события относительно исходного компонента;

`int getX()`, `int getY()` – возвращают горизонталь `x` и вертикаль `y` позиции события относительно исходного компонента;

`String paramString()` – возвращает строку параметра, идентифицирующую это событие;

`public Component getComponent()` – возвращает инициатора события;

`public String toString()` – возвращает строковое представление этого объекта.

Таблица – Атрибуты класса Event

Тип	Название	Описание
Object	target	Ссылка на компонент, который первоначально получил сообщение
long	when	Момент времени в который произошло событие
int	id	Тип события*.
int	x, y	Координата x, y точки в которой произошло действие, относительно координат компоненты которая обрабатывает событие.
int	key	Для событий клавиатуры — код нажатой клавиши
int	modifiers	Комбинация значений SHIFT_MASK, CTRL_MASK, META_MASK и ALT_MASK для клавиш shift, control, meta и alt
int	clickCount	Количество последовательных нажатий мыши (используется в MOUSE_DOWN).
Object	arg	Аргумент зависящий от события (для объектов Button представляет собой объект типа String, который содержит надпись на этой кнопке)
Event	evt	Следующее событие в связанном списке событий.

3. Напишите программу, которая позволяет рисовать линии, следуя движениям мыши, т. е. пользователь нажимает клавишу мыши, перемещает курсор и затем отпускает клавишу мыши. При движении мыши рисуется тонкая серая линия, которая заменяется линией с заранее выбранным цветом после отпущения клавиши мыши.

Для этого реализуйте для панели, на которой происходит рисование интерфейс MouseMotionListener и метод paint() с прорисовкой полигона, запомненного во время движения мыши.

Переопределив метод mouseDown, можно отслеживать нажатия клавиши мыши (прототип метода `public boolean mouseDown(Event evt, int x, int y);`). Через параметр evt методу передается ссылка на объект Event, с помощью которой метод может получить полную информацию о событии. Анализируя содержимое параметров x и y, приложение может определить координаты курсора на момент возникновения события.

Следует обратить внимание, что некоторые события являются составными, так для отслеживания двойного щелчка мыши не предусмотрено никакого отдельного метода. Однако анализируя содержимое поля clickCount переменной evt, можно определить кратность щелчка мыши:

```

if(evt.clickCount > 1) // Двойной щелчок
    showStatus("Mouse Double Click");
else // Одинарный щелчок
    showStatus("Mouse Down");

```

При отпущении клавиши мыши управление получает метод mouseUp:

```

public boolean mouseUp(Event evt, int x, int y);

```


Когда пользователь перемещает курсор мыши над элементом управления, в процессе перемещения происходит вызов метода `mouseMove`:

```
public boolean mouseMove(Event evt, int x, int y);
```

Операция Drag and Drop выполняется следующим образом: пользователь нажимает клавишу мыши и, не отпуская ее, начинает перемещать курсор мыши. При этом происходит вызов метода `mouseDrag`:

```
public boolean mouseDrag(Event evt, int x, int y);
```

в котором через переменные `x` и `y` передаются текущие координаты курсора мыши.

6.3 Примеры реализации обработчиков событий

Пример 1. Программа, позволяющая рисовать на панели линии, выбранным с помощью стандартного диалога цветом.

Интерфейс состоит из панели для рисования (`JPanel panel`), кнопки выбора цвета `button` (`JButton("Выбор цвета")`) и текстовой метки `eventLabel` (`JLabel("Действия мыши")`).

В классе приложения задействованы переменные:

`Color col` – для хранения выбранного цвета рисования;

`int x, y` – для сохранения предыдущей позиции мыши.

Реализация интерфейса `MouseListener`:

```
public void mouseDragged(MouseEvent arg0) {  
    eventLabel.setText("Moved");  
    panel.getGraphics();  
    g.setColor(col);  
    g.drawLine(x, y, arg0.getX(), arg0.getY()  
    }  
    public void mouseMoved(MouseEvent arg0) {  
        eventLabel.setText("Dragged");  
        x = arg0.getX();  
        y = arg0.getY();  
    }
```

Обработчик движения мыши добавляется к панели:

```
panel.addMouseListener(this);
```

Анонимный класс-адаптер для обработки нажатия на кнопку с целью задания цвета рисования может быть реализован следующим образом:

```
public void mouseClicked(MouseEvent arg0) {  
    col = JColorChooser.showDialog(panel, "Цвет",  
col); //выбор цвета из палитры  
    }
```

Пример 2. Программа, перемещения мяча по экрану с помощью клавиш

Реализация интерфейса-слушателя клавиш:

```
public void keyPressed(KeyEvent e) {  
    switch (e.getKeyCode()) {  
        case KeyEvent.VK_UP:  
            y -= 10; repaint();break;  
        case KeyEvent.VK_DOWN:
```

```

        y += 10; repaint();break;
    case KeyEvent.VK_LEFT:
        x -= 10; repaint();break;
    case KeyEvent.VK_RIGHT:
        x += 10; repaint();break;
    }
}

```

Присоединение слушателя к компоненту:

```
panel.addKeyListener(this);
```

Реализация функции перерисовки

```
public void paint(Graphics g){
    g.fillOval(x, y, 15, 15); }

```

7. РАСШИРЕНИЕ ФУНКЦИОНАЛА GUI-ПРИЛОЖЕНИЙ

Цель работы: Закрепление навыков работы с элементами управления библиотеки Swing

7.1. Теоретическая справка

Для расширения функциональных возможностей приложения можно использовать множество других компонентов Swing (прил. А).

Флажки-переключатели являются объектами класса JCheckBox. Самая общая форма конструктора:

```
JCheckBox (String str, Icon id_icon, boolean state)
```

Управление состоянием флажка из программы осуществляется с помощью методов:

```
boolean getSelected()
```

```
void setSelected(boolean state) .
```

Самая общая форма конструктора радиокнопок:

```
JRadioButton (String str, Icon id_icon, boolean state)
```

Радиокнопки должны быть объединены в группу, в пределах которой будет обеспечиваться уникальность выбора элемента. Для создания группы необходим экземпляр объекта специального класса ButtonGroup. Далее все созданные радиопереключатели добавляются в созданную группу (что не отменяет необходимости их добавления к панели содержания).

```
// объявление и создание 3-х радиокнопок
```

```
JRadioButton jrb1 = new JRadioButton("Red", false);
```

```
JRadioButton jrb2 = new JRadioButton("Green", false);
```

```
JRadioButton jrb3 = new JRadioButton("Yellow", true);
```

```
// объявление и создание объекта для группировки кнопок
```

```
ButtonGroup bg = new ButtonGroup();
```

```
// объединение кнопок в одну группу
```

```
bg.add(jrb1); bg.add(jrb2); bg.add(jrb3);
```

```
// добавление кнопок в контейнер JPanel cp = JPanel();
```

```
cp.add(jrb1); cp.add(jrb2); cp.add(jrb3);
```

Списком называется набор элементов, один или несколько из которых могут быть выбраны из создаваемого окна с прокруткой.

Для создания списка необходима инициализация объекта типа String. Конструкторы JList(Vector v[]) или JList(Object obj[]) могут принимать любые объекты, но список необходимо заполнять данными в момент создания.

Кроме того, объекты типа JList не поддерживают автоматическую прокрутку. Поэтому, для того, чтобы элементы в списке могли прокручиваться, список необходимо поместить в специальный объект JScrollPane, а уже этот объект добавить в контейнер.

```
String s[] = {"1", "2", "3", "4", "5", "6", "7", "8", "9", "10"};
JList jl = new JList(s);
JScrollPane p = new JScrollPane(jl);
cp.add(p);
f.setVisible(true);
```

Для определения выбранного элемента (элементов) используются методы:

Object getSelectedValue() или Object[] getSelectedValues()

int getSelectedIndex() или int[] getSelectedIndices.

Однострочная область ввода текста или текстовое поле JTextField дает возможность вводить строки, редактировать их с помощью клавиш-стрелок, Backspace, Delete, пользоваться буфером обмена.

Конструктор	создает текстовое поле
JTextFiled()	пустое
JTextFiled(int numChars)	шириной numChars символов
JTextFiled(String str)	заполненное строкой str
JTextFiled(String str, int numChars)	заполненное строкой str и ограниченное по ширине numChars символов

Основные методы класса JTextField:

String getText() – возвращает строку, содержащуюся в поле ввода;

void setText(String str) – устанавливает в поле ввода текст str;

String getSelectedText() – возвращает выделенную пользователем часть текста.

7.2 Задания

Для приложения «Раскраска» интерфейс которого был разработан в предшествующей работе реализуйте дополнительный функционал:

1. Реализуйте задание цвета с использованием 7 переключателей CheckBox.

2. Организуйте возможность выбора графического примитива для отображения из списка.

3. Предусмотрите возможность прорисовки элемента с места, на которое указывает курсор.

4. Создайте статусную строку, отображающие выбранные настройки (например, «Рисую окно» и т.п.)

5. Реализуйте возможность по щелчку мыши разместить на изображении

надпись (Щелчком визуализировать TextEdit, ввести надпись, скрыть компонент и отобразить текст в графическом контексте).

7.3 Рекомендации по выполнению задания

Поскольку для рисования графического примитива потребуется изменять цвет, размеры и координаты эффективно создать иерархию графических фигур. В основу иерархии следует поместить базовый класс, который отражает общие атрибуты и методы. От базового класса породить производные, которые будут отвечать за рисование различных примитивов. В идеале учесть, что из нескольких фигур может быть составлена новая, которая будет использоваться как отдельный примитив. В дальнейшем возможности использования созданных классов можно увеличить за счет сохранения и загрузки изображений, и введения дополнительных команд.

1. Создание абстрактного класса

Каждая фигура характеризуется:

- положением на экране;
- размером;
- цветом и толщиной линий;
- цветом и стилем заливки.

Действия, которые фигура должна выполнять:

- нарисовать себя на указанном контексте;
- сообщить свое имя, видимость, размер и другие атрибуты.

```
public class Share {  
    public boolean isd = true; // видимость  
    public aRect pos; // координаты отображения  
    public Color fullcol, brcol; // Цвет заполнения и рамки  
    public String name; // название элемента  
    public Share () {  
        //установить цвета и размер по умолчанию  
    }  
    public aRect getPos() { // получить размер  
        return pos;  
    }  
  
    public void setPos(aRect r) { // установить размер  
        pos = r;  
    }  
  
    public boolean isDraw() { // определить видимость  
        return isd;  
    }  
  
    public void setDraw(boolean b) { // установить отрисовку  
        isd = b;  
    }  
  
    public void draw(Graphics cv, aRect r) {
```

}

Отдельный класс `aRect` для пропорционального изменения размеров графических примитивов представлен в приложении А.

2. Для каждого из изображений следует создать собственный класс:

```
public class Windw extends Share {  
    public Windw () { // конструктор  
        isd = true; // видимость  
        public aRect pos; // координаты отображения  
        fullcol=Color.White;  
        brcol=Color.Black; /  
        name="Окно"; //  
    }  
}
```

И реализовать в нем метод прорисовки, в котором выполнять действия только, если объект видим:

```
public void draw(Graphics g, aRect r) {  
    if(isd) {  
        g.SetColor(fullcol);  
        g.drawfullOval(r.X(), r.Y(), r.W(), r.H());  
        g.SetColor(brcol);  
        g.drawOval(r.X(), r.Y(), r.W(), r.H());  
    }  
}
```

3. В обработчике соответствующего события для прорисовки необходимо вызвать метод `draw` для заранее созданного объекта соответствующего класса, например

```
Windw bi =new Windw();  
bi.Draw;
```

В идеале следует реализовать паттерн Компоновщик.

ТРЕБОВАНИЯ К РЕЗУЛЬТАТАМ ВЫПОЛНЕНИЯ РАБОТ И ОФОРМЛЕНИЮ КОДА

Результаты выполнения каждой работы оформляются в виде отчета. Отчет должен содержать:

- полное описание задания и ожидаемых результатов работы программы;
- диаграмму классов с пояснением назначения методов и атрибутов;
- листинг логической части программы с комментариями;
- скриншот тестового примера.

При оформлении кода программы на языке Java следует руководствоваться следующими принципами [3].

При выборе имени класса, поля, метода использовать целые слова, полностью исключить сокращения. По возможности опускать предлоги и очевидные связующие слова. Аббревиатуры использовать только в том случае, если они очевидны.

Имя класса всегда пишется с большой буквы: `Coin`, `Developer`. Если имя класса состоит из двух и более слов, то второе и следующие слова пишутся слитно с предыдущим и начинаются с большой буквы: `AncientCoin`,

FrontendDeveloper.

Имя метода всегда пишется с маленькой буквы: `perform()`, `execute()`. Если имя метода состоит из двух и более слов, то второе и следующие слова пишутся слитно с предыдущим и начинаются с большой буквы: `performTask()`, `executeBaseAction()`.

Имя поля класса, локальной переменной и параметра метода всегда пишется с маленькой буквы: `weight`, `price`. Если имя поля класса, локальной переменной и параметра метода состоит из двух и более слов, то второе и следующие слова пишутся слитно с предыдущим и начинаются с большой буквы: `priceTicket`, `typeProject`.

Константы и перечисления пишутся в верхнем регистре: `DISCOUNT`, `MAX_RANGE`.

Все имена пакетов пишутся с маленькой буквы. Сокращения допустимы только в случае, если имя пакета слишком длинное: 10 или более символов.

Использование цифр и других символов нежелательно.

ЗАКЛЮЧЕНИЕ

Проведение лабораторных работ с использованием предложенных методических указаний позволяет студентам познакомиться с особенностями разработки графического интерфейса пользователя на основе библиотеки Swing языка Java.

В методических указаниях даны теоретические сведения и описаны практические действия, необходимые для освоения одной из современных сред разработки программных приложений Eclipse.

Все представленные работы имеют необходимые теоретические сведения, практические задания, рекомендации по их выполнению и примеры программных кодов.

Методические указания могут быть использованы для проведения лабораторных работ по дисциплине «Технология программирования» направления 09.03.02 «Информационные системы и технологии» и дисциплине «Программная инженерия» направления 09.03.03 «Прикладная информатика».

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Cole B Java™ Swing, 2nd Edition /Brian Cole, Robert Eckstein, James Elliott, Marc Loy, David Woo – O'Reilly, 2002.
2. The Swing classes (part of the Java™ Foundation Classes (JFC) software) – Oracle Java Se Документация – <http://docs.oracle.com/javase/6/docs/technotes/guides/swing>
3. Base code conventions – Oracle Java Se Документация – <http://www.oracle.com/technetwork/java/javase/documentation/codeconvtoc-136057.html>
4. Портянкин И. Swing: Эффектные пользовательские интерфейсы, 2-е издание – Санкт-Петербург: «Лори», 2011.
5. Шилдт Г. SWING: руководство для начинающих — М.: «Вильямс», 2007.
6. Эккель Б. Философия Java. Библиотека программиста. – М.: "Питер", 2003.

ПРИЛОЖЕНИЕ А. ПРИМЕРЫ ПРОГРАММ С GUI

Отдельный класс для пропорционального изменения размеров графических примитивов может быть использован при задании ширины **width** и высоты **height** текущего контекста отображения

```
public class aRect {
    public double h;
    public static int height; //высота текущего контекста отображения
    public double w;
    public static int width; //ширина текущего контекста отображения
    public double x;
    public double y;
    public static aRect full = new aRect(0.0,0.0,1.0,1.0);
//задание относительных координат прямоугольной области отрисовки
    public aRect(double p1, double p2, double p3, double p4) {
        x=p1;
        y=p2;
        w=p3;
        h=p4;
    }
//вычисление относительных координат одной области внутри другой
    public aRect getARect(aRect p1) {
        return new aRect(p1.x+x*p1.w,p1.y+y*p1.h,w*p1.w,h*p1.h);
    }
//методы вычисления абсолютных координат прямоугольной области отрисовки
    public int X() //начальная координата отрисовки X
        return (int) (x*width);
    }
    public int Y() //начальная координата отрисовки Y
        return (int) (y*height);
    }
    public int W() //ширина области отрисовки
        return (int) (w*width);
    }
    public int H() //высота области отрисовки
        return (int) (h*height);
    }
}
```


ПРИЛОЖЕНИЕ Б. ТАБЛИЦА – КОМПОНЕНТЫ SWING

Компонент	Назначение компонента
JToggleButton	Кнопка с фиксацией. Может быть одной из нескольких таких кнопок в группе, в этом случае нажатие одной кнопки вызывает отпускание другой. Работа группы обеспечивается компонентом ButtonGroup, который должен быть перетащен на форму, а затем назначен свойству buttonGroup
JCheckBox	Переключатель – пункт выбора с независимой фиксацией
JRadioButton	Радиокнопка – элемент двухвариантного выбора с зависимой фиксацией, должен быть одним из нескольких в группе. Работа группы обеспечивается компонентом ButtonGroup.
ButtonGroup	Обеспечивает работу групп компонентов JToggleButton или JRadioButton
JComboBox	Список с возможностью ввода значений
JList	Список
JTextField	Текстовое поле для однострочного ввода и редактирования текста.
JTextArea	Текстовая область для многострочного ввода и редактирования текста.
JScrollPane	Панель с полосами прокрутки
JMenuBar	Меню формы предназначено для расположения в нем компонентов типа JMenu (заголовков меню).
JPopupMenu	Всплывающее меню предназначено для расположения в нем компонентов типа JMenuItem (пунктов меню).
JSlider	Ползунок для плавной регулировки числовых величин, а также связанных с ними программно-регулируемых изменений.
JProgressBar	Полоса для отображения уровня выполнения задачи
JTextPane	Текстовая панель с автоматическим переносом текста.
JEditorPane	Панель текстового редактора
JTable	Таблица
JToolBar	Панель инструментов

ПРИЛОЖЕНИЕ В. ОСНОВНЫЕ СОБЫТИЯ АWT

Класс события	Интерфейс слушателя	Элемент	Методы слушателя	Значение
ActionEvent	ActionListener	Button List MenuItem TextField	actionPerformed()	Нажатие кнопки/ Двойной щелчок /Выбор пункта меню/ Конец редактирования (Enter)
AdjustmentEvent	AdjustmentListener	Scrollbar	adjustmentValueChanged()	Реализация прокрутки
ComponentEvent	ComponentListener	Component	componentHidden() componentMoved() componentResized() componentShown()	Перемещение, изменение размеров, скрытие/видимость
ContainerEvent	ContainerListener	Container	componentAdded() componentRemoved()	Добавление /удаление из контейнера
FocusEvent	FocusListener	Component	focusGained() focusLost()	Получение/потеря фокуса
ItemEvent	ItemListener	Checkbox List	itemStateChanged()	Установка/сброс флага
KeyEvent	KeyListener	Component	keyPressed() keyReleased() keyTyped()	Нажатие/отпускание клавиши
MouseEvent	MouseListener	Component	mouseClicked() mouseEntered() mouseExited() mousePressed() mouseReleased()	Нажатие/отпускание мыши, во- шел/покинул область элемента,
	MouseMotionListener	Component	mouseDragged() mouseMoved()	Перемещение мыши без/с учетом нажа- той кнопки
TextEvent	TextListener	TextComponent	textValueChanged()	Изменения в тексте элемента

ОГЛАВЛЕНИЕ

Введение	3
1. Знакомство с Eclipse.....	4
1.1 Теоретические сведения.....	4
1.2 Задания.....	5
1.3. Указания по созданию программы в среде Eclipse	5
1.4. Указания по запуску и отладке программы в среде Eclipse.....	7
2. Рисование графических примитивов в окне.....	8
2.1. Теоретическая справка	8
2.2. Задания и указания к их выполнению	9
2.3. Примеры построения изображений	12
3. Создание простого GUI-приложения с кнопкой.....	13
3.1 Теоретическая справка	13
3.2 Рекомендации по использованию WindowsBuilder для создания оконного приложения в среде Eclipse.....	14
3.3 Задание и рекомендации по его выполнению	16
3.4 Этапы выполнения работы с использованием Дизайнера форм	17
4. Создание приложения с панелью для рисования.....	20
4.1 Теоретическая справка	20
4.2. Задание и указание по его выполнению.....	21
4.3 Этапы выполнения работы	22
5. Размещение элементов управления в окне	23
5.1 Теоретическая справка	23
5.2 Практические задания	26
5.3 Рекомендации по выполнению заданий	27
6. Обработка событий мыши.....	28
6.1 Теоретическая справка	28
6.2 Задания и рекомендации по их выполнению.....	30
6.3 Примеры реализации обработчиков событий.....	33
7. Расширение функционала GUI-приложений.....	34
7.1. Теоретическая справка	34
7.2 Задания.....	35
7.3 Рекомендации по выполнению задания	36
Требования к результатам выполнения работ и оформлению кода	37
ЗАКЛЮЧЕНИЕ	38
СПИСОК Использованных источников.....	39
Приложение А. Примеры программ с GUI.....	40
Приложение Б. Таблица – Компоненты Swing	41
Приложение В. Основные события AWT	42

Разработка графического интерфейса пользователя

*Методические указания
к выполнению лабораторных работ по дисциплине «Технологии программирования»
для студентов бакалавриата направления
09.03.02 «Информационные системы и технологии» всех форм обучения*

Составитель: Минакова Ольга Владимировна

Подписано в печать 04.07. 2016. Формат 60 х 84 1/16 Уч.-изд. л. 2,8.
Усл.-печ. л. 2,9. Бумага писчая. Тираж 60 экз. Заказ № _____

Отпечатано: отдел оперативной полиграфии
издательства учебной литературы и учебно-методических пособий
Воронежского ГАСУ
396006, Воронеж, ул. 20-летия Октября,84