

РОСЖЕЛДОР

**Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Ростовский государственный университет путей сообщения»
(ФГБОУ ВО РГУПС)**

О.В. Игнатъева

ОБЪЕКТНО ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Учебно-методическое пособие
для практических работ

Ростов-на-Дону
2017

УДК 681.3.06(07) + 06

Рецензент – кандидат технических наук, доцент В.В. Жуков

Игнатьева, О.В.

Объектно ориентированное программирование: учебно-методическое пособие для практических работ / О.В. Игнатьева; ФГБОУ ВО РГУПС. – Ростов н/Д, 2017. – 84 с.

Учебно-методическое пособие посвящено изучению основ программирования графики на объектно ориентированном языке Java. Рассмотрены вопросы работы в дизайнера форм WindowBuilder при использовании средств AWT и Swing для создания GUI-приложений на Java. Описана установка и использование WindowBuilder по базовым компонентам Swing (панели, кнопки, метки, поля редактирования и др.) и по созданию генерируемых изображений средствами Graphics.

Приведены задания на практические работы по созданию GUI-приложений с вводом и выводом текста, создание изображений различной степени сложности средствами Graphics.

Может быть использовано студентами, изучающими дисциплины, связанные с разработкой GUI-приложений на Java.

Предназначено для студентов и магистрантов направлений «Информатика и вычислительная техника», «Информационные системы и технологии» и «Механотроника и робототехника» для углубленного изучения программирования на аудиторных занятиях и самостоятельного изучения материала по дисциплине «Объектно-ориентированное программирование», а также для всех студентов магистратуры, бакалавриата и специалитета различных направлений, изучающих дисциплины по программированию и спецкурсов.

Одобрено к изданию кафедрой «Вычислительная техника и автоматизированные системы управления».

© Игнатьева О.В., 2017

© ФГБОУ ВО РГУПС, 2017

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ	4
ЗНАКОМСТВО С РАЗРАБОТКОЙ GUI ПРИЛОЖЕНИЙ НА JAVA.....	5
Установка WindowBuilder.....	5
Создание простейшей GUI программы при помощи WindowBuilder	7
Код простейшего GUI приложения и его работа.....	11
Работа с дизайнером форм - на примере приложения, считающего нажатия кнопки	13
Создание GUI приложения с вводом, вычислением и выводом -на примере вычисления факториала числа.....	28
Пример выполнения работы D - вывод узора из чисел	37
СОЗДАНИЕ ИЗОБРАЖЕНИЙ СРЕДСТВАМИ JAVA	46
Экранная система координат	46
Графические примитивы Graphics	47
Рисование в собственной панели с использованием Graphics	53
Создание приложения с собственной панелью	55
Генерирование изображения с использованием цикла.....	64
Создание сложного изображения из повторяющегося простого	70
ЗАДАНИЯ ДЛЯ ПРАКТИЧЕСКИХ РАБОТ	75
Практическая работа №1 - создание узора из чисел	75
Практическая работа №2 - создание статического изображения	76
Практическая работа №3 - создание динамического изображения.....	77
Практическая работа №4 - создание сложного изображения из повторяющегося простого	78
ЗАКЛЮЧЕНИЕ	81
БИБЛИОГРАФИЧЕСКИЙ СПИСОК.....	82

ВВЕДЕНИЕ

Язык Java является одним из наиболее популярных как в коммерческой разработке, так и в обучении студентов основам объектно-ориентированного программирования. Язык Java с равным успехом используют для создания веб-систем, для написания мобильных и десктоп-приложений. При разработке десктоп-приложений часто используют библиотеки AWT и Swing, являющиеся стандартными средствами Java и позволяющие создавать кроссплатформенные GUI приложения. Для быстрого создания интерфейса приложения часто используется визуальный дизайнер форм WindowBuilder, который прост в изучении и в использовании. Эта простота явилась причиной выбора WindowBuilder в качестве основного инструмента для знакомства с разработкой GUI приложений под Java в рамках дисциплины «Объектно-ориентированное программирование».

Существует множество источников для изучения Java – это и книги, и обучающие онлайн курсы, и многочисленные статьи по тем или иным вопросам на популярных Интернет-ресурсах. Но по WindowBuilder очень мало источников, особенно на русском языке. Также крайне плохо освещены вопросы старта новичка в GUI разработке на Java и особенно при работе с WindowBuilder.

Поэтому в данных методических указаниях автором собран в одно целое материал как по началам работы в WindowBuilder, так и в разработке GUI приложений в целом. В частности, здесь собрана в одном месте минимально необходимая информация по установке и использованию WindowBuilder, базовым компонентам Swing (панели, кнопки, метки, поля редактирования и др.), созданию генерируемых изображений средствами Graphics. Приведены задания на практические работы по созданию GUI приложений с вводом и выводом текста, по созданию изображений различной степени сложности встроенными средствами языка Java.

Данные методические указания можно применять как в процессе выполнения практических работ в аудитории, так и для самостоятельного изучения основ разработки GUI приложений на Java.

ЗНАКОМСТВО С РАЗРАБОТКОЙ GUI ПРИЛОЖЕНИЙ НА JAVA

Графический пользовательский интерфейс (GUI - Graphical User Interface) является стандартом де-факто всех приложений, взаимодействующих с конечным пользователем. Поэтому студенту, изучающему программирование на любом современном языке, крайне важно изучить возможности этого языка именно в части создания GUI. В Java существуют мощные средства для создания GUI. Библиотеки AWT и Swing являются наиболее проверенными из таких средств. По этим библиотекам существует масса литературы и справочников, нюансы работы с ними описаны на тысячах форумов и в множестве статей. Поэтому в случае необходимости получения ответа на конкретный вопрос Вы легко его найдете через google.com или через другой поисковик.

Здесь же мы с Вами по шагам познакомимся с созданием простейших GUI приложений на AWT/Swing с применением дизайнера форм WindowBuilder. После проработки этого раздела Вы сможете создать простой GUI для написания вычислительных задач, которые на вход получают набор чисел или строк и на выходе выдают числа или строки.

Установка WindowBuilder

WindowBuilder является одним из самых простых и функциональных дизайнеров форм для создания GUI в Java. Мы будем использовать именно его. Для того чтобы использовать WindowBuilder, Вам нужно иметь на Вашем компьютере установленный JDK и установленный Eclipse.

Чтобы установить WindowBuilder в Ваш Eclipse, Вам нужно выполнить следующие шаги:

1. Выбрать "Help > Install New Software..." в главном меню

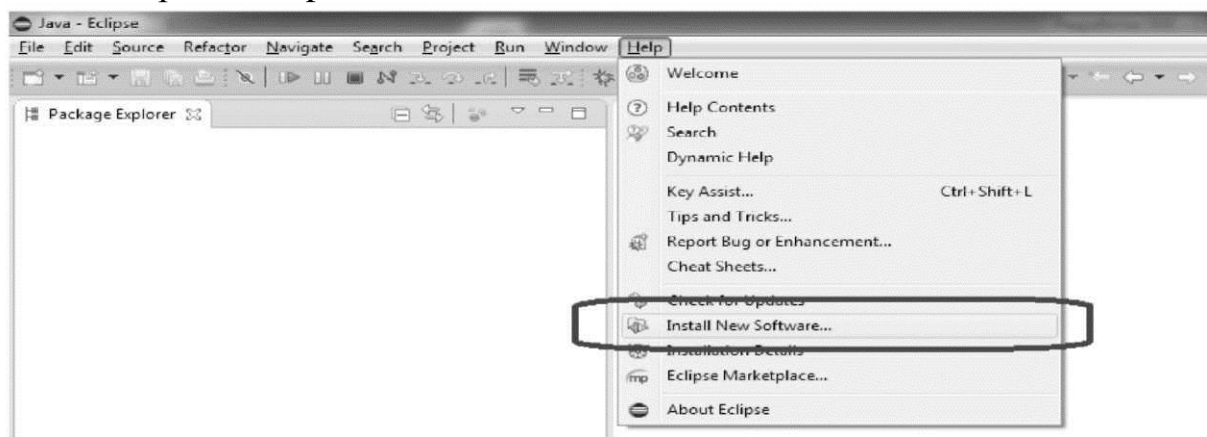


Рис. 1.1. Выбираем «Install new software...»

2. Скопировать ссылку

<http://download.eclipse.org/windowbuilder/WB/integration/4.4/> в поле "Work with" окна "Install":

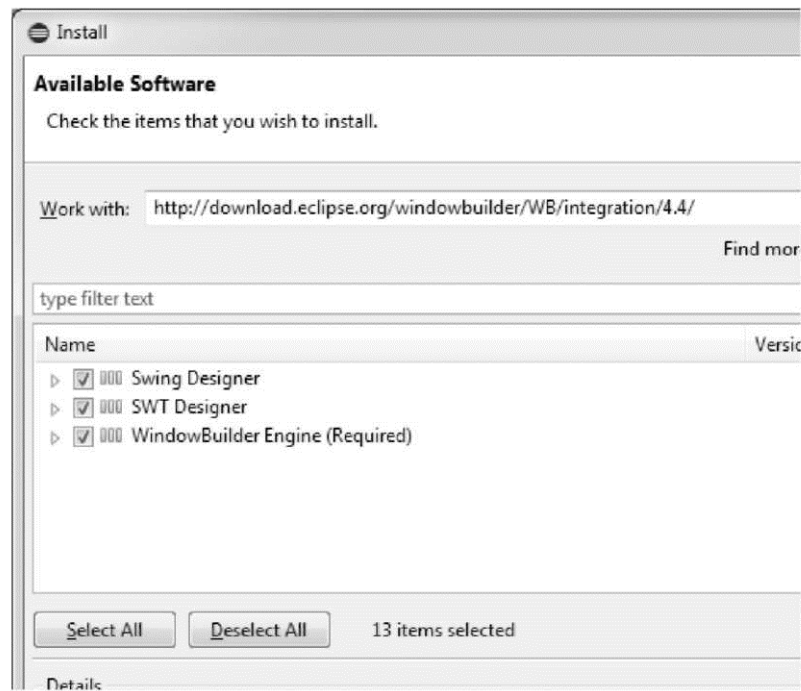


Рис. 1.2. Копируем ссылку на WindowBuilder

3. Нажать кнопку Next. После некоторого времени (до 10-15 минут), требующегося на поиск и выкачку необходимых компонентов, Вы увидите такую форму Install:

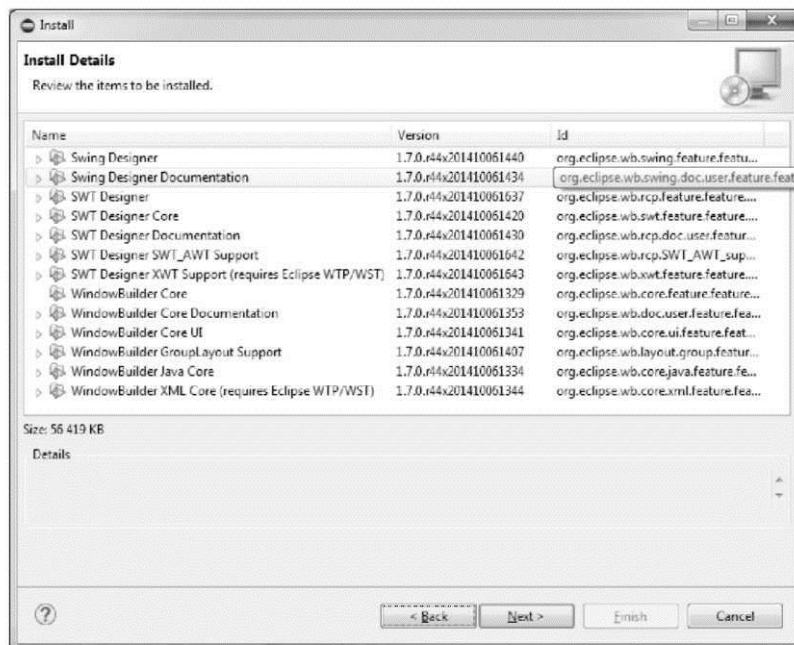


Рис. 1.3. Подтверждаем наш выбор

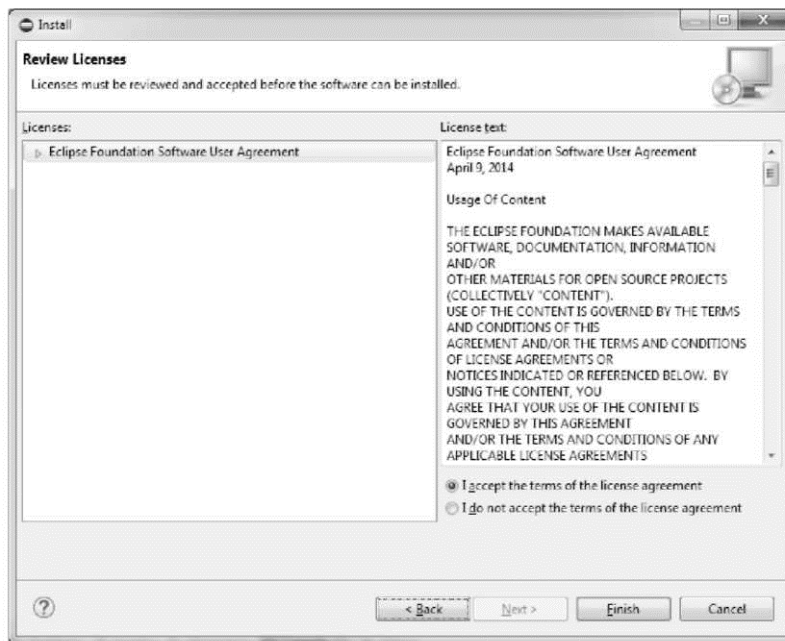


Рис. 1.4. Соглашаемся с лицензией

4. Снова нажать Next. Появится текст лицензии, который нужно принять, выбрав пункт «I accept...». После этого станет доступна кнопка Finish, которую надо нажать.
5. После этого Eclipse выкачает все необходимые компоненты -ему потребуется несколько минут. И появится такое окно:

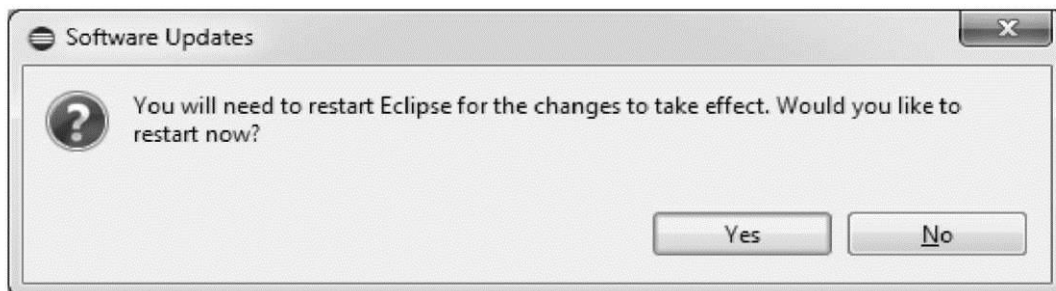


Рис. 1.5. Перезапускаем Eclipse

6. После перезапуска Eclipse можно начинать использовать WindowBuilder для создания GUI приложений.

Создание простейшей GUI программы при помощи WindowBuilder

Для начала нужно создать новый проект в Eclipse. Для этого нажимаем на вкладку File , далее New => Java Project.

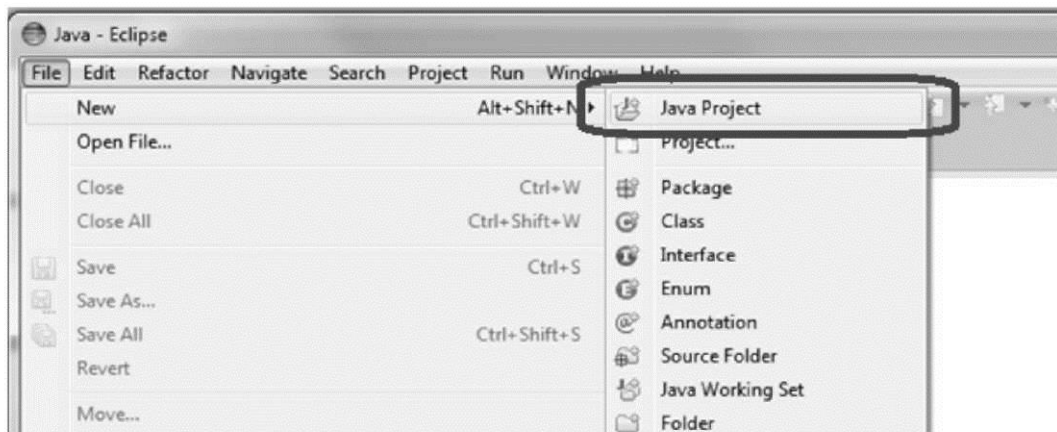


Рис. 1.6. Создание нового проекта

В открывшемся окне напишем имя проекта, пусть будет «GUI_Test»:

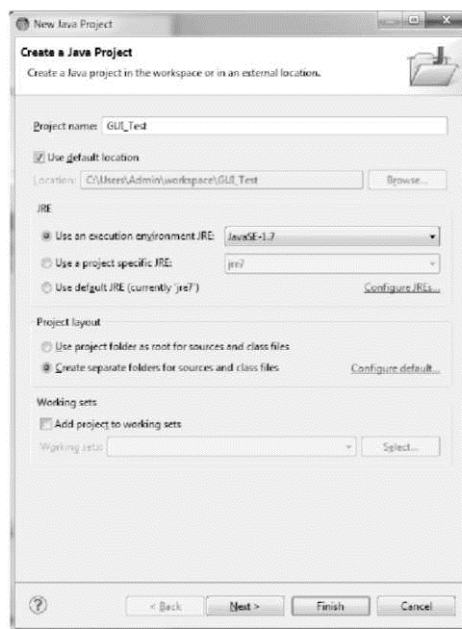


Рис. 1.7. Имя проекта

Далее, в Package Explorer кликаем по только что созданному проекту GUI_Test правой кнопкой мыши и выбираем New => Other (можно использовать сочетание клавиш Ctrl + N).

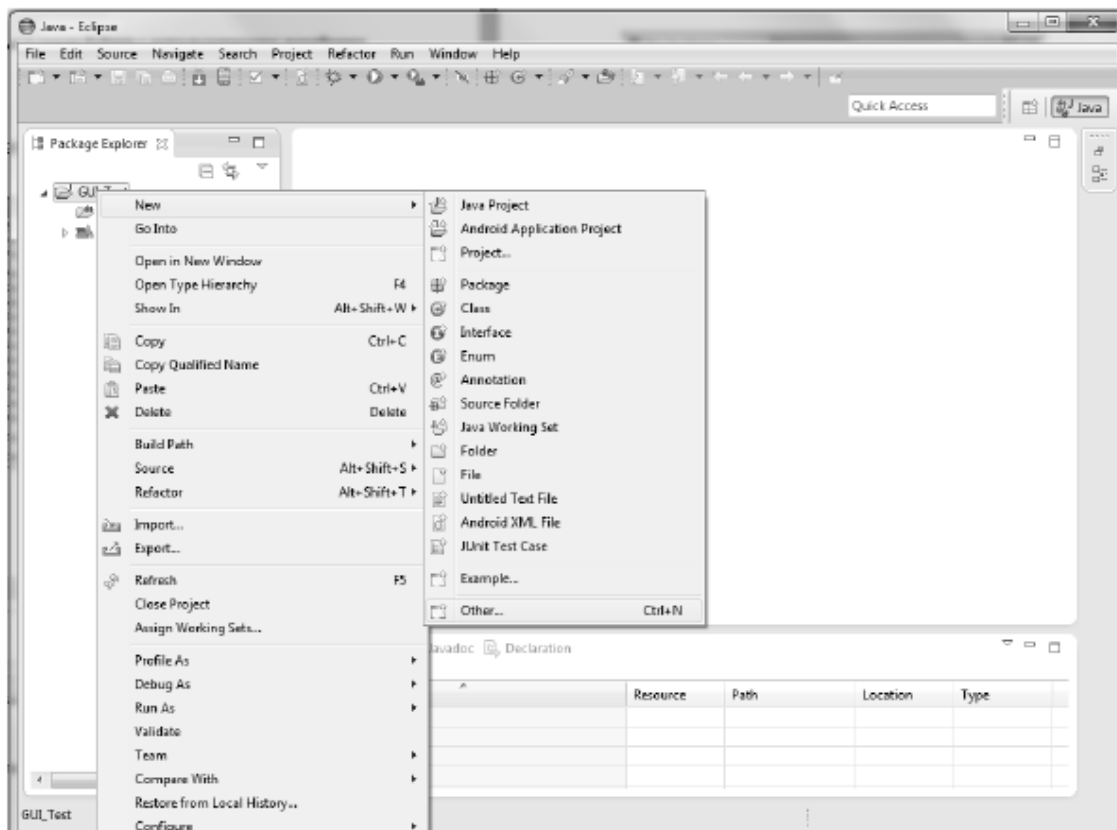


Рис. 1.8. Выбор New/Other...

В открывшемся окне выбираем Window Builder => Swing Designer => Application Window.

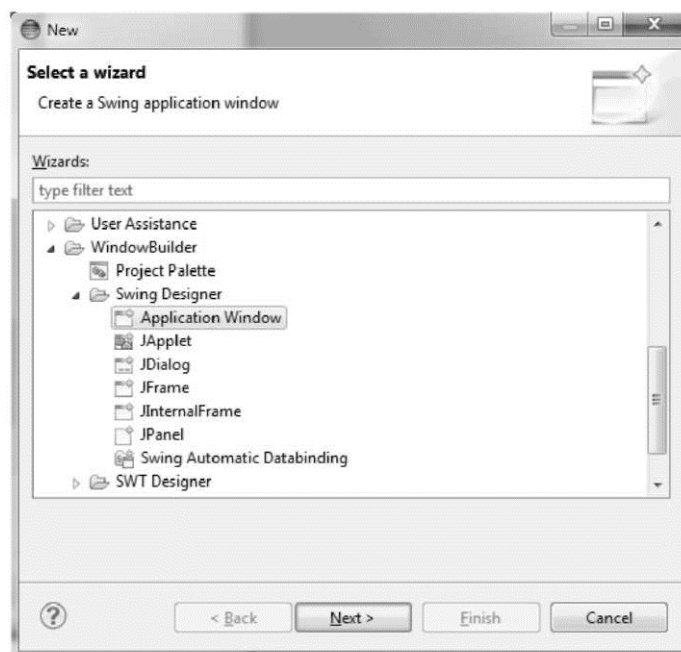


Рис. 1.9. Создание Application Window

И нажимаем Next.

Далее нужно ввести имя нашего приложения. Назовем его, к примеру, «MyFirst_GUI». И нажимаем кнопку Finish

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

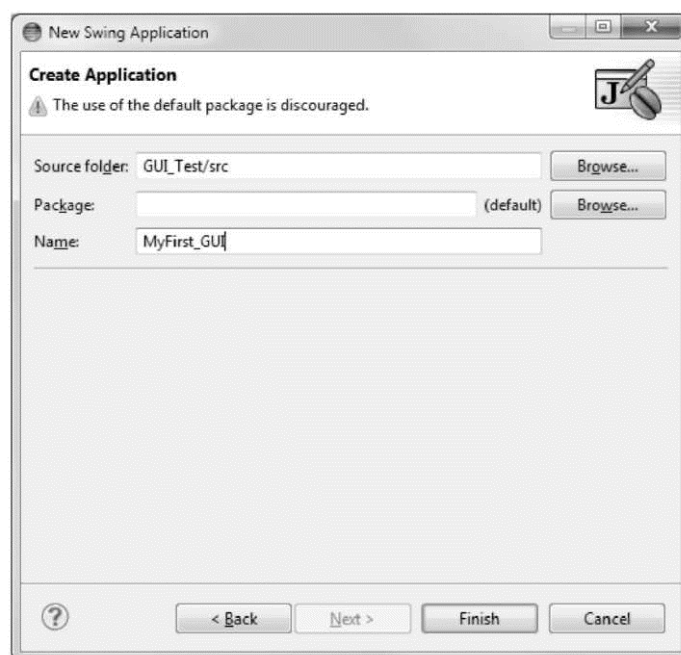


Рис. 1.10. Имя программы

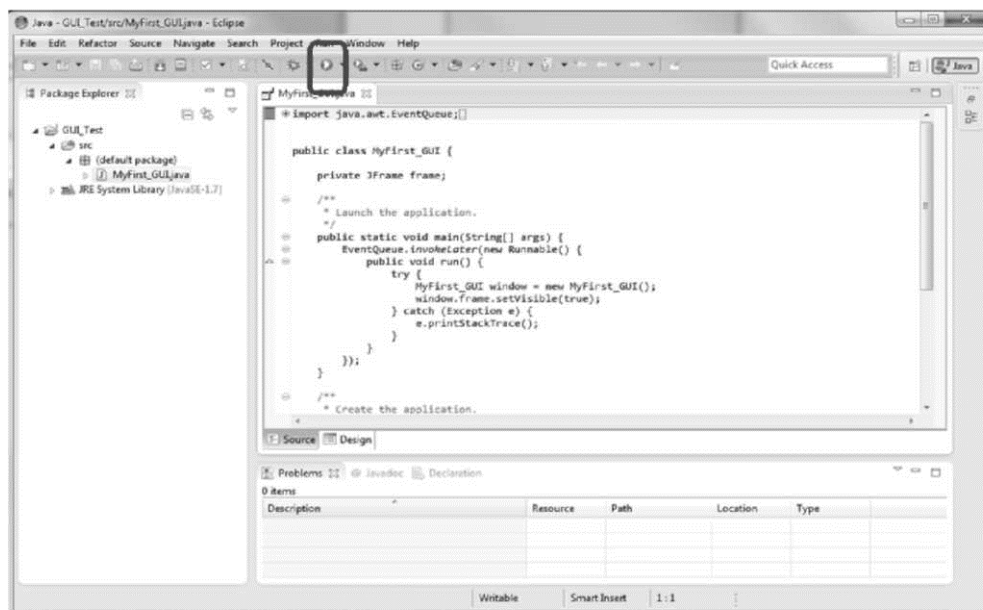


Рис. 1.11. Внешний вид Eclipse после создания GUI приложения

Нажмите кнопку Run, как показано на рис. 1.11, чтобы запустить наше приложение на выполнение. В итоге Вы увидите наше приложение в работе:

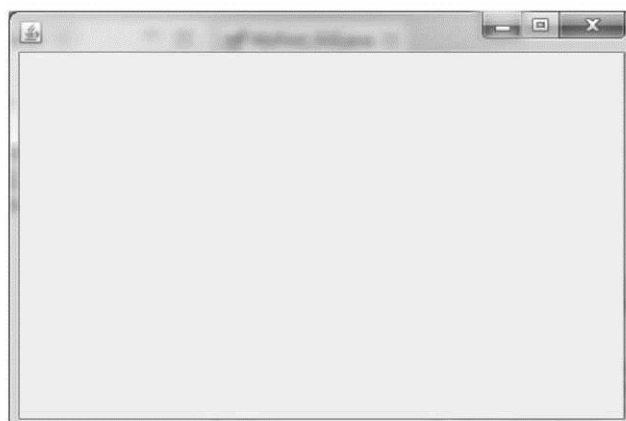


Рис. 1.12. Наше GUI приложение в работе

Поздравляем Вас. Вы только что создали и запустили свое первое GUI приложение на Java!

Код простейшего GUI приложения и его работа

Вы сможете создавать простые GUI приложения, сильно не разбираясь внутри кода, который генерируется WindowBuilder. Как Вы увидите позже, очень легко создавать GUI, используя визуальные инструменты и лишь изредка поправляя или добавляя пару строк кода вручную.

Однако для некоторых читателей очень важно понимать, как ЭТО устроено. Поэтому изучим внимательно код класса MyFirst_GUI, который собственно и представляет собой наше созданное приложение.

Если Вам не интересно знать его устройство - можете смело пропустить следующий раздел.

Как это работает

Класс `MyFirst_GUI` содержит статический метод `main()` - точку входа в Java приложение и еще два метода: метод `MyFirst_GUI()` (конструктор класса) и метод `initialize()`. Также класс содержит поле `JFrame frame`, которое и есть окно нашего приложения.

Логика работы кода такая:

1. Когда мы запускаем нашу программу, реализованную классом `MyFirst_GUI`, на выполнение (например, нажав кнопку `Run` в `Eclipse`), вызывается метод `MyFirst_GUI.main()`.
2. В методе `MyFirst_GUI.main()` вызывается метод `EventQueue.invokeLater()`, в котором обрабатываются события от нашего приложения.
3. Наше приложение создается внутри потока выполнения, который создается конструкцией `new Runnable() { ... }`. С точки зрения Java `new Runnable() { ... }` - это анонимный класс.
4. Внутри анонимного класса переопределяется метод `run()`, где собственно создается объект нашего класса `MyFirst_GUI`. Для этого вызывается конструктор `MyFirst_GUI()`.
5. В конструкторе `MyFirst_GUI()` есть единственная строка кода - вызов метода `initialize()`.
6. В методе `initialize()` собственно и создается окно `JFrame`, и задаются его параметры.

Код простейшего GUI приложения

```
// Импорт нужных классов import java-
va.awt.EventQueue;
import javax.swing.JFrame;

// класс приложения public class
MyFirst_GUI {

    // Окно приложения
    private JFrame frame;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        // Обработка событий создаваемого анонимного класса
        // new Runnable() {
        EventQueue.invokeLater(new Runnable() {
            // run() - это метод потока выполнения
            // В нем выполняется наше приложение
            public void run() {
                try {
                    // Создание объекта нашего класса
```

```

        MyFirst_GUI window = new
MyFirst_GUI();

        // Окно приложения делается видимым
        // Пока его не закроют - наше
приложение

        // будет работать
        window.frame.setVisible(true);
    } catch (Exception e) {
        e.printStackTrace();
    }
    }
});
}

// Конструктор класса нашего приложения
/**
 * Create the application.
 */
public MyFirst_GUI() {
    initialize();
}
/**
 * Initialize the contents of the frame. */
private void initialize() {
    // Создается окно нашего приложения
    frame = new JFrame();
    // Задаются границы окна
    frame.setBounds(100, 100, 450, 300);
    // Задается операция в случае закрытия окна
    // EXIT_ON_CLOSE - при закрытии окна - выход из программы
    frame.setDefaultCloseOperation(JF rame.EXIT_ON_CLOSE);
}
}

```

Работа с дизайнером форм - на примере приложения, считающего нажатия кнопки

Если Вы пропустили предыдущий раздел и не стали разбираться, как устроен код приложения, то Вы все равно сможете создавать простейшие GUI приложения. Когда Вам захочется разобраться в логике работы сгенерированного кода, Вы можете вернуться и изучить пропущенный раздел.

Теперь мы приступим к изучению возможностей дизайнера форм. Мы создадим простое приложение, которое считает, сколько раз пользователь нажал на кнопку, и выводит сообщение о количестве нажатий.

Внешний вид окна счетчика нажатий кнопки

Сделаем приложение с таким интерфейсом. При запуске выводится сообщение «Кнопка нажата раз: 0». После каждого следующего нажатия сообще-

ние меняется на «Кнопка нажата раз: 1», «Кнопка нажата раз: 2» и т. д. Ниже представлен вид окна после 5 нажатий на кнопку:

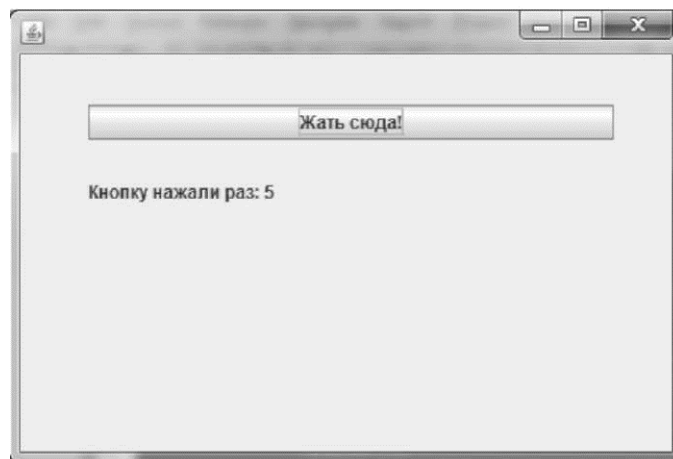


Рис. 1.13. Внешний вид приложения «Счетчик нажатий на кнопку»

Переключение в режим дизайнера форм

Чтобы начать использовать дизайнер форм, нужно переключиться в режим дизайнера форм:

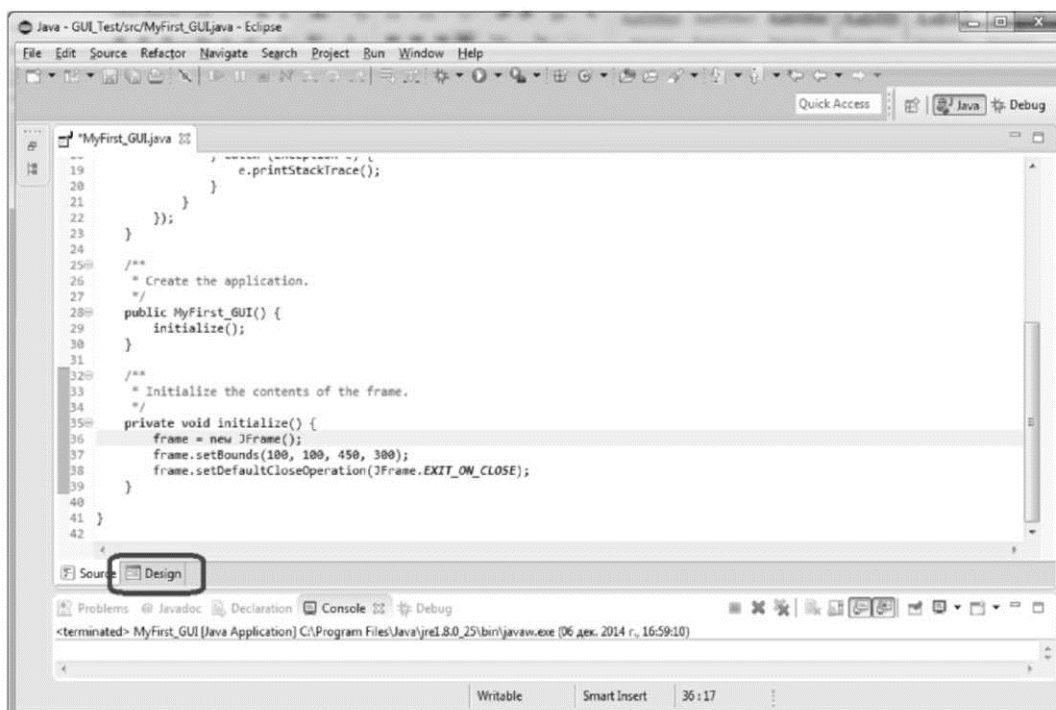


Рис. 1.14. Переключение в режим дизайнера форм

После переключения Вы увидите Eclipse в таком виде:

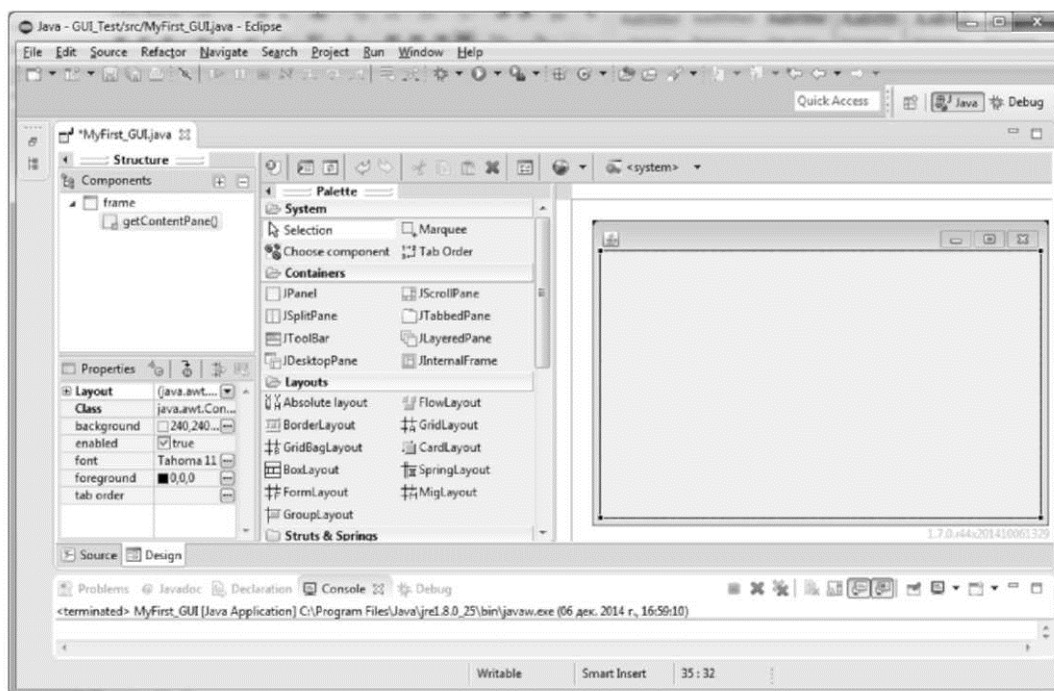


Рис. 1.15. Eclipse в режиме дизайнера форм

На дизайнере форм Вы можете увидеть внешний вид редактируемого окна (справа на рис. 1.15), виды (View) Structure (Структура) и Palette (Палитра) (слева на рис. 1.15).

В виде (View) Palette собраны элементы, из которых собственно строится окно. Они сгруппированы по своему назначению на Containers, Layouts, Components и другие. В рамках данных методических указаний мы познакомимся лишь с несколькими из них.

Вид (View) Structure содержит два раздела. Вверху расположен раздел Components, где отображаются все элементы, из которых состоит окно. Внизу расположен раздел Properties, где отображаются и могут редактироваться свойства активного элемента.

Добавление Absolute Layout в окно приложения

Для создания «Счетчика нажатий кнопки» мы будем использовать ранее созданный класс MyFirst_GUI.

Для начала добавим Absolute Layout. Это позволит размещать элементы GUI в произвольных местах нашего окна. Absolute Layout находится в Palette\Layouts

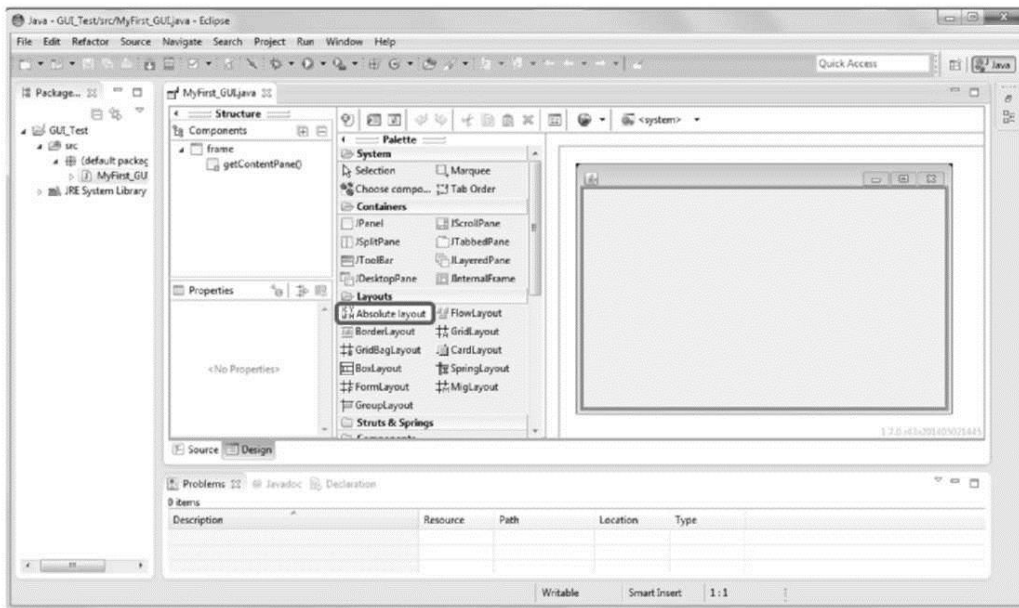


Рис. 1.16. Добавление Absolute Layout

Добавление кнопки в окно приложения

Добавим кнопку, нажатия которой будут считаться в приложении. Кнопка находится в Palette\Components и называется JButton. Нужно выбрать JButton в Палитре, затем курсор переместить вправо на внешний вид редактируемого окна:

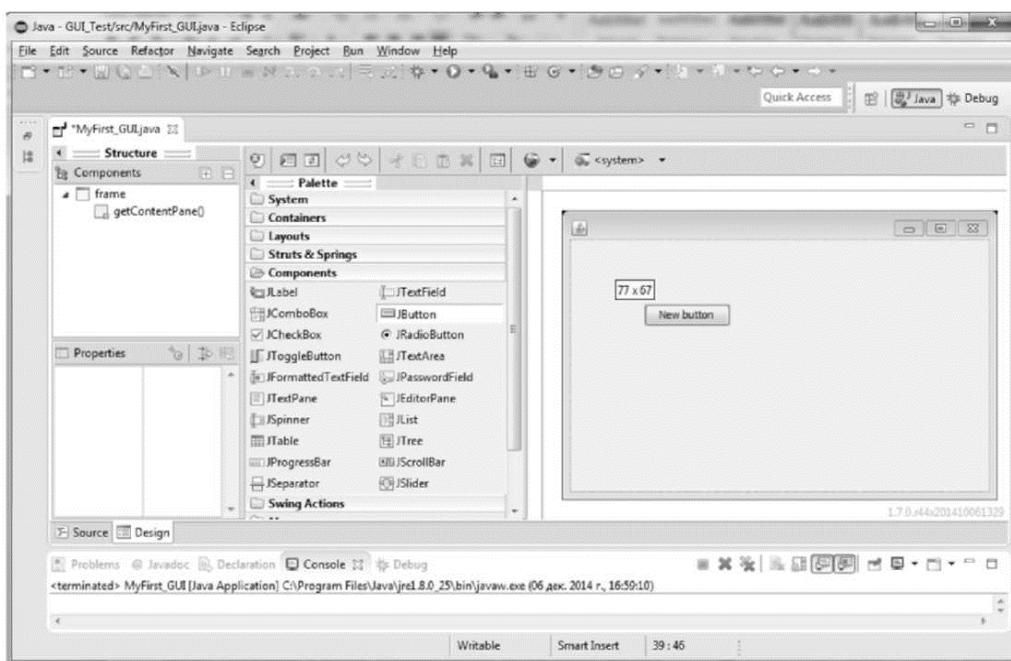


Рис. 1.17. Процесс добавления кнопки JButton

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. 1.18, отмечен свежедобавленный элемент button).

Сразу после добавления кнопки мы имеем возможность задать ей текст, который в ней будет отображаться. Воспользуемся этой возможностью и введем текст «Жать сюда!», нажмем Enter:

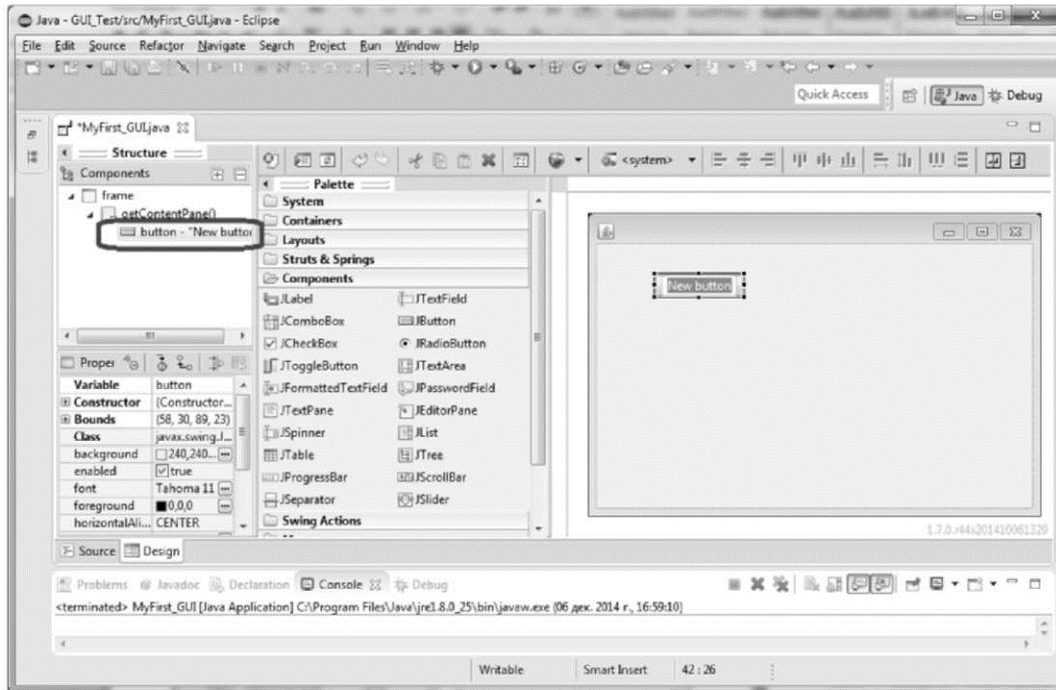


Рис. 1.18. JButton только что добавлен на форму

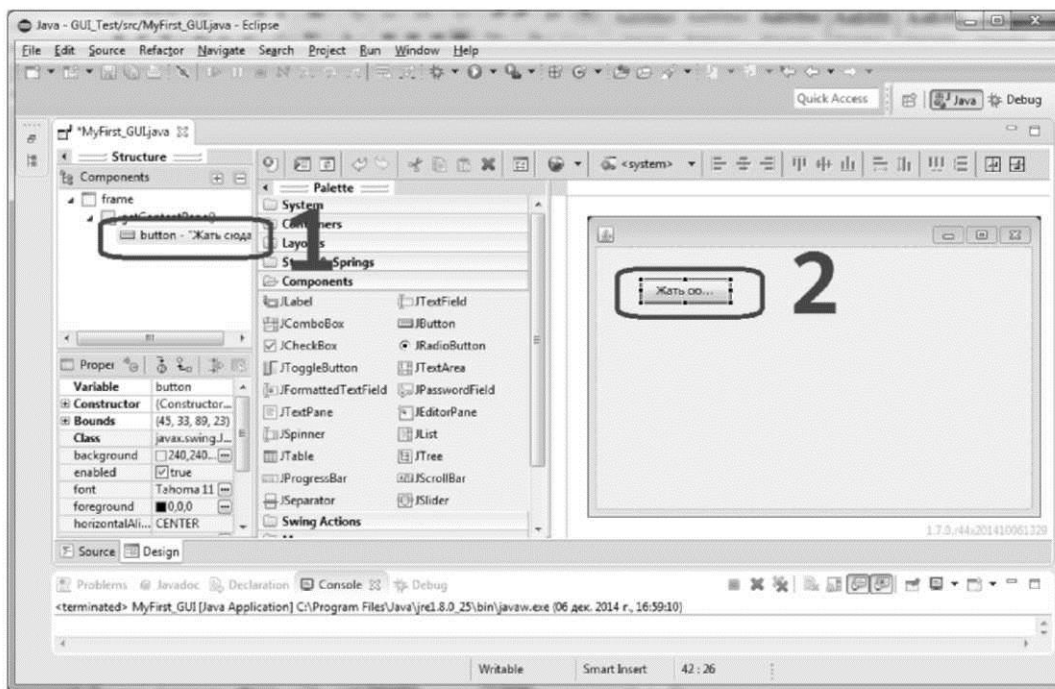


Рис. 1.19. Введен текст в кнопку

И снова обратите внимание на список Components - текст кнопки «Жать сюда!» отобразился в Components (метка 1 на рис. 1.19).

И обратите внимание на то, что наш текст не помещается в кнопку (метка 2 на рис. 1.19). Это легко исправить - нужно подвести курсор мыши к одному из маленьких черных квадратиков вокруг границ кнопки и, захватив его, перетащить границу кнопки в нужное место.

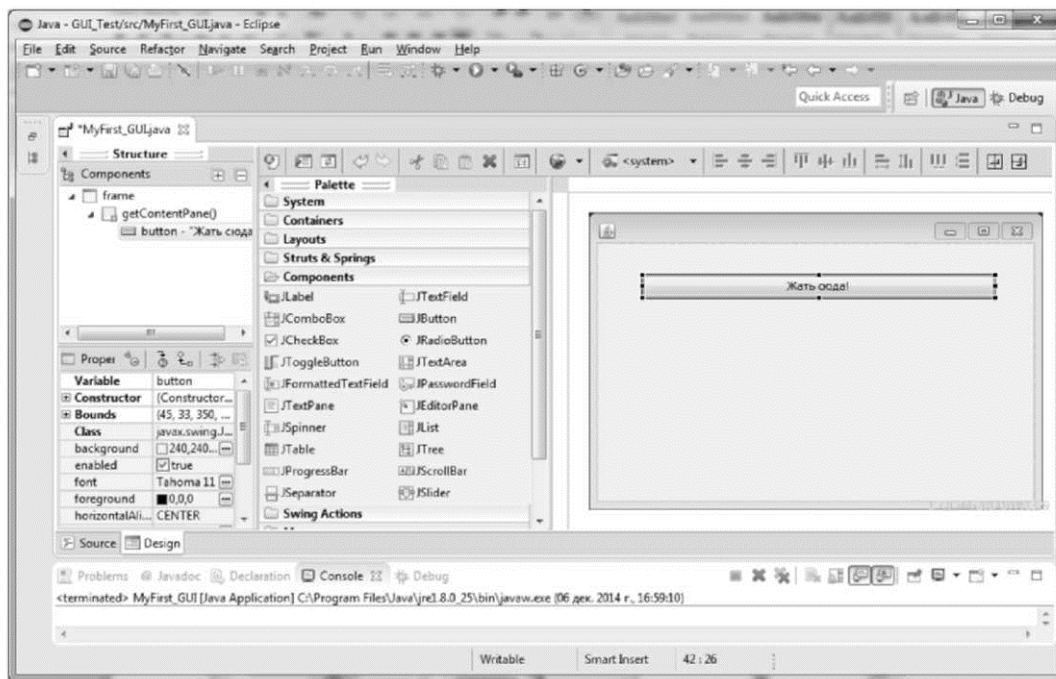


Рис. 1.20. JButton после изменения размера

Добавление метки в окно приложения

Добавим метку, в которой будем выводить текст «Кнопка нажата раз: 0». Метка находится в Palette\Components и называется JLabel. Нужно выбрать JLabel в Палитре, затем курсор переместить вправо на внешний вид редактируемого окна:

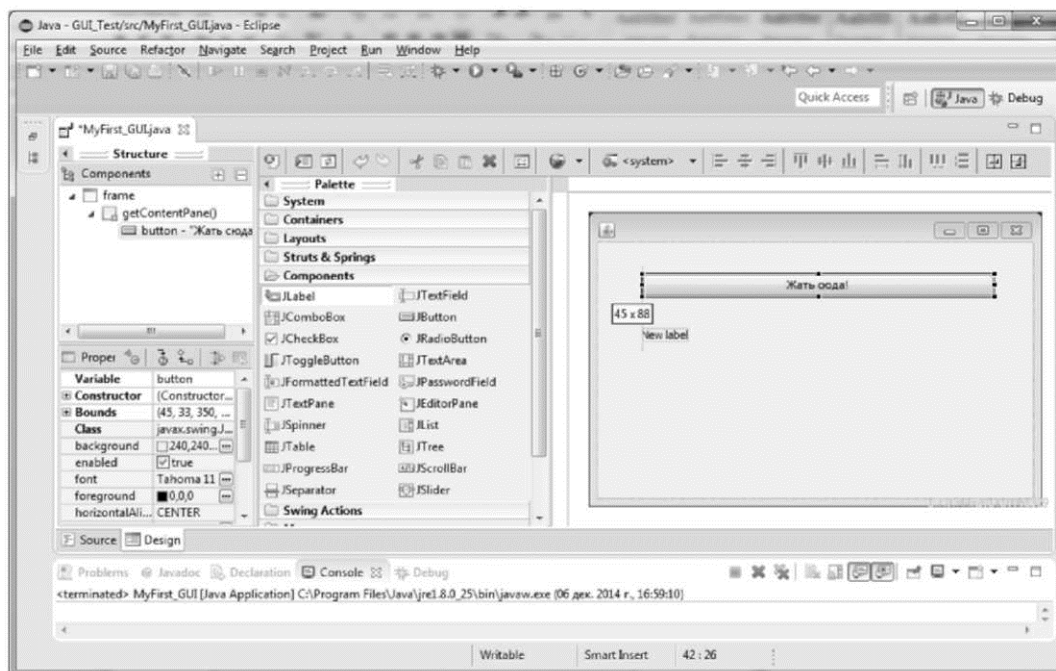


Рис. 1.21. JLabel в процессе добавления

Нужно выбрать место для добавления метки, кликнуть левой кнопкой мыши - метка зафиксируется. Потом надо ввести текст метки «Кнопка нажата раз: 0». И изменить размер метки так, чтобы текст вмещался в нее.

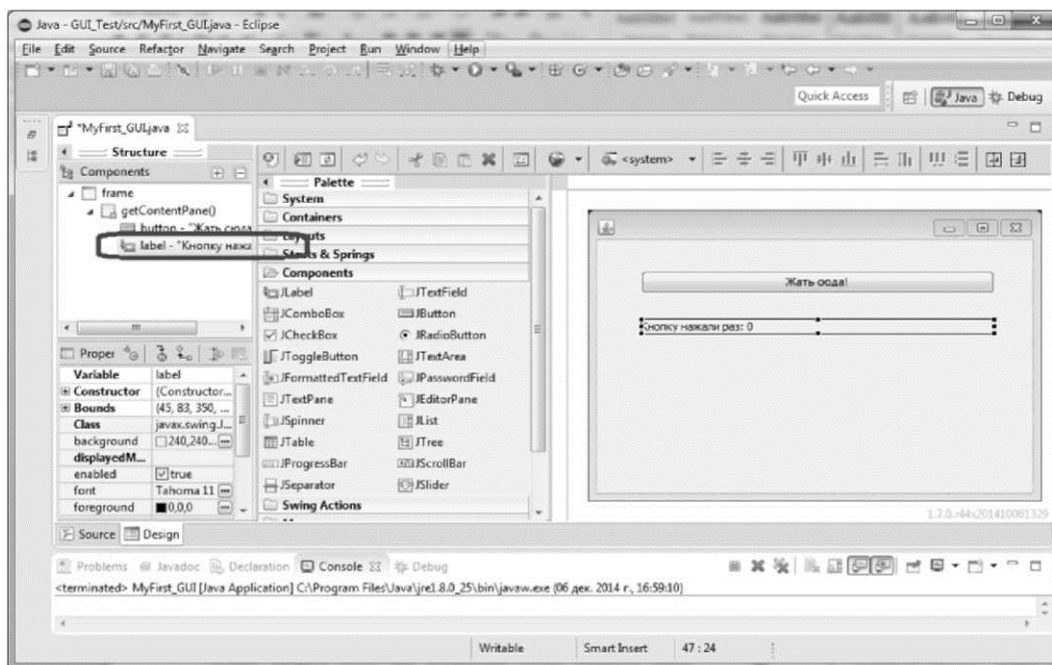


Рис. 1.22. JLabel добавлена

И снова обратите внимание на рис. 1.22, на список Components -туда добавился элемент label с текстом «Кнопка нажата раз: 0».

Если Вы сейчас запустите программу на выполнение, то увидите, что внешний вид полностью соответствует нашим требованиям:

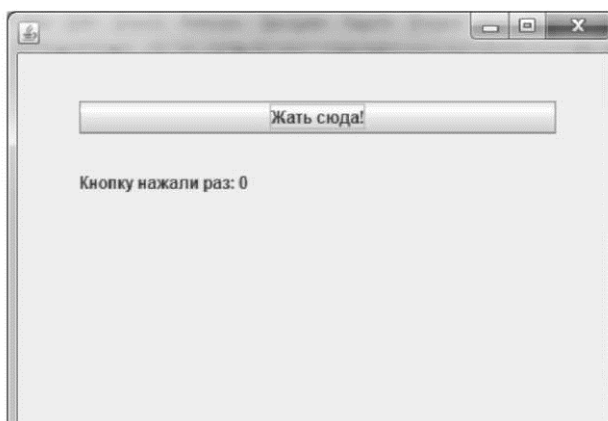


Рис. 1.23. Приложение в работе

Но программа не меняет текст в метке при нажатии на кнопку. Даже если Вы нажмете на кнопку 10 раз, текст в метке все равно будет оставаться «Кнопка нажата раз: 0».

Переключение в режим редактирования кода

Перейдет из режима дизайнера форм в редактор кода:

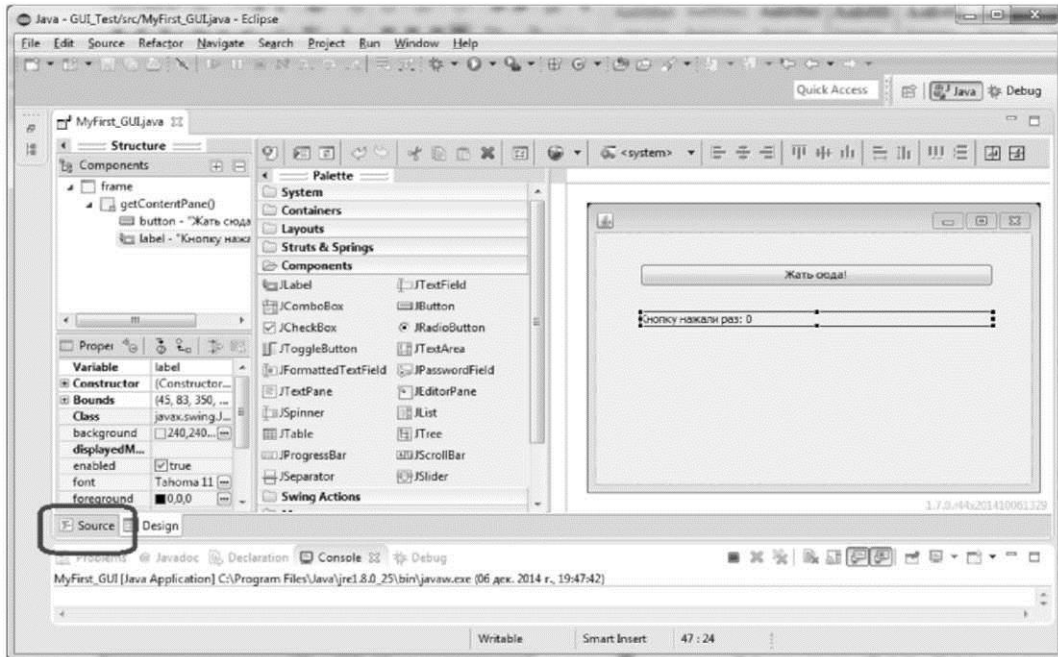


Рис. 1.24. Переход в режим редактирования кода

Вы увидите измененный текст класса MyFirst_GUI:

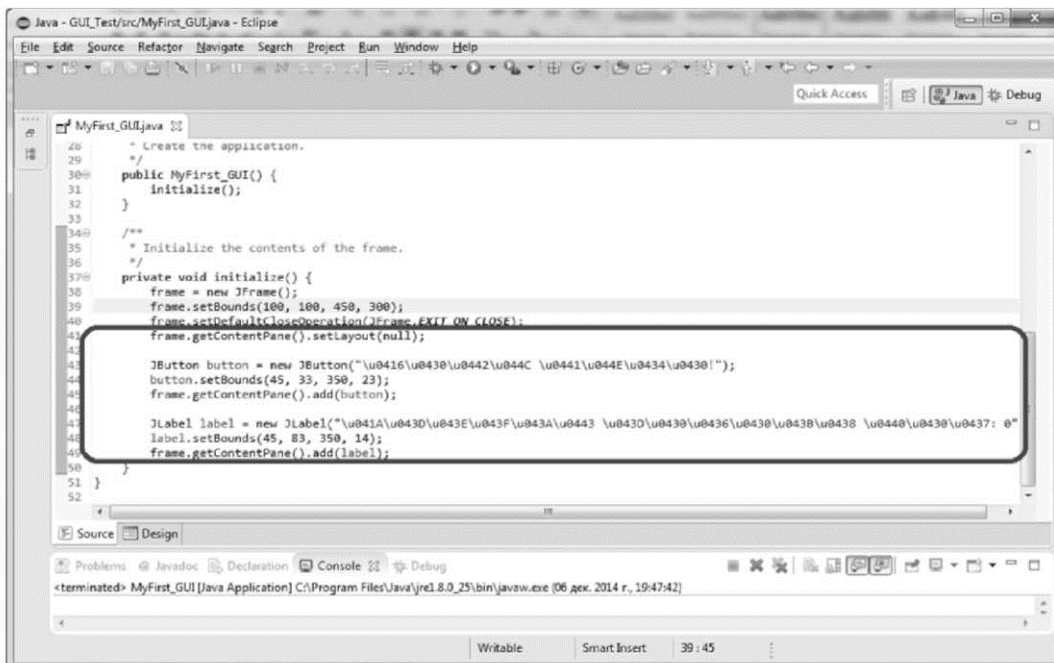


Рис. 1.25. Режим редактирования кода

На рис. 1.25 отмечен тот фрагмент кода, который добавился в результате добавления в окно кнопки и метки. Изучим внимательно новый код.

Код после добавления кнопки и метки

Новый код выделен жирным. Легко заметить, что изменения четко локализованы в двух местах - изменился список импортов и изменился метод `initialize()`. Все остальное осталось таким же, как было сразу после создания класса `MyFirst_GUI`. Новый код прокомментирован подробно.

```
import java.awt.EventQueue;
import javax.swing.JFrame;

// Импортируются классы JButton и JLabel
import javax.swing.JButton;
import javax.swing.JLabel;

public class MyFirst_GUI {

    private JFrame frame;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    MyFirst_GUI window = new
MyFirst_GUI();
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public MyFirst_GUI() {
        initialize();
    }

    /**
     * Initialize the contents of the frame. */
    private void initialize() {
        frame = new JFrame();
        frame.setBounds(100, 100, 450, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        // Добавлен Absolute Layout
        frame.getContentPane().setLayout(null);
    }
}
```

```

// Создается локальная переменная button
JButton button
// создается кнопка с надписью «Жать сюда!»
// текст написан в Unicode:
// \u0416 - этот код буквы 'Ж'
// \u0430 - этот код буквы 'а' и т.д.
= new JButton("\u0416\u0430\u0442\u044c"
+ " \u0441\u044e\u0434\u0430!");
// Задаем размещение и размеры кнопки
button.setBounds(45, 33, 350, 23);
// Добавляем кнопку на frame - окно приложения
frame.getContentPane().add(button);

// Создается локальная переменная label
JLabel label
// создается метка с надписью «Кнопку нажали раз: 0»
// текст написан в Unicode:
= new JLabel("\u041a\u043d\u043e\u043f\u043a\u0443" + "
\u043d\u0430\u0436\u0430\u043b\u0438 \u0440\u0430\u0437:
0");
// Задаем размещение и размеры метки
label.setBounds(45, 83, 350, 14);
// Добавляем метку на frame - окно приложения
frame.getContentPane().add(label);

```

Добавление поля к классу MyFirstGUI

Чтобы считать количество нажатий на кнопку, нам требуется переменная. Назовем ее counter (счетчик). И добавим ее как поле в класс MyFirst_GUI. Добавим ее вручную в редакторе кода. После изменения начало кода класса будет выглядеть так:

```

public class MyFirst_GUI {
private JFrame frame;
    // counter будет считать количество нажатий на кнопку
    // при запуске программы количество нажатий = 0
private int counter = 0;
/**
 * Launch the application.
 */
public static void main(String[] args) {

```

Добавление обработчика события для кнопки

Текст сообщения в метке должен меняться при каждом нажатии кнопки button. Чтобы реализовать это, мы добавим обработчик нажатия кнопки к нашей кнопке button. Для этого мы должны вернуться в режим дизайнера форм, нажав закладку «Design».

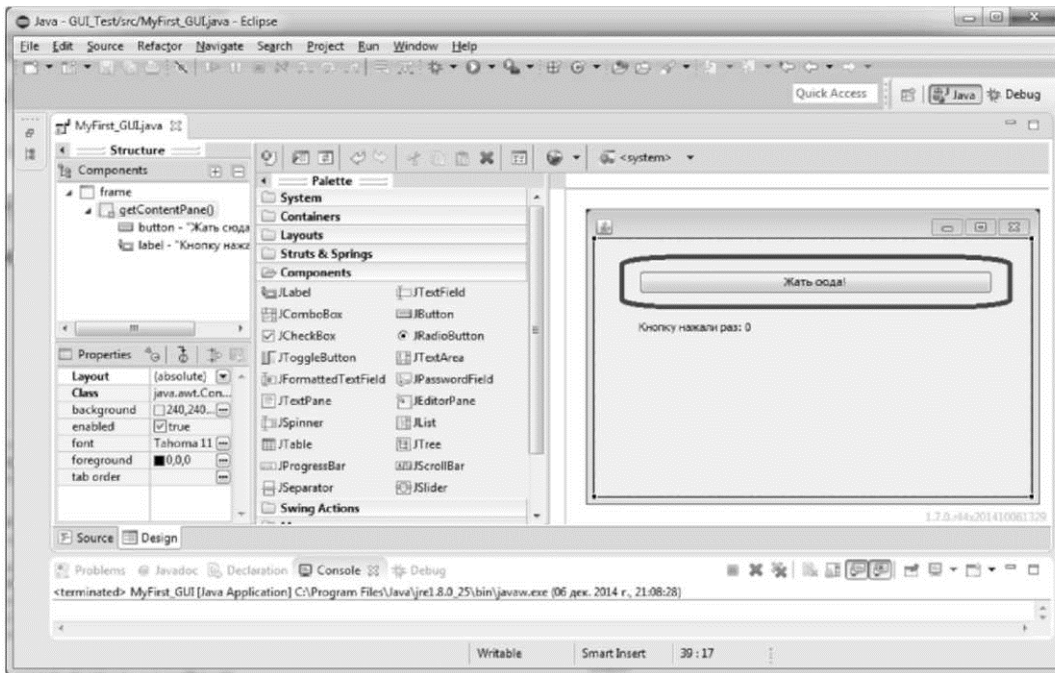


Рис. 1.26. Добавление обработчика на нажатие кнопки

Для добавления обработчика на кнопку нужно дважды кликнуть левой кнопкой мыши по кнопке - на рис.1.26 она отмечена. После этого в код будет добавлен пустой обработчик, и Eclipse автоматически перейдет в режим дизайнера форм. При этом курсор будет поставлен в начало заголовка метода-обработчика, как это показано на следующем рисунке:

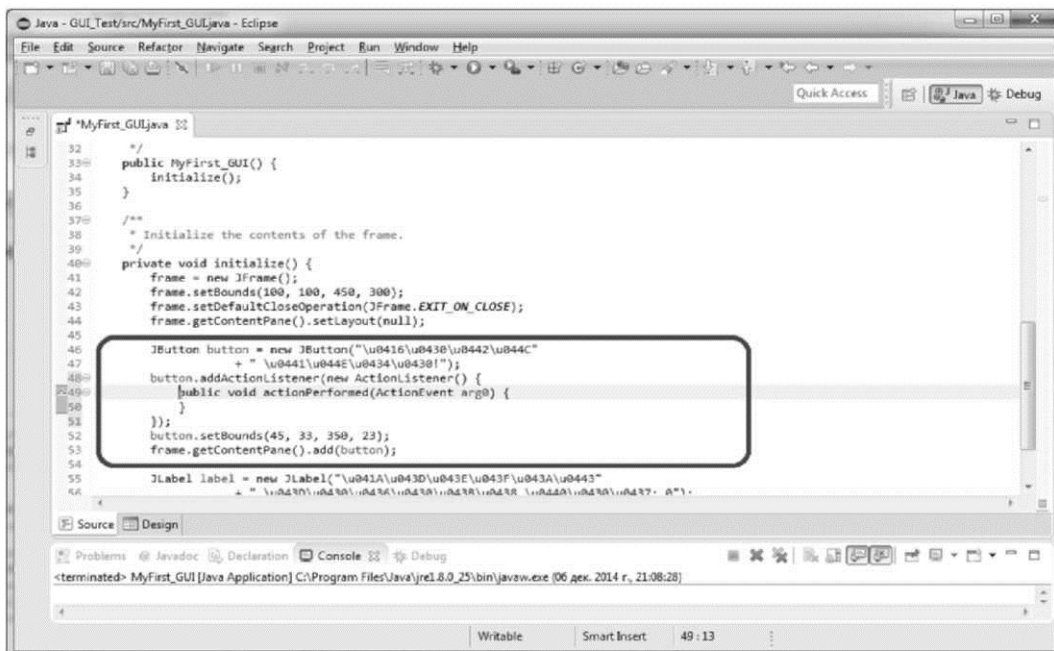


Рис. 1.27. Обработчик на нажатие кнопки добавлен

Рассмотрим внимательно добавленный код - он выделен жирным:

```
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
```

```

/**
 * Initialize the contents of the frame.
 */
private void initialize() {
frame = new JFrame();
frame.setBounds(100, 100, 450, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.getContentPane().setLayout(null);

    JButton button = new JButton(
ton("\u0416\u0430\u0442\u044c"
        + " \u0441\u044e\u0434\u0430!");

    // К кнопке button добавляется «слушатель» действий -
    // по сути «слушатель» - это анонимный класс
    // создаваемый так: new ActionListener() {}
    button.addActionListener(new ActionListener() {
    // этот метод - обработчик события «нажатие»
    // он будет вызываться всякий раз, когда
    // кнопку будут нажимать
        public void actionPerformed(ActionEvent e) {
        }
    });
    button.setBounds(45, 33, 350, 23);
    frame.getContentPane().add(button);
}

```

Конвертация метки из локальной переменной в поле класса

В обработчике нажатия кнопки actionPerformed нам нужно добавить код, который будет изменять текст в метке label. Но сейчас метка label - это локальная переменная метода initialize. И из обработчика нажатия кнопки эта переменная недоступна. Нужно превратить ее в поле класса. Для этого нужно внести небольшие изменения в код.

БЫЛО:

```

public class MyFirst_GUI {

    private JFrame frame;
    private int counter = 0;

    /**
     * Launch the application. */
    public static void main(String[] args) {

        private void initialize() {

            // Сейчас label - локальная переменная
            JLabel label = new
                JLabel("\u0416\u0430\u043e\u043f\u043a\u0443"
+ " \u0434\u0430\u0436\u0430\u043b \u0438 \u0440\u0430\u0437:0");
        }
    }
}

```



```
}
```

СТАЛО:

```
public class MyFirst_GUI {
    private JFrame frame;
    // label объявлена как приватное поле класса MyFirst_GUI
    private JLabel label;
    private int counter = 0;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {

        private void initialize() {

            // Сейчас label - поле класса и она доступна
            // в любом методе класса MyFirst_GUI
            label = new
JLabel("\u041A\u043D\u043E\u043F\u043A\u0443"
+ " \u043D\u0430\u0436\u0430\u043B\u0438 \u0440\u0430\u0437: 0");

        }
    }
}
```

Написание текста обработчика события для кнопки

Возвращаясь собственно к обработчику, напишем теперь код, который будет формировать строку «Кнопку нажали раз: [N]», где [N] - значение, равное количеству нажатий на кнопку. Вспомним, что мы уже добавили поле класса

```
private int counter = 0;
```

И что мы уже превратили в поле класса метку `private JLabel label;`

Таким образом, у нас есть уже все, чтобы написать нужный код. Ниже представлен окончательный вид кода обработчика нажатия на кнопку:

```
JButton button = new JButton("\u0416\u0430\u0442\u044C"
+ " \u0441\u044E\u0434\u0430!");
button.addActionListener(new ActionListener() {
    // обработчик события «нажатие»
    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);
    }
});
button.setBounds(45, 33, 350, 23);
frame.getContentPane().add(button);
```

Запускаем нашу программу. На экране появляется окно:

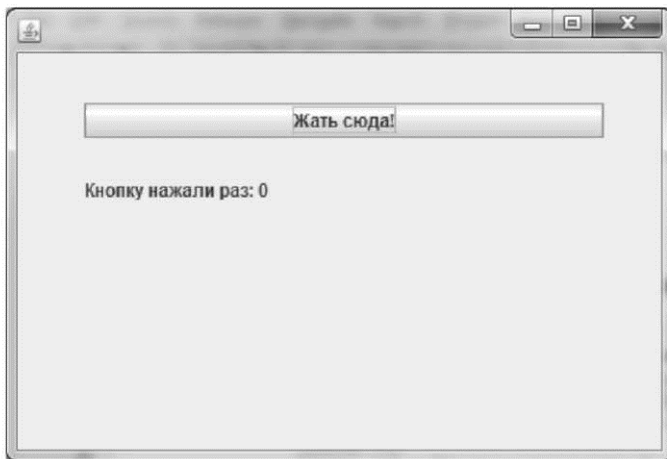


Рис. 1.28. Окно приложения сразу после запуска

Нажмем несколько раз на кнопку «Жать сюда!»:

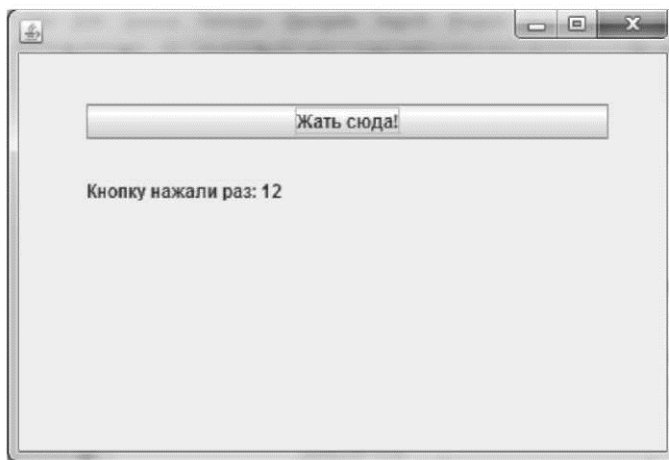


Рис. 1.29. Окно приложения после 12 нажатий кнопки

Если Вы писали код вместе с нами, поздравляем Вас с первым GUI приложением на Java!

Финальный код приложения, считающего нажатия кнопки

Ниже Вы найдете финальный код приложения, которое получилось у нас:

```
import java.awt.EventQueue;  
import javax.swing.JFrame;  
import javax.swing.JButton;  
import javax.swing.JLabel;
```

```

import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;

public class MyFirst_GUI {
    private JFrame frame;
    private JLabel label;
    private int counter = 0;
    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    MyFirst_GUI window = new MyFirst_GUI();
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    /**
     * Create the application.
     */
    public MyFirst_GUI() {
        initialize();
    }
    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frame = new JFrame();
        frame.setBounds(100, 100, 450, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);
        JButton button = new JButton("\u0416\u0430\u0442\u044c \u0441\u0447\u0438\u0442\u0430\u0442\u044c!");
        button.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                counter++;
                String str = "Кнопку нажали раз: " +counter;
                label.setText(str);
            }
        });
        button.setBounds(45, 33, 350, 23);
        frame.getContentPane().add(button);
        label = new JLabel("\u0416\u0430\u0442\u044c \u0441\u0447\u0438\u0442\u044c");
        label.setBounds(45, 83, 350, 14);
        frame.getContentPane().add(label);
    }
}

```

Создание GUI приложения с вводом, вычислением и выводом -на примере вычисления факториала числа

Напишем программу для вычисления факториала введенного числа. Напомним, что факториал натурального числа N вычисляется как произведение всех натуральных чисел от 1 до N включительно: $N! = 1 * 2 * \dots * (N - 1) * N$. Например, факториал числа 5 вычисляется так: $5! = 1 * 2 * 3 * 4 * 5 = 120$.

Мы хотим сделать приложение, в котором будет текстовое поле для ввода N , будет текстовое поле для вывода результата и будет кнопка, по нажатию на которую будет вычисляться и выводиться факториал. Интерфейс программы будет выглядеть так:

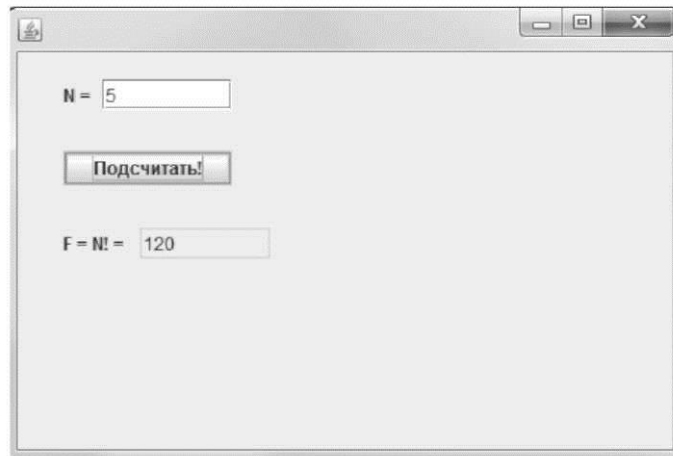


Рис. 1 .30. Работа программы после вычисления $5!$

Сначала вспомним, как реализуется само вычисление факториала - без GUI. Напишем программу, которая вычисляет значение факториала 5, используя цикл `for` и вывод в консоль:

```
public static void main(String[] args) {
    int n = 5;
    int f = 1;

    for (int i = 1; i <= n;)
    {
        f = f * i;
    }

    System.out.println("f= " + f);
}
```

Создадим новое окно приложения. Его можно создать прямо в нашем старом проекте - `GUI_Test`. Кликаем правой кнопкой мыши по проекту и выбираем `New => Other => Application Window`:

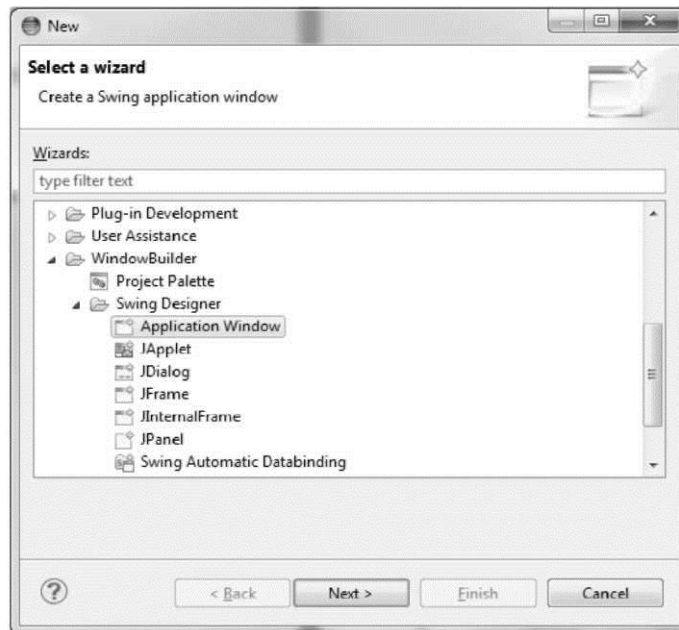


Рис. 1.31. Создание нового окна приложения

Теперь нужно придумать имя нашему приложению. Назовем его Factorial:

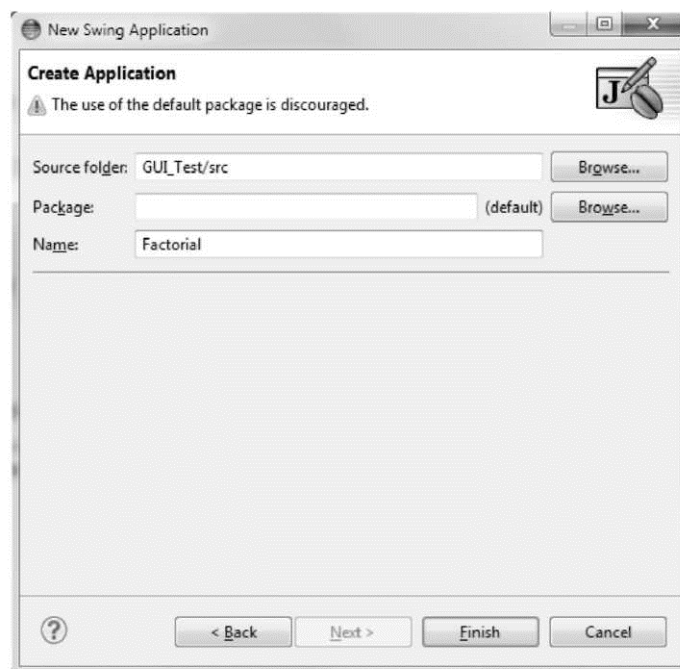


Рис. 1 .32. Имя приложения

После того как будет создан класс Factorial, нужно перейти в режим дизайнера форм и добавить на форму Absolute Layout:

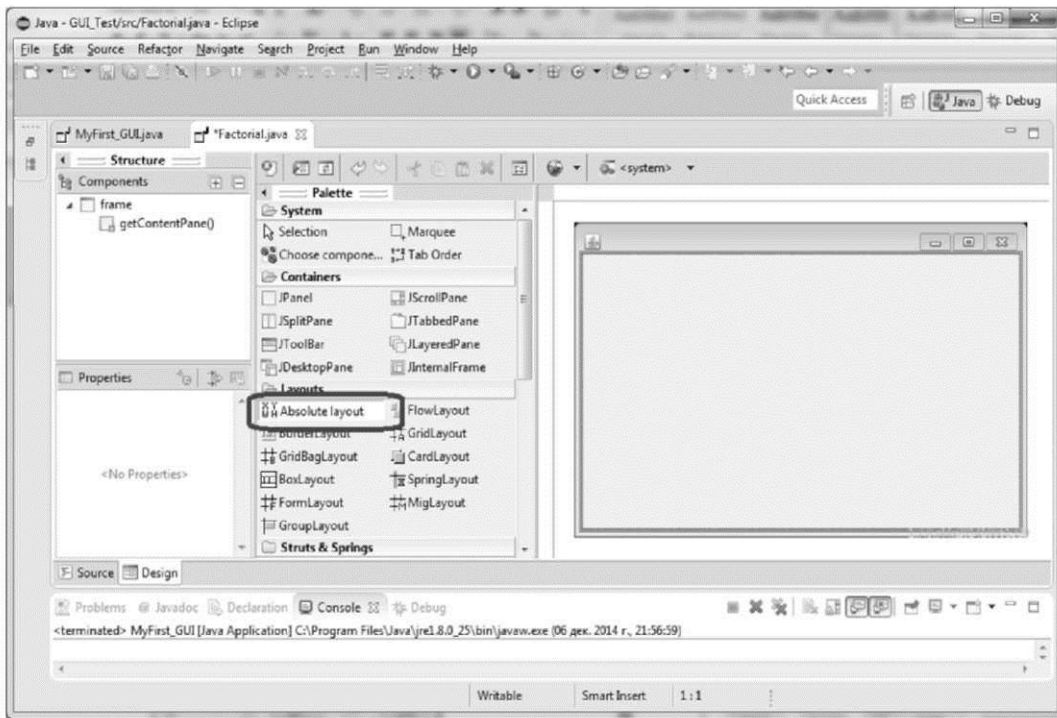


Рис. 1.33. Добавление Absolute Layout

Мы создали заготовку для нашего окна приложения, теперь наполним его элементами.

Добавление метки и изменение имени элемента

Добавляем метку с текстом «N⇒». И меняем название ее переменной на labelN. Для этого нужно выбрать нашу метку в списке компонентов (на рис. 2.34. - метка 1). Затем в Properties надо найти поле Variable (на рис. 2.34. - метка 2), и в поле Variable вписать «labelN» и нажать Enter:

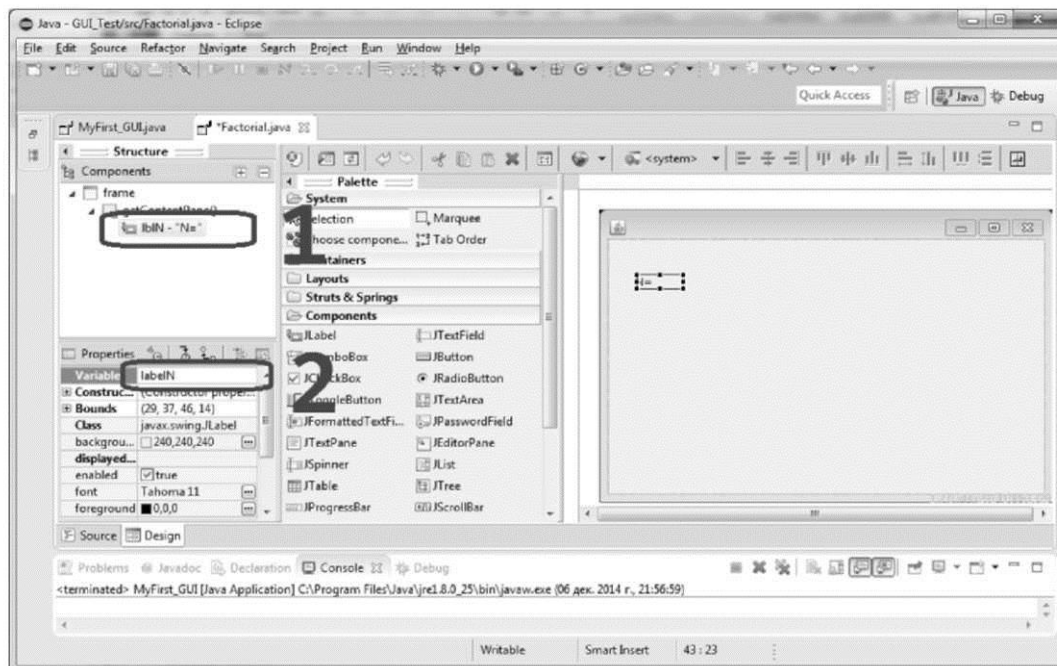


Рис. 1.34. Изменение имени переменной для метки

Аналогично добавляем метку с текстом «N! =>» и меняем имя ее переменной на labelFactorial. Рисунок ниже показывает состояние Eclipse после изменения имен переменных для обеих меток:

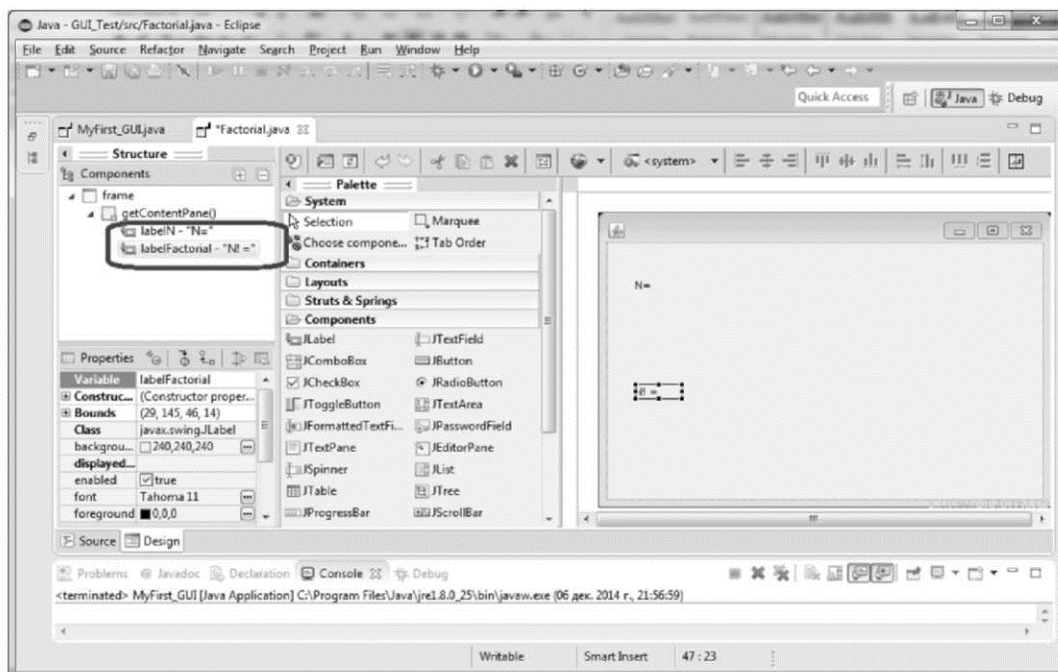


Рис. 1.35. После изменения имен переменных меток

Добавление кнопки и изменение текста уже созданного элемента

Добавим кнопку для подсчета факториала. И оставим пока текст кнопки «New button» по умолчанию. В итоге название переменной этой кнопки будет btnNewButton, как видно из следующего рисунка:

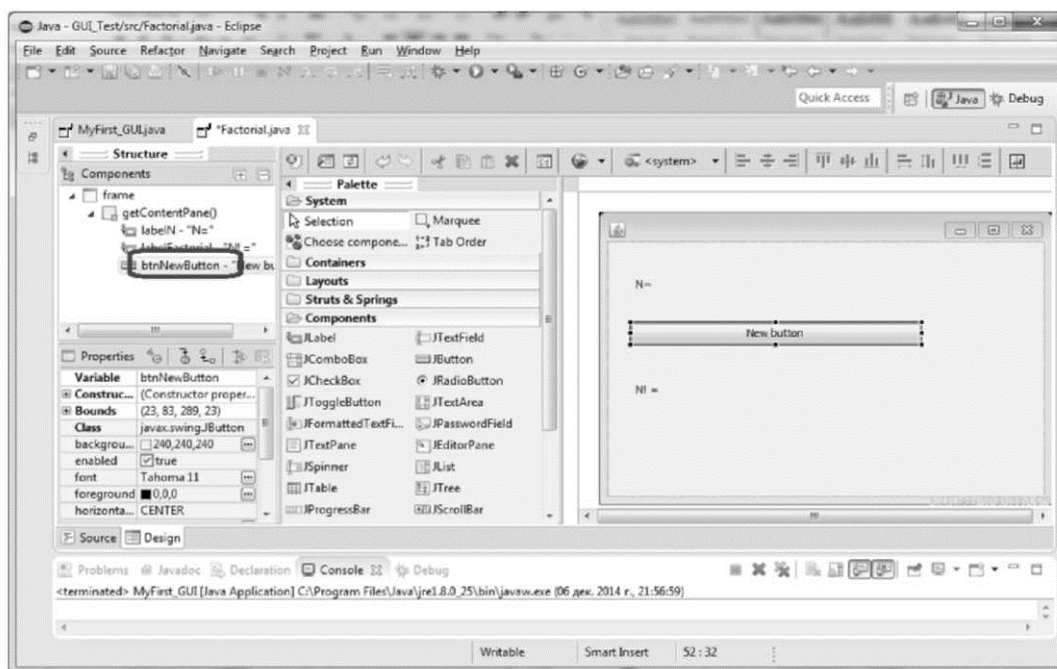


Рис. 1.36. Добавление кнопки

Теперь разберемся, как поменять текст уже созданной кнопки. Для этого нужно выбрать нашу кнопку в списке компонентов (на рис. 2.37 - метка 1). Затем в Properties надо найти поле text (на рис. 2.37 -метка 2), и в поле text вписать «Подсчитать!» и нажать Enter:

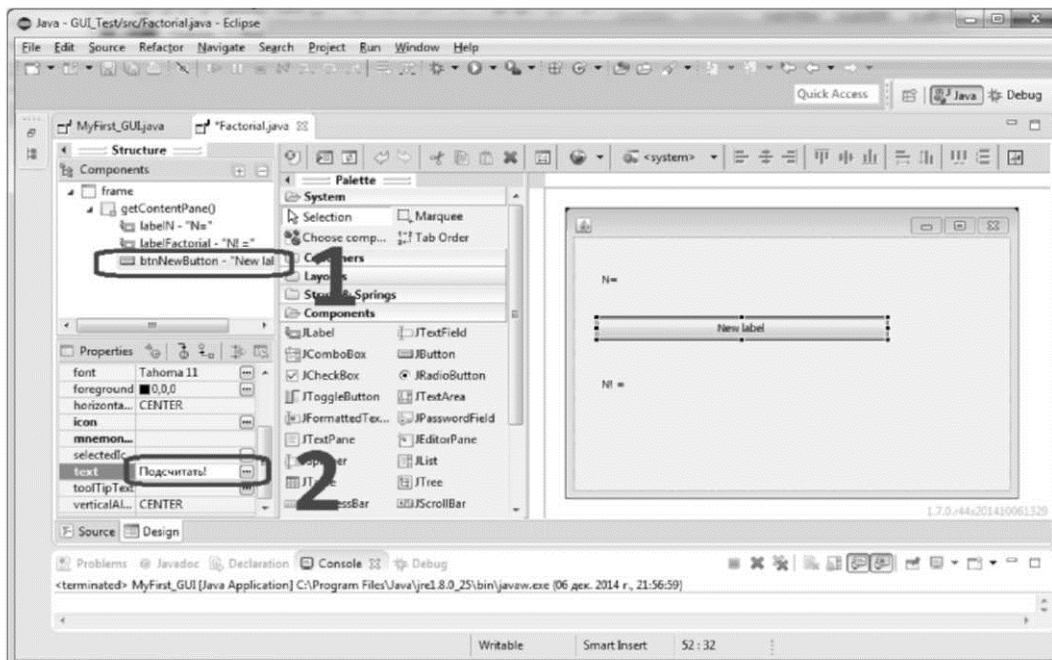


Рис. 1 .37. Изменение текста кнопки

Теперь нужно поменять имя переменной кнопки с труднопро-

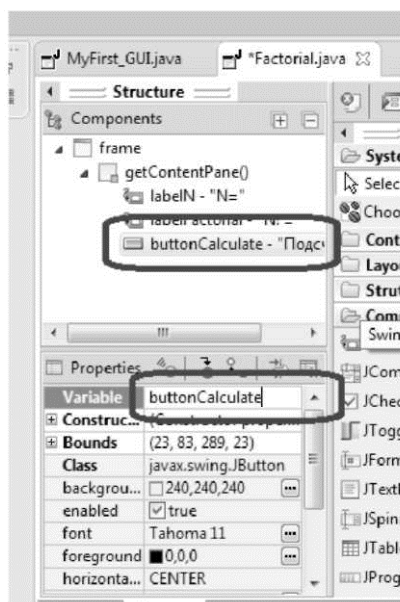


Рис. 1.38. Изменение названия имени переменной кнопки износимой btnNewButton на buttonCalculate:
Добавление текстового поля ввода

Добавляем текстовое поле ввода (JTextField) на наше окно:

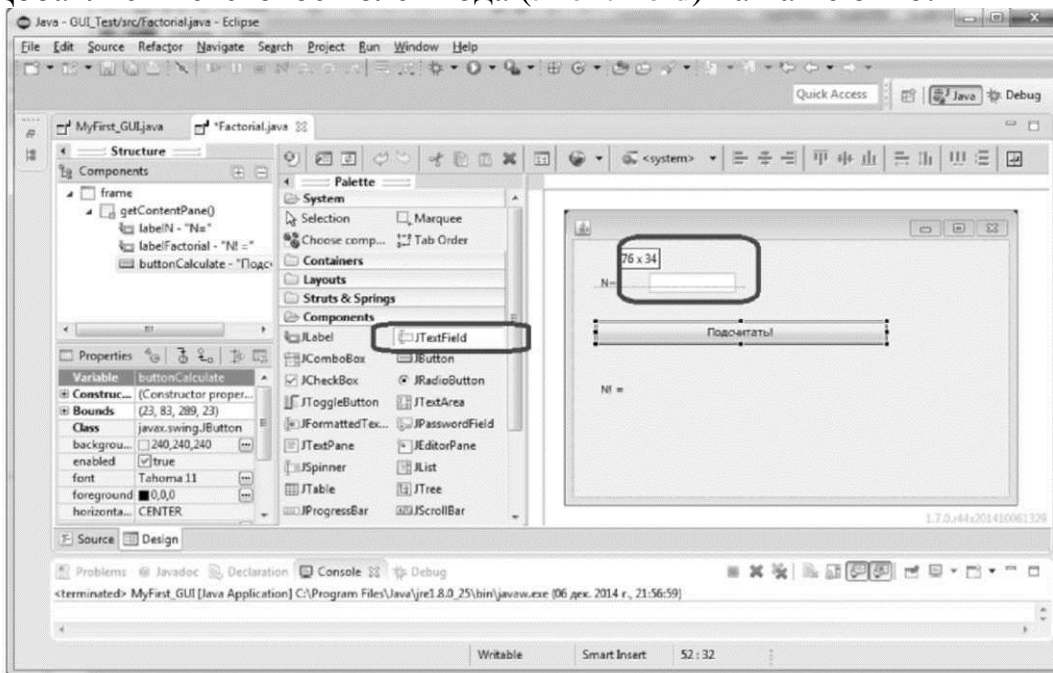


Рис. 1.39. Добавление JTextField

Меняем имя его переменной на textFieldN (на рис. 2.40 - метка 1) и изменяем его ширину, подгоняя ее под ширину кнопки (на рис. 2.40 - метка 2):

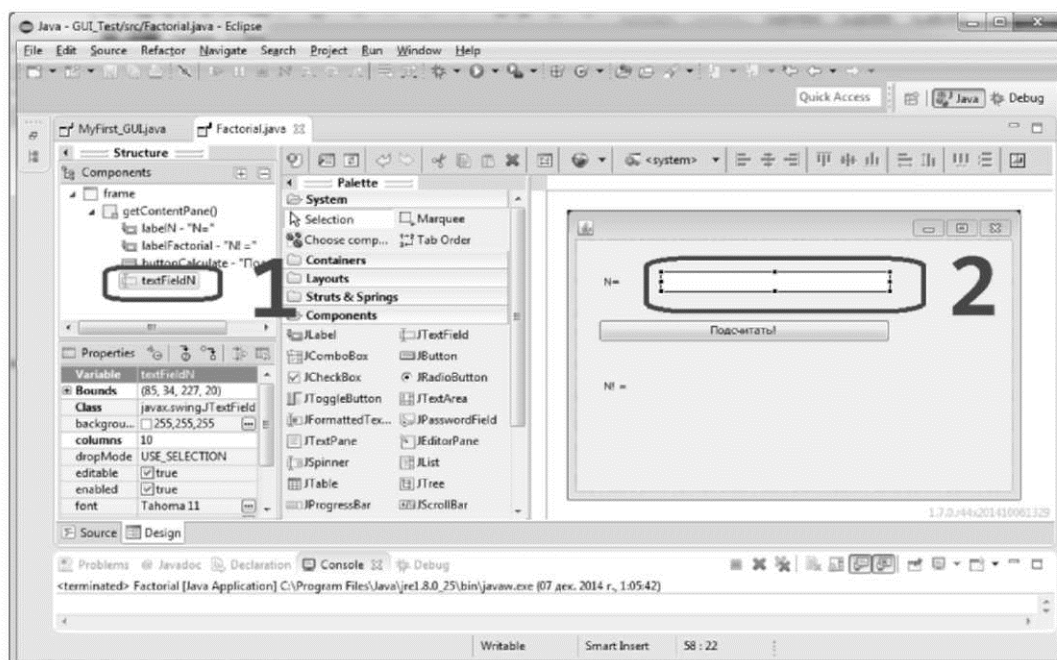


Рис. 1.40. Изменение параметров для JTextField

Добавление не редактируемого текстового поля для вывода

Аналогично предыдущему шагу добавляем еще одно текстовое поле для вывода значения факториала. Называем его переменную textFieldFactorial (на

рис. 2.41 - метка 1). И делаем это текстовое поле не редактируемым - в свойство `editable` убираем галочку (на рис. 2.41- метка 2).

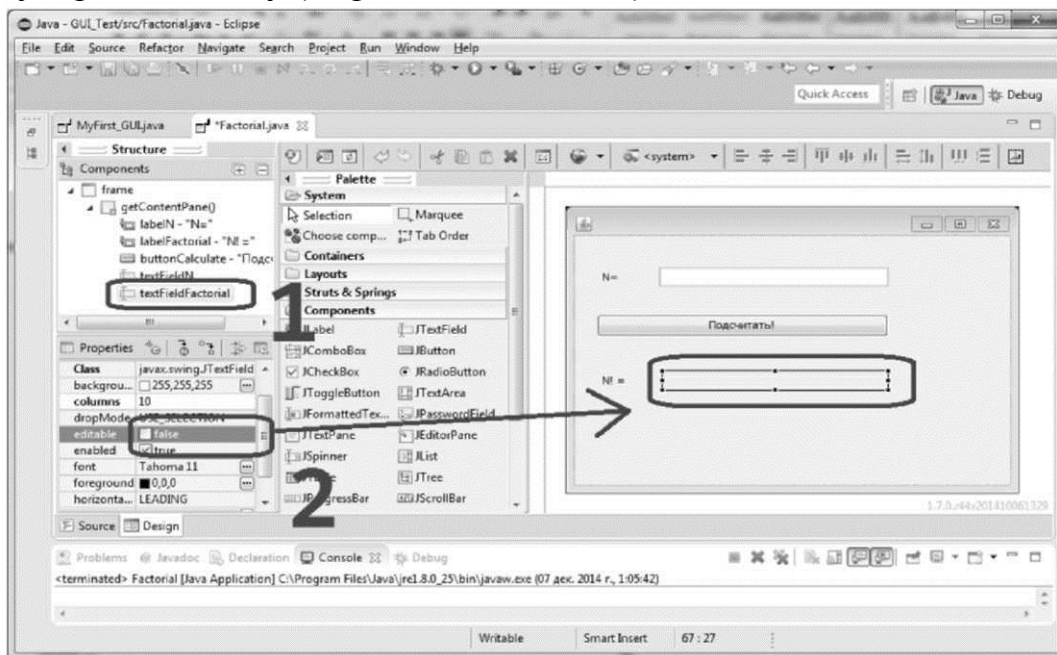


Рис. 1.41. Не редактируемое текстовое поле

Добавление обработчика нажатия кнопки

Дважды кликаем левой кнопкой на кнопку «Подсчитать», и у нас появляется заготовка для обработчика события нажатия кнопки:

```
private void initialize() {
    JButton buttonCalculate = new JButton(
        "\u041f\u043e\u0434\u0441\u0447\u0438\u0442\u0430\u0442\u044c!");
    buttonCalculate.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
        });
    buttonCalculate.setBounds(23, 83, 289, 23);
    frame.getContentPane().add(buttonCalculate);
}
```

Добавим в эту заготовку код, который вычисляет факториал F числа N , используя цикл `for`. Также добавим код, который получает число N из текстового поля `textFieldN` и выводит результат F в текстовое поле `textFieldFactorial`. Вот этот код:

```
private void initialize() {
    JButton buttonCalculate = new JButton(
        "\u041f\u043e\u0434\u0441\u0447\u0441\u0442\u0438\u0442\u0442\u0430\u0442\u044c!");
    buttonCalculate.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            int n;
            try {
```

```

        // берем значение n из текстового поля
        String strN = textFieldN.getText();
        // конвертируем значение из строки в число
        n = Integer.parseInt(strN);
    } catch (NumberFormatException ex) {
        // если ввод некорректен, выдаем текст
        // ошибки вместо значения факториала
        textFieldFactorial.setText( ex.getMessage());
        // и выходим из функции return;
    }
    // Вычисляем факториал int f =
    1;
    for (int i = 1; i <= n;)    {
        f = f * i;
    }
    // Выводим значение факториала
    String strF = "" + f;
    textFieldFactorial.setText(strF);
}
});
buttonCalculate.setBounds(23, 83, 289, 23);
frame.getContentPane().add(buttonCalculate);

```

Запускаем наше приложение. Вводим значение N, нажимаем кнопку «Подсчитать!», - результат выводится в поле рядом с меткой «N!=»:

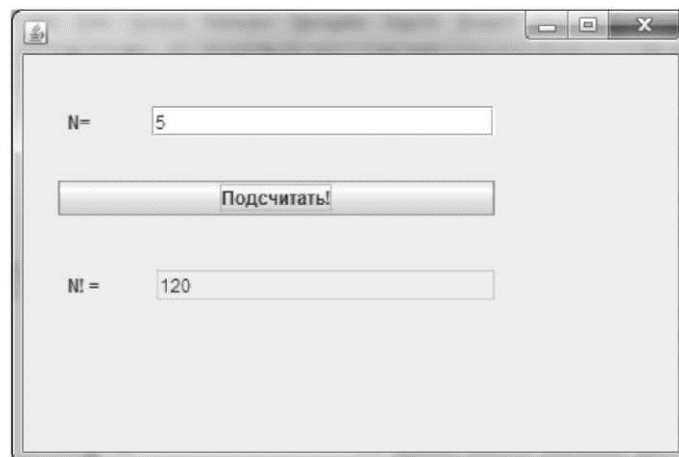


Рис. 1.42. Вычисление факториала работает корректно

Полный код приложения для вычисления факториала

```

import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JButton;
import javax.swing.JTextField;
import java.awt.event.ActionListener;

```

```

import java.awt.event.ActionEvent;

public class Factorial {

    private JFrame frame;
    private JTextField textFieldN;
    private JTextField textFieldFactorial;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Factorial window = new Factorial();
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public Factorial() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frame = new JFrame();
        frame.setBounds(100, 100, 450, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);

        JLabel labelN = new JLabel("N=");
        labelN.setBounds(29, 37, 46, 14);
        frame.getContentPane().add(labelN);

        JLabel labelFactorial = new JLabel("N! =");
        labelFactorial.setBounds(29, 145, 46, 14);
        frame.getContentPane().add(labelFactorial);

        JButton buttonCalculate = new JButton(
"\u041F\u043E\u0434\u0441\u0447\u0438\u0442\u0430\u0442\u044C!");
        buttonCalculate.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {

```

```

        int n;
        // Получаем значение N из поля ввода
        try {
            String strN = textFieldN.getText();
            n = Integer.parseInt(strN); }
        catch (NumberFormatException ex) {
            textFieldFactorial.setText( ex.getMessage());
            return;
        }
        // Вычисляем факториал int f =
        1;
        for (int i = 1; i <= n;)
        {
            f = f * i;
        }
        // Выводим значение факториала
        String strF = "" + f;
        textFieldFactorial.setText(strF);
    });
});

buttonCalculate.setBounds(23, 83, 289, 23);
frame.getContentPane().add(buttonCalculate);

textFieldN = new JTextField(); text-
FieldN.setBounds(85, 34, 227, 20);
frame.getContentPane().add(textFieldN); text-
FieldN.setColumns(10);

textFieldFactorial = new JTextField(); textFieldFactori-
al.setEditable(false); textFieldFactorial.setBounds(88, 142,
224, 20); frame.getContentPane().add(textFieldFactorial);
textFieldFactorial.setColumns(10);
}
}

```

Обратите внимание на то, что имена переменных и методов настолько говорящие, что требуется минимальное дополнительное комментирование кода. Такой код, написанный с понятными именами и с прозрачной логикой, называется самодокументированным.

Пример выполнения работы D - вывод узора из чисел

Создадим программу для вывода чисел в виде узора. Для числа 5 узор должен выглядеть так:

```

1
12
123
1234
12345

```

Программа должна содержать GUI с двумя элементами - поле для ввода числа и поле для вывода узора. При изменении значения в поле ввода узор должен перерисовываться автоматически.

Разработаем такую программу. Внешний вид программы будет таким:

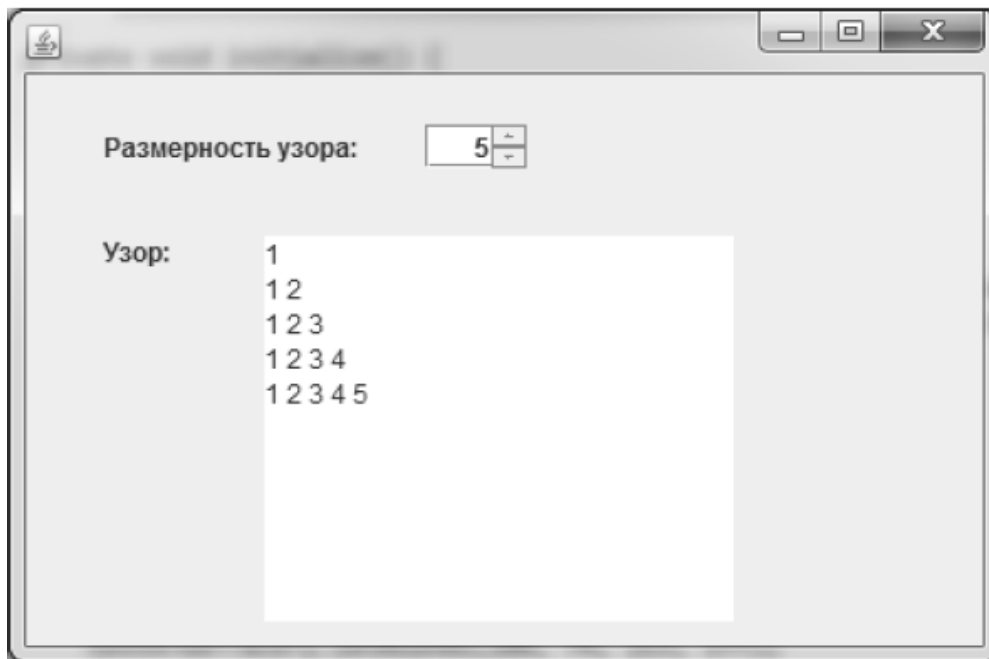


Рис. 1.43. Рисунок из чисел

Сначала создадим простую программу с выводом в консоль узора (для числа 5):

```
public class TestTracery {  
  
    public static void main(String[] args) {  
        int size = 5;  
        for (int i = 1; i <= size; i++) {  
            for (int j = 1; j <= i; j++) {  
                System.out.print(j + " ");  
            }  
            System.out.println();  
        }  
    }  
}
```

Создадим для нашей программы новое окно приложения. Создадим его в нашем проекте «GUI_Test» и назовем «Tracery». Добавим на форму Absolute Layout. Добавим метку с текстом «Размерность узора:» и метку с текстом «Узор:».

Добавление элемента JTextArea

Затем добавим компонент JTextArea и назовем его переменной textArea-Tracery:

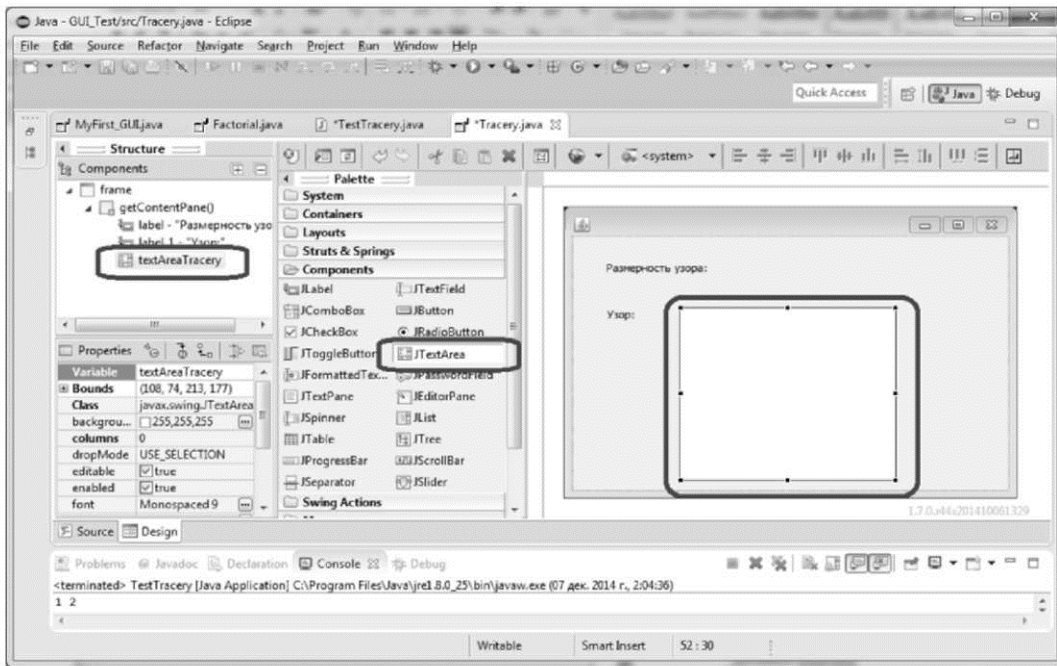


Рис. 1.44. Добавление JTextArea

Добавление элемента JSpinner и установка параметров его модели
 Добавим компонент JSpinner и назовем его переменной spinnerSize:

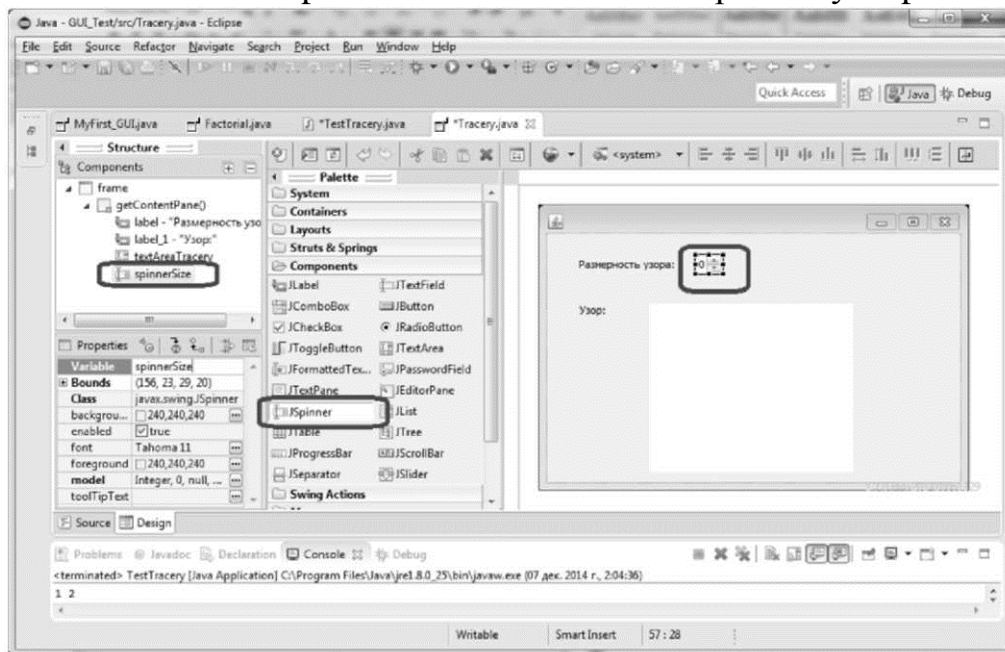


Рис. 1.45. Добавление JSpinner

Выберем у JSpinner свойство model и зададим в ней следующие параметры - Number type = Integer, Initial value = 0, Minimum = 0, Maximum = 9, Step size = 1:

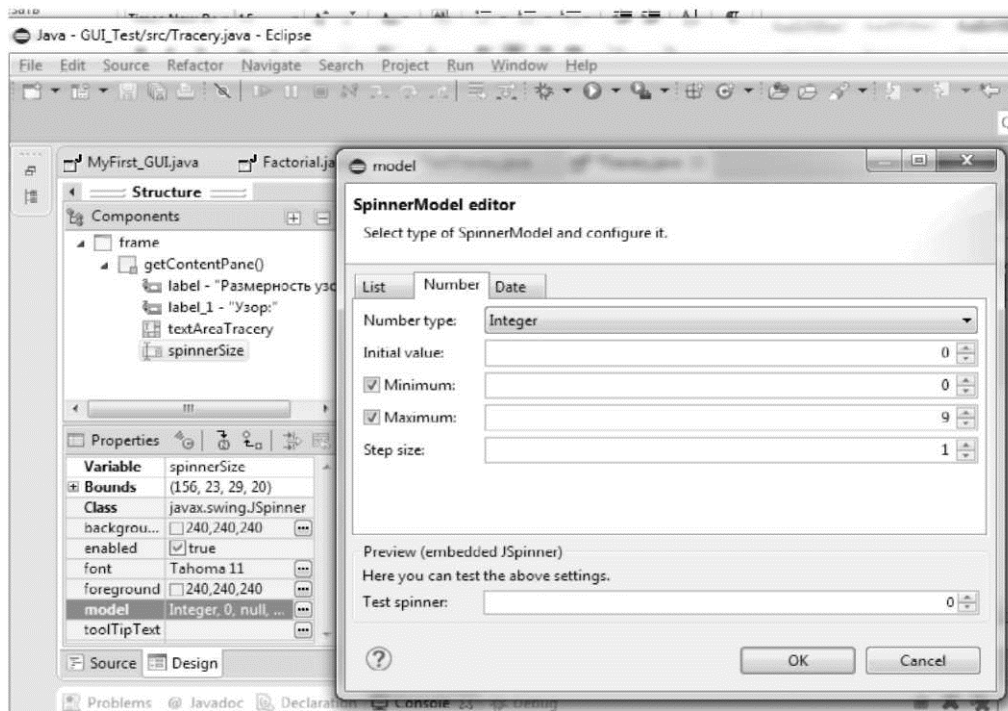


Рис. 1.46. Изменение параметров модели для JSpinner

Запустим приложение. Попробуем изменить значение «Размерность узора». Все работает!

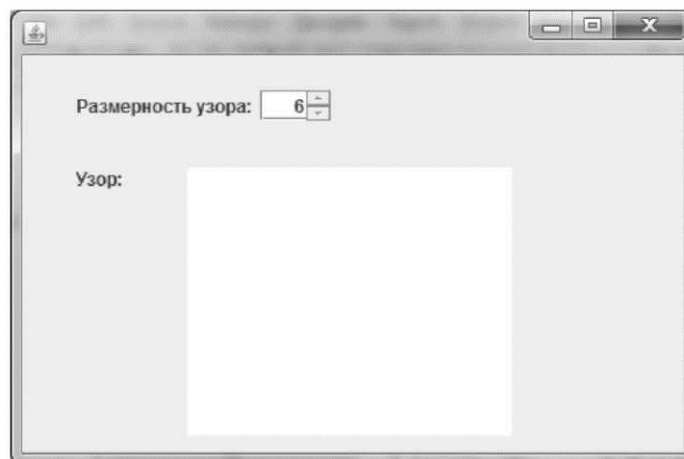


Рис. 1.47. Работающее приложение (без начинки - только GUI)

Код приложения с JTextArea и JSpinner

```
import java.awt.EventQueue;  
import javax.swing.JFrame;  
import javax.swing.JLabel;  
import javax.swing.JTextArea;
```



```

import javax.swing.JSpinner;
import javax.swing.SpinnerNumberModel;

public class Tracery {

    private JFrame frame;

    /**
     * Launch the application. */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Tracery window = new Tracery();
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application. */
    public Tracery() {
        initialize();
    }

    /**
     * Initialize the contents of the frame. */
    private void initialize() {
        frame = new JFrame();
        frame.setBounds(100, 100, 450, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);

        JLabel label = new JLabel(
            "\u0420\u0430\u0437\u043c\u0435\u0440\u043d\u043e\u0441\u0441\u0442\u044c"
            + "\u044c \u0443\u0437\u043e\u0440\u0430\u0430\u0430\u0430");
        label.setBounds(35, 26, 121, 14);
        frame.getContentPane().add(label);

        JLabel label_1 = new JLabel("\u0420\u0437\u043e\u0440\u0440\u0440\u0440");
        label_1.setBounds(35, 74, 46, 14);
        frame.getContentPane().add(label_1);

        // textAreaTracery - это локальная переменная
        JTextArea textAreaTracery = new JTextArea();
        textAreaTracery.setBounds(108, 74, 213, 177);
        frame.getContentPane().add(textAreaTracery);

        // spinnerSize - это локальная переменная
        JSpinner spinnerSize = new JSpinner();
    }
}

```

```
spinnerSize.setModel(new SpinnerNumberModel(0, 0, 9,1));
spinnerSize.setBounds(181, 23, 46, 20);
frame.getContentPane().add(spinnerSize);
```

Конвертация локальных переменных в поля класса

Обратите внимание на выделенные жирным строки в коде выше, в них сказано, что переменные `textAreaTracery` и `spinnerSize` -это локальные переменные. Нам для написания обработчика для `JSpinner` нужно превратить их в поля класса. Для этого мы вынесем их определение в тело класса и изменим код метода `initialize()` как показано ниже:

```
public class Tracery {

    private JFrame frame;
    JSpinner spinnerSize;
    JTextArea textAreaTracery;

    private void initialize() {

        // textAreaTracery - это поле класса Tracery
        textAreaTracery = new JTextArea();
        textAreaTracery.setBounds(108, 74, 213, 177);
        frame.getContentPane().add(textAreaTracery);

        // spinnerSize - это поле класса Tracery
        spinnerSize = new JSpinner();
        spinnerSize.setModel(new SpinnerNumberModel(0, 0, 9,1));
        spinnerSize.setBounds(181, 23, 46, 20);
        frame.getContentPane().add(spinnerSize);
    }
}
```

Добавление обработчика событий изменения состояния JSpinner

Добавим обработчик события изменения состояния `JSpinner`. Для этого в текстовом редакторе в код `initialize()` вставим код, который помечен жирным:

```
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

    private void initialize() {
        // spinnerSize - это поле класса Tracery
        spinnerSize = new JSpinner();

        // к spinnerSize добавляем слушатель событий
        // изменения значения JSpinner
        spinnerSize.addChangeListener(new ChangeListener() {
        // собственно этот метод будет вызываться когда
        // состояние JSpinner измениться
    }
}
```

```

public void stateChanged(ChangeEvent e) {
    // значение из spinnerSize помещаем в size
    int size = (Integer)spinnerSize.getValue();

    // В строке str будем формировать узор
    String str = "";
    // В цикле формируем узор
    for (int i = 1; i <= size; i++) {
        for (int j = 1; j <= i;) {
            str = str + j + " ";
        }
        // "\n" - это следующая строка
        str = str + "\n";
    }
    // Сформированный узор помещаем в
    // textAreaTracery
    textAreaTracery.setText(str);
}
});
spinnerSize.setModel(new SpinnerNumberModel(0, 0, 9,1));
spinnerSize.setBounds(181, 23, 46, 20);
frame.getContentPane().add(spinnerSize);
}

```

Проверка работы приложения с узорами

Запустим на выполнение нашу программу. Попробуем поменять значение в JSpinner. Мы увидим, что узор меняется сразу же после изменения JSpinner:

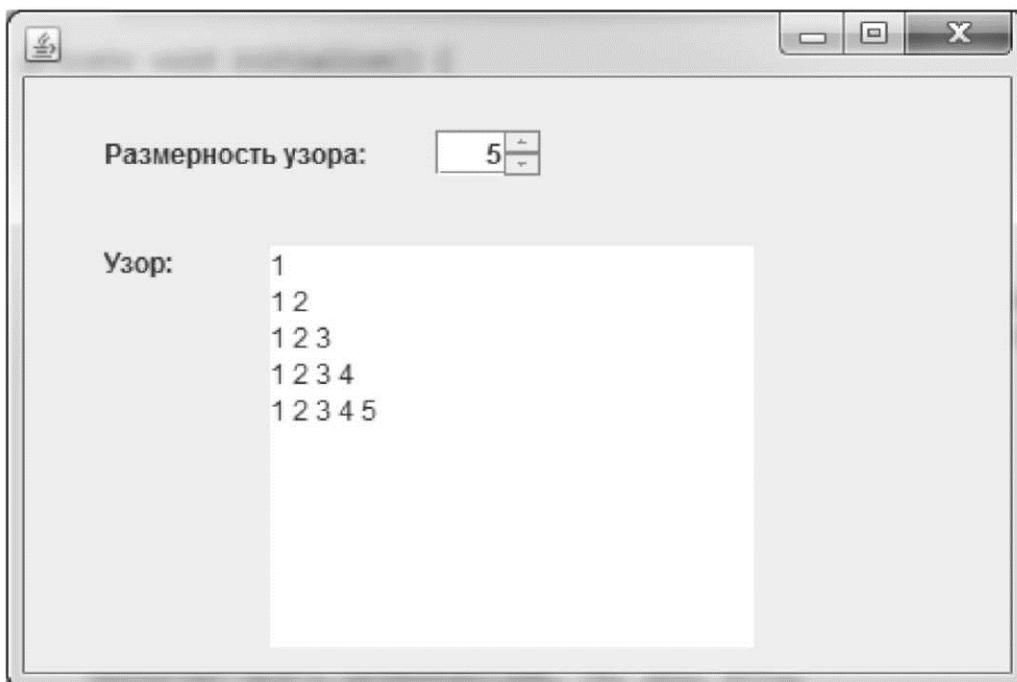


Рис. 1.48. Работающее полностью готовое приложение

Полный код приложения рисования узора

```
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextArea;
import javax.swing.JSpinner;
import javax.swing.SpinnerNumberModel;
import javax.swing.event.ChangeEvent;
import javax.swing.event.ChangeListener;

public class Tracery {
    private JFrame frame;
    JSpinner spinnerSize;
    JTextArea textAreaTracery;
    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {

                    Tracery window = new Tracery();
                    window.frame.setVisible(true); }
                catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
    /**
     * Create the application.
     */
    public Tracery() { initialize();
    }
    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frame = new JFrame();
        frame.setBounds(100, 100, 450, 300);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);

        JLabel label = new JLabel(
            "\u0420\u0430\u0437\u043C\u0435\u0440\u043D\u043E\u043E\u0441\u0442\u0442\u044C"
            + "\u044C \u0443\u0437\u043E\u0440\u0440\u0430:");
    }
}
```

```

label.setBounds(35, 26, 121, 14);
frame.getContentPane().add(label);

JLabel label_1 = new JLabel("\u0423\u0437\u043E\u0440:");
label_1.setBounds(35, 74, 46, 14);
frame.getContentPane().add(label_1);

// textAreaTracery - это поле класса Tracery
textAreaTracery = new JTextArea();
textAreaTracery.setBounds(108, 74, 213, 177);
frame.getContentPane().add(textAreaTracery);

// spinnerSize - это поле класса Tracery
spinnerSize = new JSpinner();
// При каждом изменении значения JSpinner
// перерисовываем узор
spinnerSize.addChangeListener(new ChangeListener() {
public void stateChanged(ChangeEvent e) {
    int size = (Integer)spinnerSize.getValue();
    String str = "";

    for (int i = 1; i <= size; i++) {
        for (int j = 1; j <= i; j++) {
            str = str + j + " ";
        }
        str = str + "\n";
    }
    textAreaTracery.setText(str);
}
});
spinnerSize.setModel(new SpinnerNumberModel(0, 0, 9,1));
spinnerSize.setBounds(181, 23, 46, 20);
frame.getContentPane().add(spinnerSize);
}
}

```

СОЗДАНИЕ ИЗОБРАЖЕНИЙ СРЕДСТВАМИ JAVA

Программы очень часто выводят результат своей работы в виде графических образов - графиков, рисунков или чего-то другого. Например, программа «симулятор полета на самолете» выводит в реальном времени изображение, видимое из кабины пилота. Программа, встроенная в аппарат УЗИ, выводит на экран изображение на основе данных с датчиков и т.д.

Естественно, во всех языках высокого уровня есть встроенные средства для построения графических образов. В Java в библиотеке AWT используются классы на основе абстрактного класса Graphics. Если Вы используете библиотеку AWT или Swing (в том числе и через WindowBuilder), Вы можете создавать графические образы при помощи средств Graphics. Ниже Вы познакомитесь с простейшими средствами Graphics, которых достаточно для создания простых векторных изображений. И Вы потренируетесь в создании как статичных, так и генерируемых изображений.

Экранная система координат

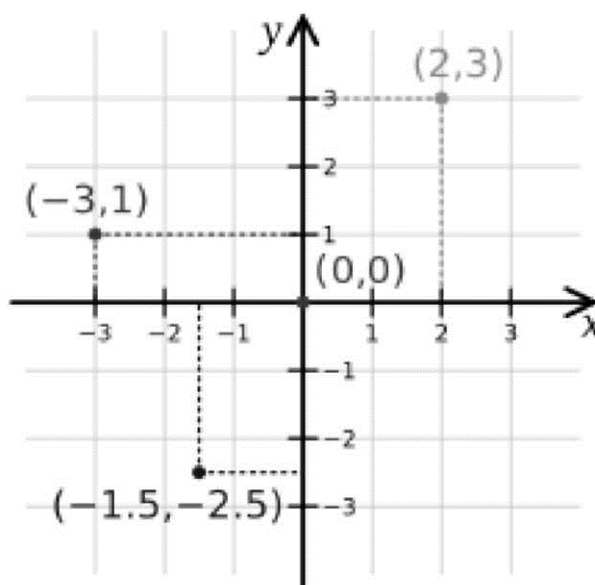


Рис. 2.1. Точки в Декартовой системе координат

Все мы знакомы с Декартовой системой координат. В ней начало координат - точка (0,0) - находится в центре, ось X направлена вправо, ось Y вверх (рис. 2.1).

Для создания изображений на экране дисплея используют экранную систему координат. Она отличается от Декартовой тем, что начало координат - точка (0, 0) - находится в верхнем левом углу экрана, ось X направлена вправо, так же как и в Декартовой системе координат, а ось Y направлена вниз. Стоит отметить еще и тот факт, что будут видны только те элементы изображения, ко-

торые имеют координаты в пределах от 0 до `WindowWidth-1` - для оси X, и от 0 до `WindowHeight-1` - для оси Y. Рисунок ниже демонстрирует это.

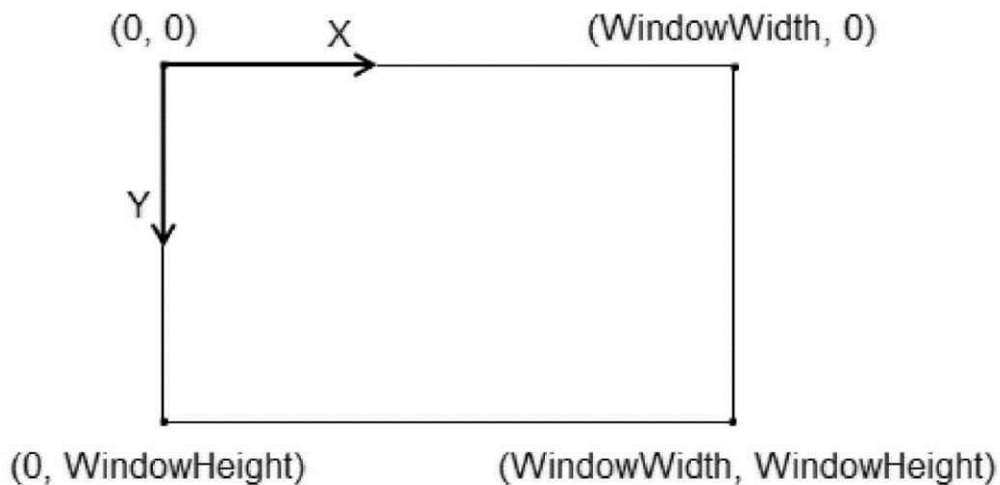


Рис. 2.2. Экранная система координат

Графические примитивы Graphics

Как было сказано выше, мы познакомимся со средствами `Graphics` для создания графических образов. Посмотрим, что можно создать при их помощи. Рисунок ниже демонстрирует рисование домика при помощи 2 линий и 1 прямоугольника:

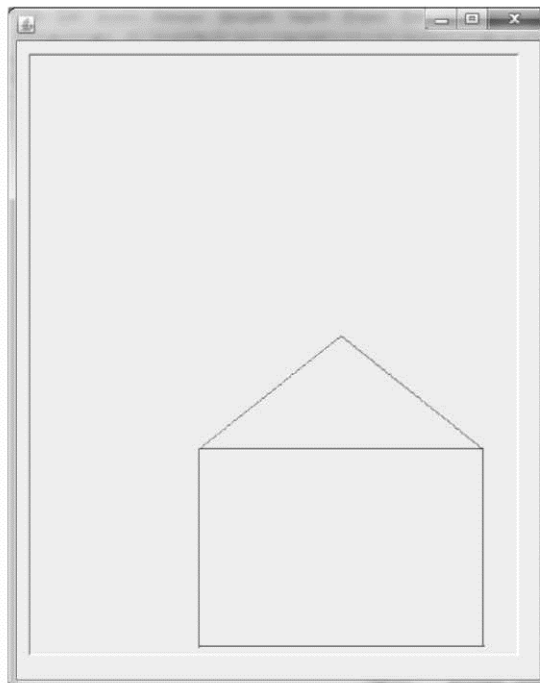


Рис. 2.3. Простейший рисунок: домик из трех элементов

Этот рисунок создается в методе `paint()`, который приведен в коде ниже:

```

// В метод paint() передается объект g, реализующий возможности
// класса Graphics.
// Чтобы создать свое изображение, мы используем разные методы
// объекта g.
public void paint(Graphics g) {
    // пурпурный цвет делается текущим - все ниже будет
    // рисоваться пурпурным цветом
    g.setColor(Color.magenta);
    // рисуем корпус домика
    g.drawRect(150, 350, 250, 175);
    // левая линия крыши
    g.drawLine(150,350,275,250);
    // правая линия крыши
    g.drawLine(275,250,400,350);
}

```

Рассмотрим по отдельности основные графические примитивы Graphics.

Graphics.setColor(Color color)

Устанавливает цвет для всех графических элементов, которые отображаются в коде ниже вызова setColor().

Color - это класс, который описывает цвет. Чаще всего цвет задают либо константами: Color.magenta, Color.red, Color.white и др., либо создают нужный цвет через конструктор Color(int Red, int Green, int Blue), где Red, Green, Blue - числа из диапазона 0..255 - составные RGB цвета из палитры цветов. Например, Color(0, 0, 255) - это синий, а Color(255, 255, 0) - это желтый цвет - смотри рисунок ниже.

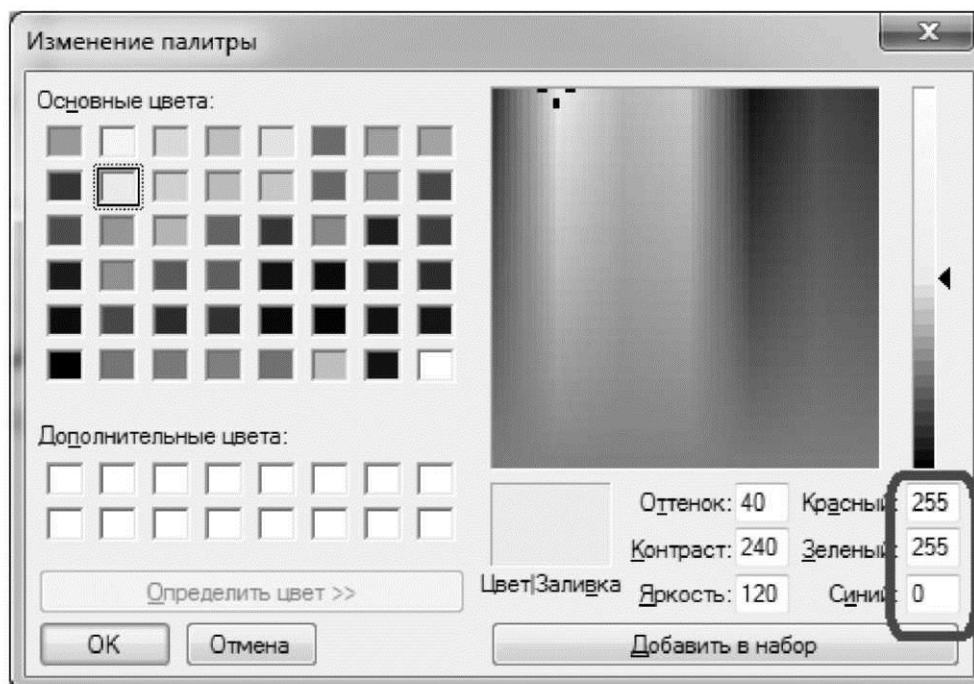


Рис. 2.4. Палитра цветов и RGB

Пример использования setColor():

```

// Текущий цвет синий - все ниже рисуется синим

```



```

g.setColor(Color.blue);
// линия рисуется синим цветом
g.drawLine(10, 50, 100, 50);
// Текущий цвет желтый - все ниже рисуется желтым
g.setColor(new Color(255, 255, 0));
// линия рисуется желтым цветом
g.drawLine(10, 55, 100, 55);
// линия рисуется желтым цветом
g.drawLine(10, 60, 100, 60);
// линия рисуется желтым цветом
g.drawLine(10, 65, 100, 65);

```

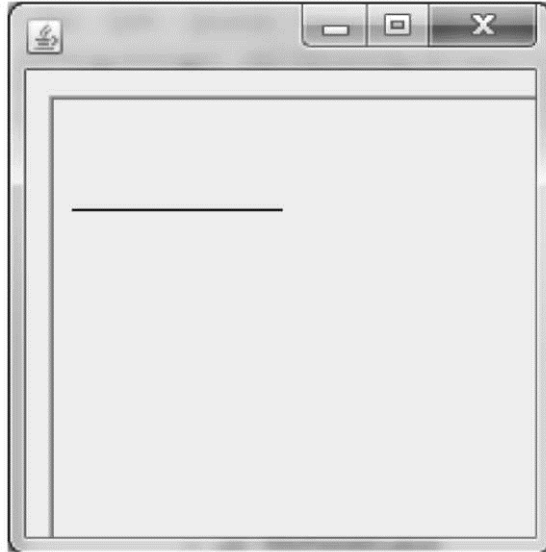


Рис. 2.5. Пример использования setColor() и drawLine()

Graphics.drawLine(int x1, int y1, int x2, int y2)

Рисует линию между точками (x1,y1) и (x2,y2).

Заметим, что, как и везде в Graphics, координаты точек задаются в экран-ных координатах. Цвет, используемый при отрисовке, берется текущий - уста-новленный предыдущим вызовом setColor(). Пример использования drawLine() можно увидеть в предыдущем примере кода с setColor()

Graphics.drawRect(int x, int y, int width, int height)

Рисует контур прямоугольника.

x, y - координаты верхнего левого угла прямоугольника, width - длина прямоугольника, height - высота прямоугольника.

Пример использования drawRect():

```

g.setColor(Color.RED);
g.drawRect(10, 50, 100, 50);

```

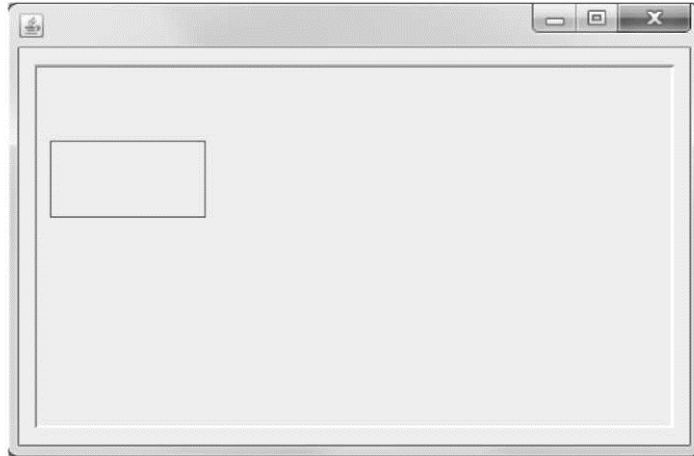


Рис. 2.6. Пример использования drawRect()

Graphics.fillRect(int x,int y, int width, int height)

Пример использования fillRect():

```
g.setColor(Color.RED);  
g.fillRect(10, 50, 100, 50);
```

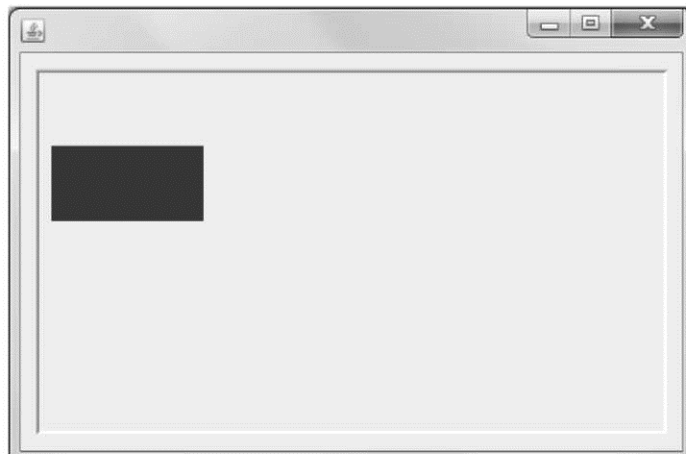


Рис. 2.7. Пример использования fillRect()

Заливает прямоугольник текущим цветом. Параметры -аналогичны drawRect()

Graphics.drawRoundRect(int x, int y, int width, int height, int rx, int ry)

Рисует контур заданного прямоугольника с закругленными углами.

Первые 4 аргумента как у обычного прямоугольника: x,y -координаты верхнего левого угла прямоугольника, width - длина, height - высота. Аргумент rx — это ширина прямоугольника, в который вписана часть овала сопряжения. Аргумент ry — это высота прямоугольника, в который вписана часть овала сопряжения.

Пример использования `drawRoundRect()`:

```
g.setColor(Color.RED);  
g.drawRoundRect(10, 50, 100, 50, 10, 30);
```

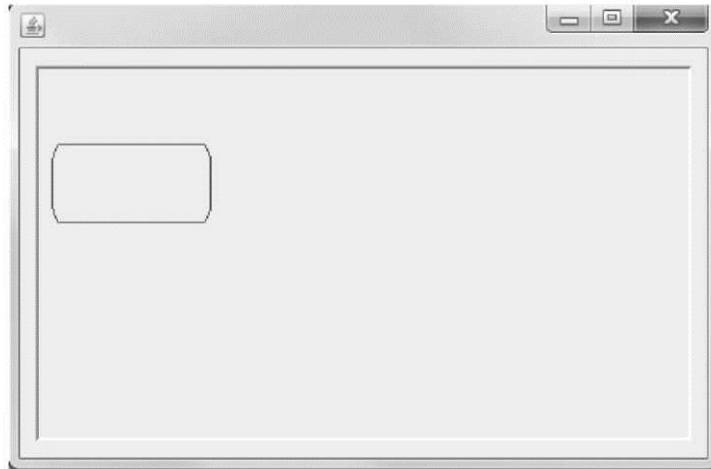


Рис. 2.8. Пример использования `drawRoundRect()`

Graphics.drawOval(int x, int y, int width, int height)

Рисует контур овала. Аргументы `x`, `y`, `width`, `height` определяют прямоугольник, в который будет вписан овал. Пример использования `drawOval()`:

```
g.setColor(Color.RED);  
g.drawOval(10, 50, 100, 50);
```

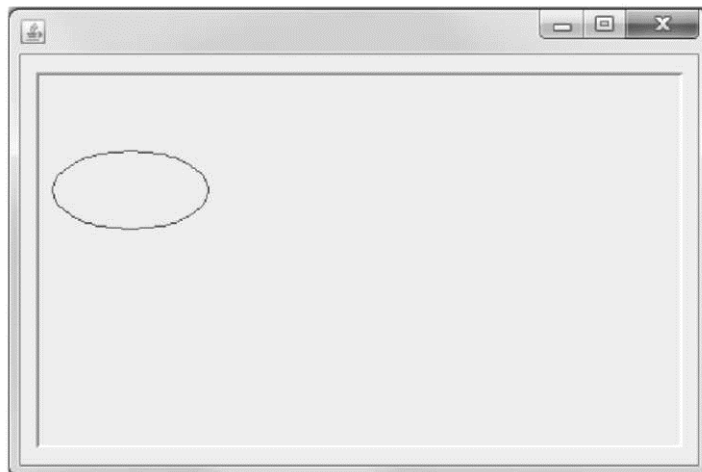


Рис. 2.9. Пример использования `drawOval()`

Graphics.drawArc(int x, int y, int width, int height, int alpha, int beta)

Рисует контур овальной дуги, вписанной в прямоугольник. Первые 4 аргумента как у обычного прямоугольника. Пятый аргумент — `alpha` — это угол, от которого отсчитывается угол самой дуги. Аргумент `beta` — это длина дуги в углах.

Пример использования `drawArc`:

```
g.setColor(Color.RED);  
g.drawArc(10, 50, 100, 50, 180, 180);
```

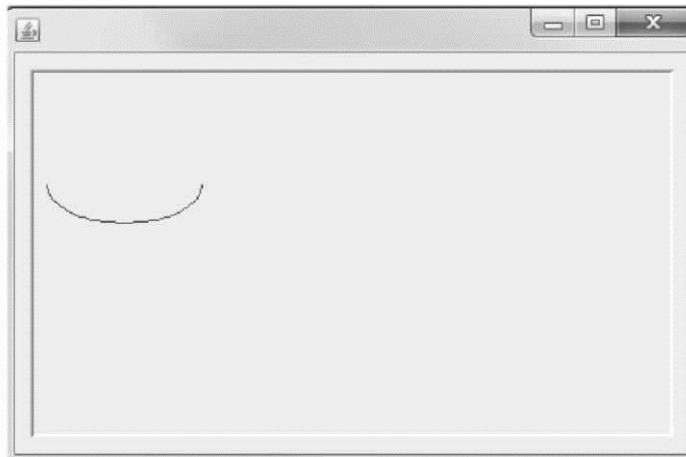


Рис. 2.10. Пример использования drawArc()

Заметим, что направление, от которого идет отсчет угла, - это направление на 3 часа (строго вправо) - это 0 градусов. Угол увеличивается при вращении против часовой стрелки. То есть направление на 12 часов (строго вверх) - это 90 градусов, направление на 9 часов (строго влево) - это 180 градусов и т. д.

Graphics.drawString(String string, int x, int y)

Выводит строку с текстом, используя текущий шрифт и цвет. Пример использования drawString():

```
g.setFont(new Font("Courier", Font.BOLD + Font.ITALIC,16));  
g.setColor(Color.RED); g.drawString("Hello World!", 100, 50);
```



Рис. 2.11. Выводим строку

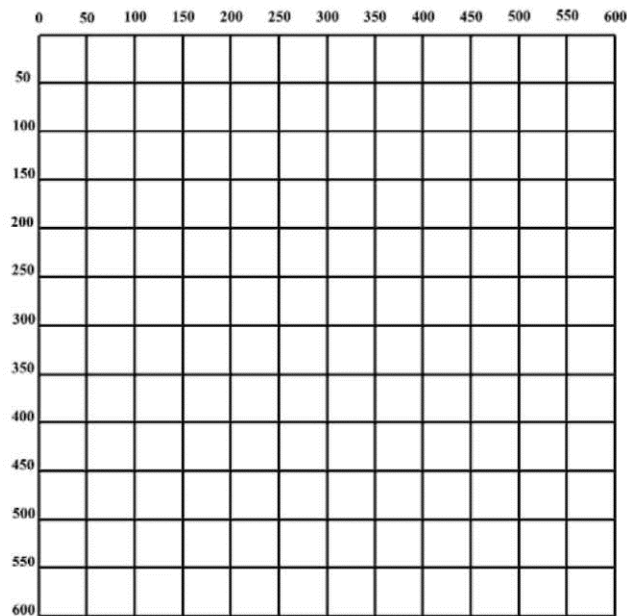
Строка «Hello World!» будет выведена от точки с координатами 100, 50.

Рисование в собственной панели с использованием Graphics

Вы познакомились с некоторыми средствами Graphics для создания графических изображений. Теперь давайте научимся использовать эти средства и что-нибудь нарисуем в окне приложения. Нарисуем, например, домик такого вида:

Координатная сетка для рисования

Чтобы нарисовать домик, нам потребуются координаты каждой из точек начала и конца линий. Чтобы точно указать координаты, нам требуется нарисовать сначала домик в координатной сетке. Координатная сетка для рисования изображения выглядит так:



Изображение в координатной сетке

Нарисуем домик в этой координатной сетке:

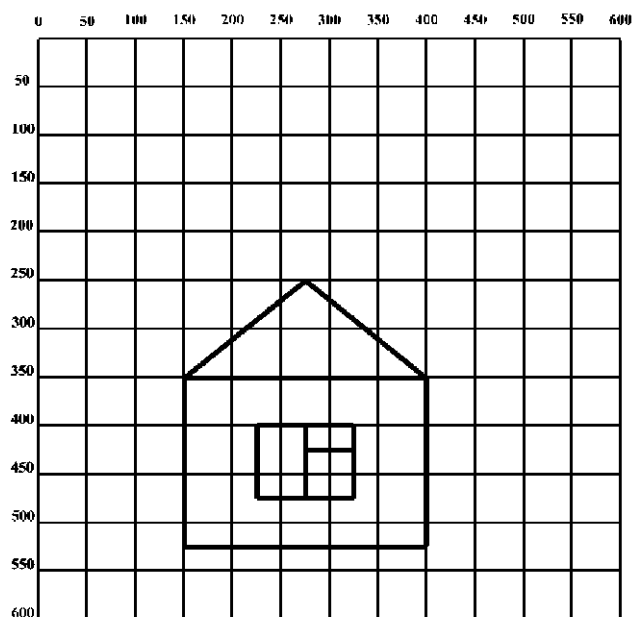


Рис. 2.14. Рисунок в координатной сетке

Оцифровка изображения в координатной сетке

Чтобы мы могли в коде прописать значения параметров для линий и прямоугольников, нам нужно точно знать значения координат.

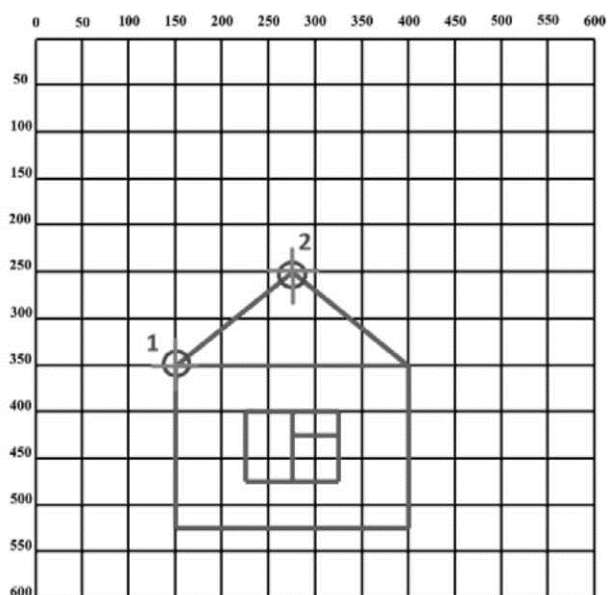


Рис. 2.15. Оцифровка точек в координатной сетке

Оцифруем координаты точек 1 и 2 на рис. 2.15:

Точка 1: (X=150; Y=350)

Точка 2: (x=275; Y=250)

Код, рисующий линию между точками 1 и 2, выглядит так:
g.drawLine(150,350,275,250);

Код метода paint() для отрисовки домика

Как уже говорилось выше, отрисовка изображений делается в методе paint(). На основе рисунка домика в координатной сетке подготовим заготовку кода метода paint():

```
// Заготовка метода paint()
public void paint(Graphics g) {

    // выберем пурпурный цвет
    g.setColor(Color.magenta);
    // рисуем корпус домика
    g.drawRect(150, 350, 250, 175);
    // рисуем внешний контур окна домика
    g.drawRect(225, 400, 100, 75);
}
```

Создание приложения с собственной панелью

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.

Создание собственной панели - класс MyPanel

Один из самых простых визуальных компонентов в Swing - это панель JPanel. Поэтому мы создадим наш визуальный компонент с домиком как расширение (наследник) класса JPanel.

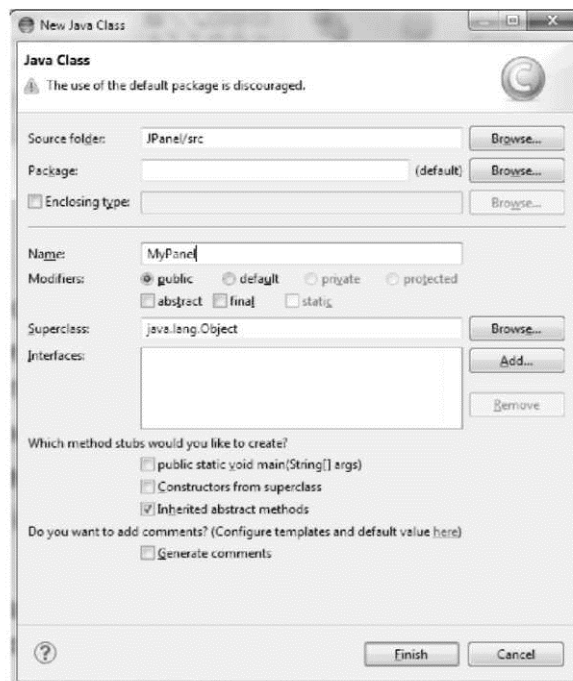


Рис. 2.16. Создание Java класса

Для этого в Eclipse создадим новый класс (обычный Java класс, не Application Window). Назовем его MyPanel.

В графе Superclass нажимаем на кнопку «Browse...». В строке Choose a type пишем JPanel и выбираем JPanel - javax.swing.

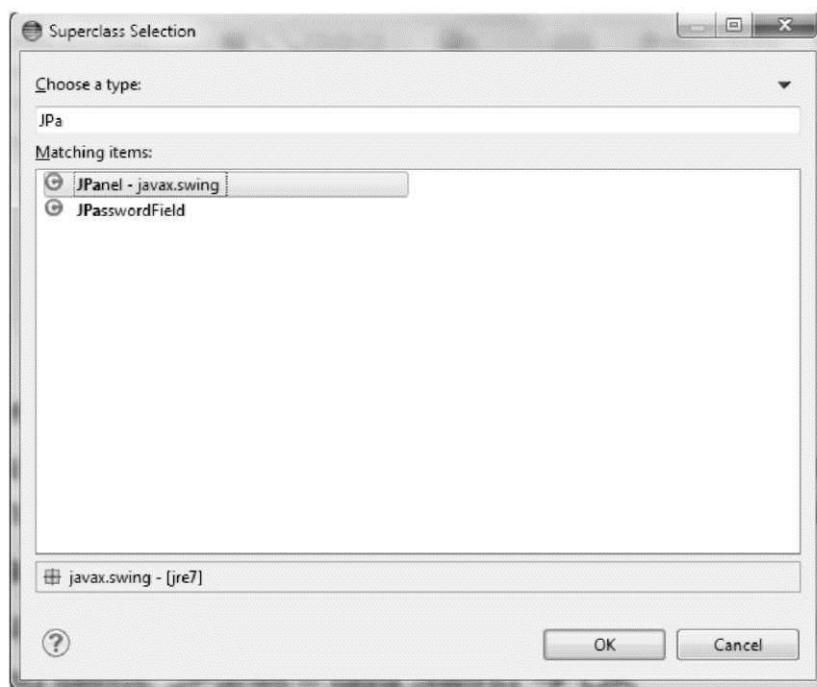


Рис. 2.17. Выбор JPanel - javax.swing

Нажимаем ОК -> Finish. Галочку возле «public static void main» НЕ СТАВИМ!

В результате создается класс MyPanel:

```
import javax.swing.JPanel;

// extends - класс MyPanel расширяет возможности JPanel
// т.е. класс MyPanel имеет и умеет все, что имеет и умеет JPanel
public class MyPanel extends JPanel { }
```

Добавляем в класс MyPanel метод public void paint(Graphics g):

```
import javax.swing.JPanel;
import java.awt.Graphics;

public class MyPanel extends JPanel {
    // @Override - означает, что метод paint переопределяет // одноименный метод в предке - т.е. в JPanel
    @Override
    // paint вызывается всегда, когда нужно перерисовать панель
    public void paint(Graphics g) {
    }
}
```

В метод paint() добавляем код, который вызывает метод paint стандартного класса JPanel - это делается для того, чтобы отрисовывалось не только то,

что мы добавили в `paint`, но и то, что наследуется от родительского класса `JPanel`:

```
import javax.swing.JPanel;
import java.awt.Graphics;

public class MyPanel extends JPanel {
    @Override
    public void paint(Graphics g) {
        // отрисовываем элементы, наследуемые от JPanel
        super.paint(g);
    }
}
```

Теперь в метод `paint()` добавляем код рисования нашего домика из ранее сделанной заготовки:

```
import javax.swing.JPanel;
import java.awt.Graphics;
import java.awt.Color;

public class MyPanel extends JPanel {
    @Override
    public void paint(Graphics g) {
        super.paint(g);
        // выберем пурпурный цвет
        g.setColor(Color.magenta);
        // рисуем корпус домика
        g.drawRect(150, 350, 250, 175);
        // рисуем внешний контур окна домика
        g.drawRect(225, 400, 100, 75);
        //вертикальная линия окна

        g.drawLine(275, 400, 275, 475); //горизонтальная линия окна
        g.drawLine(275, 425, 325, 425); // левая линия крыши
        g.drawLine(150,350,275,250); // правая линия крыши
        g.drawLine(275,250,400,350);
    }
}
```

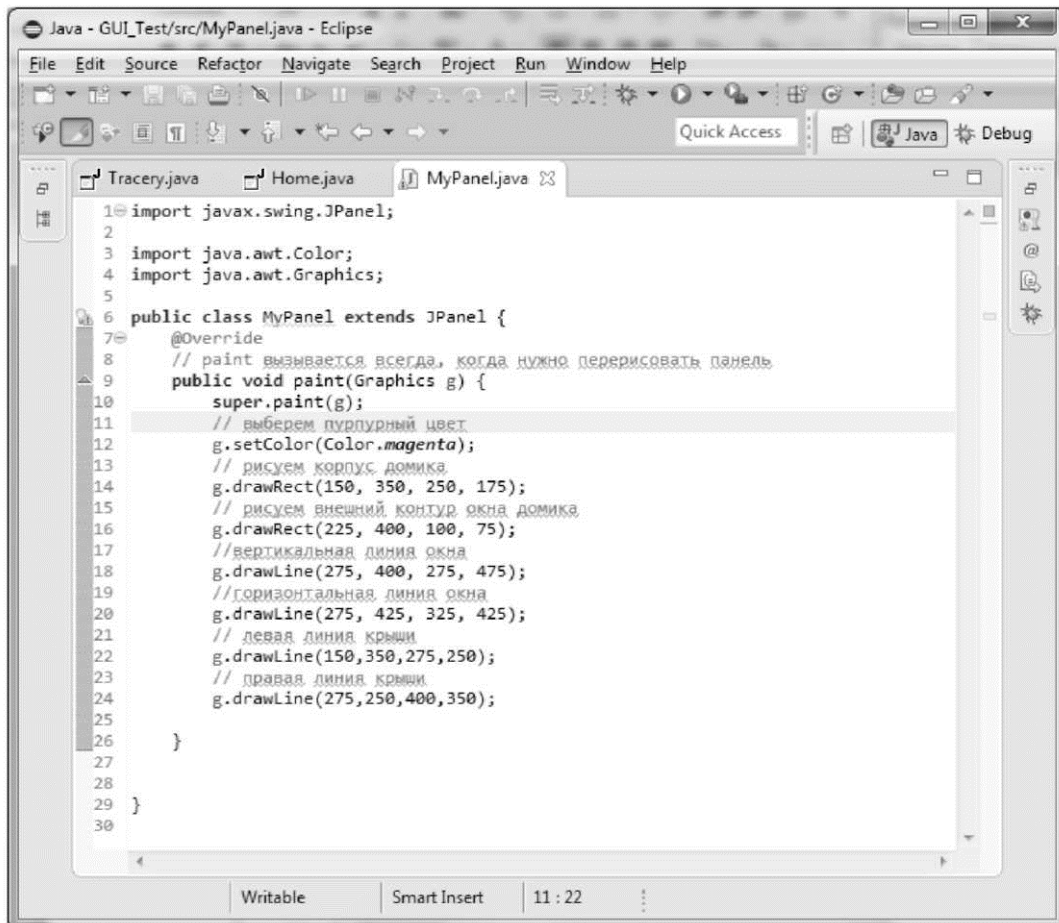


Рис. 2.18. Код рисования домика в MyPanel

Окончательный код выглядит в Eclipse так:

Создание приложения со стандартной панелью JPanel

Сначала научимся работать в окне приложения со стандартной панелью JPanel, а затем разберемся, как вместо JPanel использовать уже созданный MyPanel. Поэтому начинаем с JPanel.

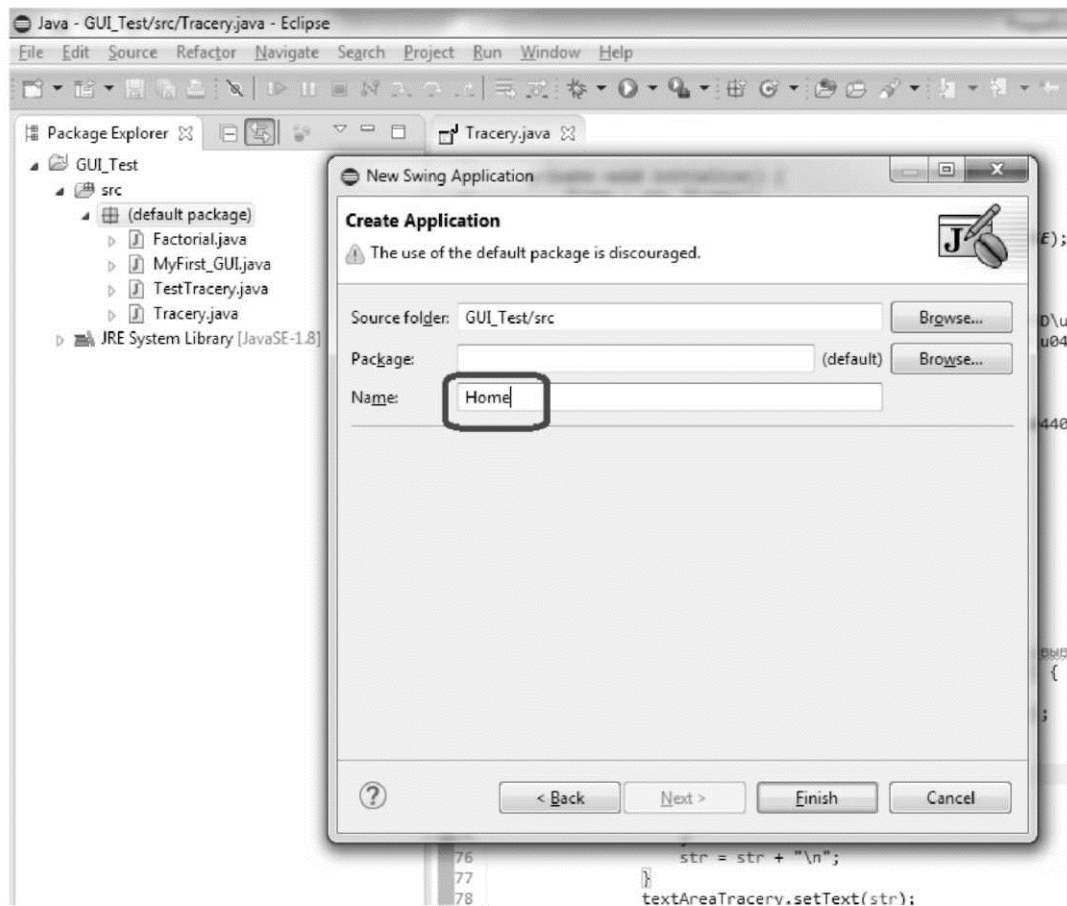


Рис. 2.19. Создание класса окна приложения Home

Создадим новое окно приложения. Его можно создать прямо в нашем старом проекте - GUI_Test. Кликаем правой кнопкой мыши по проекту и выбираем New => Other => Application Window. В открывшемся диалоге зададим имя классу окна - назовем его Home:

В окно приложения Home добавим Absolute Layout. Затем добавим контейнер JPanel, как показано на рисунке ниже:

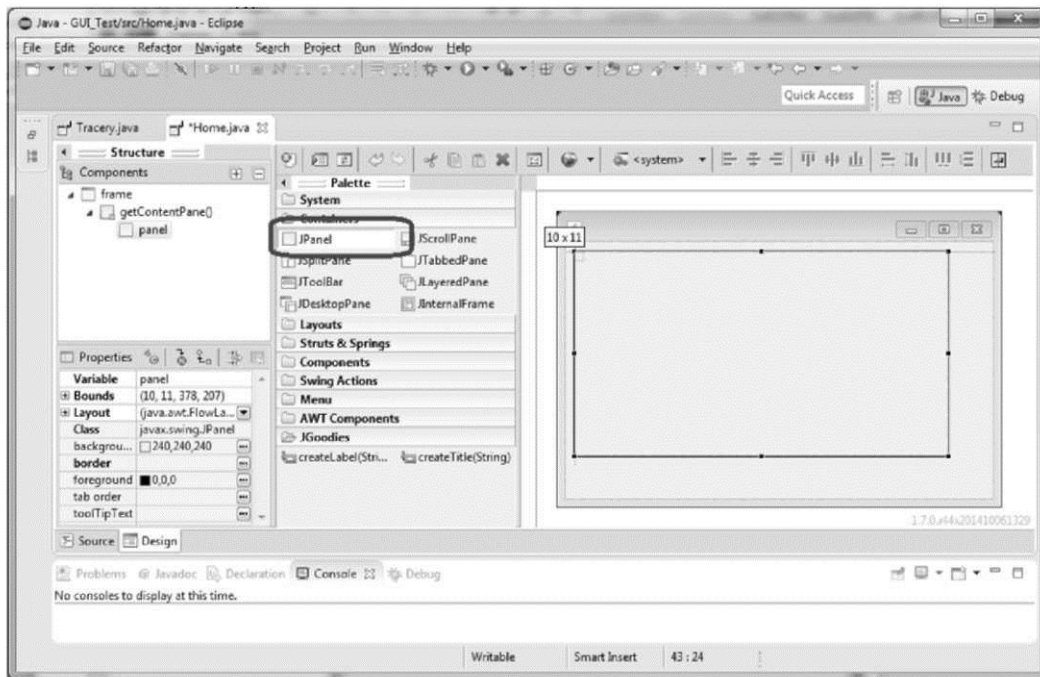


Рис. 2.20. Добавление панели JPanel

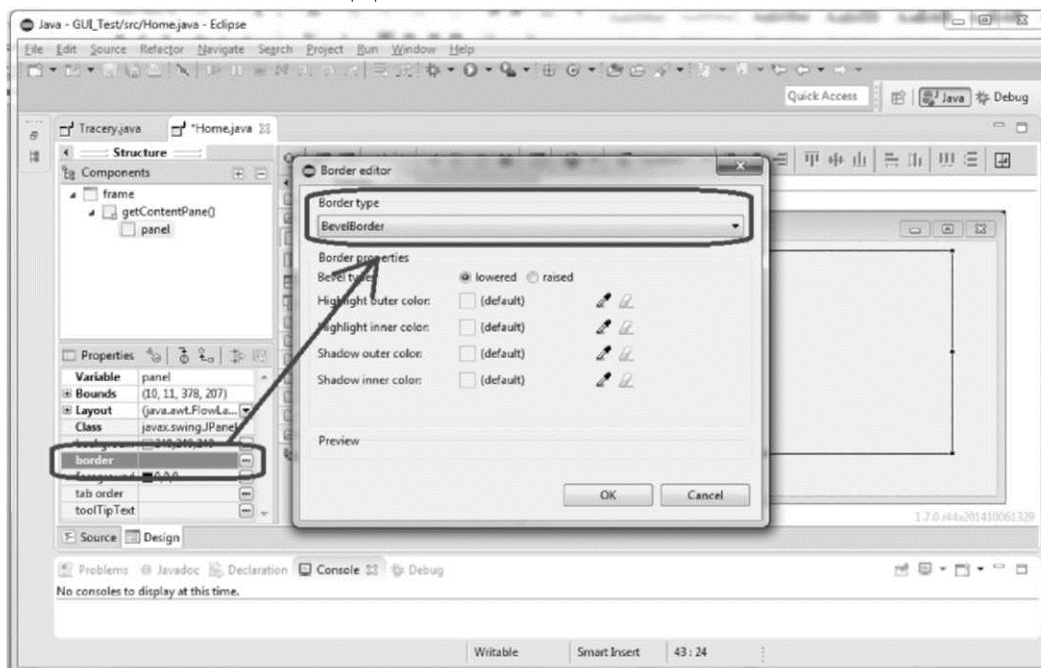


Рис. 2.21. Изменение свойств границ панели

Чтобы панель была более заметна на окне приложения, зададим ей стиль границы `BevelBorder`. Для этого изменим свойство `border`, как показано на рисунке ниже:

Нажимаем ОК. Запускаем приложение на выполнение. Увидим следующий вид работающего приложения:

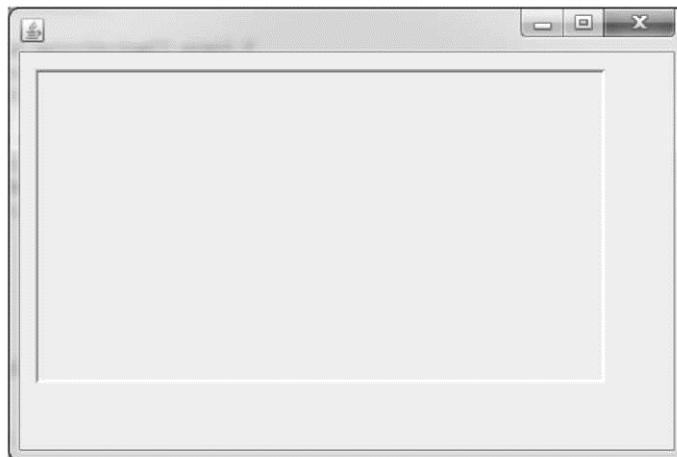


Рис. 2.22. Работающее приложение Home

Изучим код файла Home.java. Там нет ничего неожиданного - стандартный набор Home.main(), конструктор Home.Home() и метод Home.initialize():

```
import java.awt.EventQueue;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.border.BevelBorder;
public class Home {

    private JFrame frame;

    /**
     * Launch the application. */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Home window = new Home();
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application. */
    public Home() {
        initialize();
    }
}
```

```

/**
 * Initialize the contents of the frame. */
private void initialize() {
frame = new JFrame();
frame.setBounds(100, 100, 450, 300);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.getContentPane().setLayout(null);

// Создание объекта панель класса JPanel
JPanel panel = new JPanel();
// Установка стиля границы для панели
panel.setBorder(new BevelBorder(BevelBorder.LOWERED,
null, null, null, null)); pan-
el.setBounds(10, 11, 378, 207);
frame.getContentPane().add(panel);
}
}

```

Использование нашей панели MyPanel вместо стандартной JPanel

Осталось сделать пару действий и мы увидим домик в нашем окне приложения. Для начала нужно изменить размеры окна приложения Home, чтобы наш домик был в нем виден. На рисунке ниже размеры окна сделаны такими, чтобы на нем вместилась наша панель с параметрами width=432, height=534:

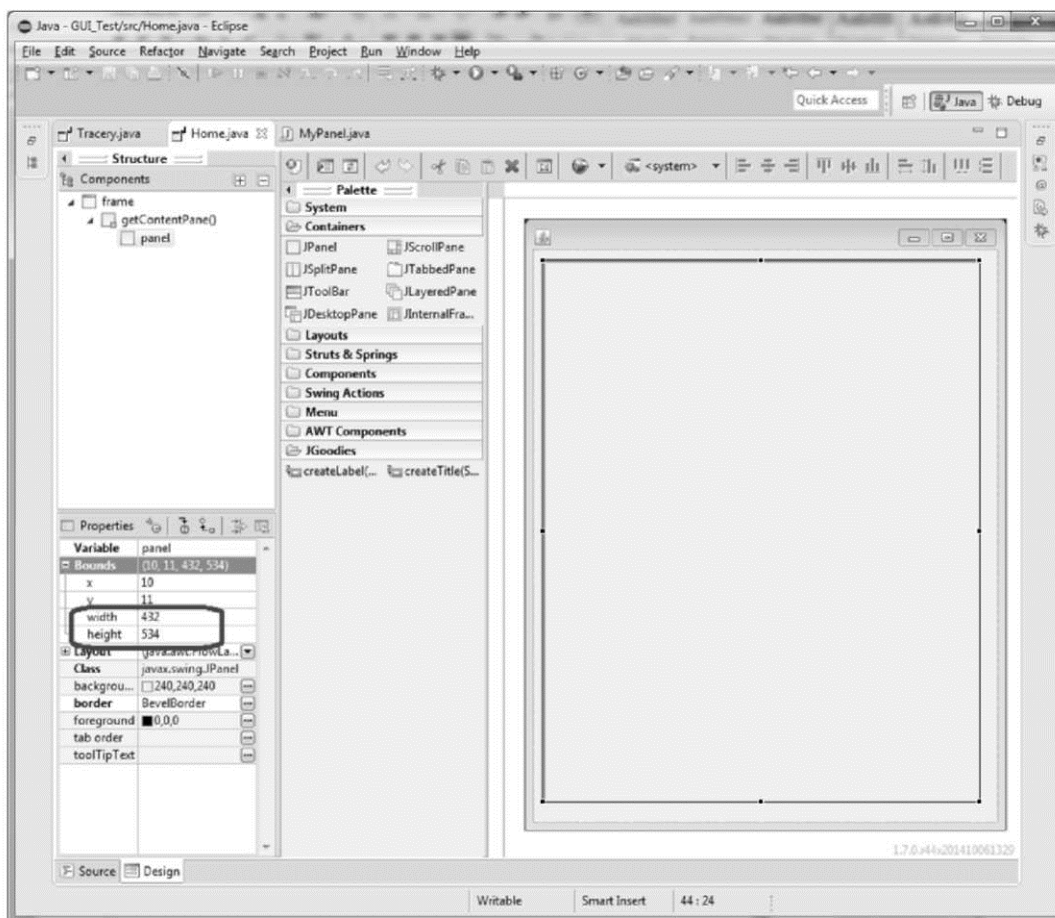


Рис. 2.23. Изменение размеров окна приложения и панели

Теперь перейдем в режим редактирования кода и найдем код метода `initialize()`:

```
* Initialize the contents of the frame. */
private void initialize() {
    frame = new JFrame();
    frame.setBounds(100, 100, 477, 604);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setLayout(null);

    JPanel panel = new JPanel();
    panel.setBorder(new BevelBorder(BevelBorder.LOWERED,
    null, null, null, null));
    panel.setBounds(10, 11, 432, 534);
    frame.getContentPane().add(panel);
}
```

Чтобы заменить стандартную панель `JPanel` на нашу собственную `MyPanel`, нам нужно изменить лишь одну выделенную строку. И вместо

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

После этого запустите приложение `Home` на выполнение, и Вы увидите такой вид работающего приложения:

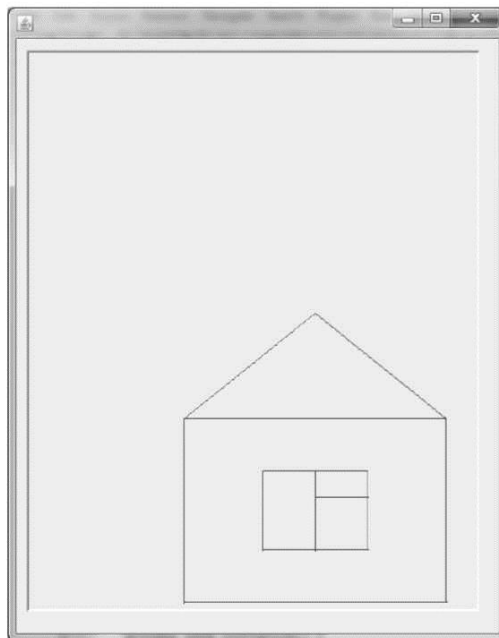


Рис. 2.24. Работающее приложение `Home` с домиком

Финальная версия кода класса `Home` выглядит так:

```
import java.awt.EventQueue;
import javax.swing.JFrame;
```

```

import javax.swing.JPanel;
import javax.swing.border.BevelBorder;

public class Home {

    private JFrame frame;

    /**
     * Launch the application.
     */
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            public void run() {
                try {
                    Home window = new Home();
                    window.frame.setVisible(true);
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }

    /**
     * Create the application.
     */
    public Home() {
        initialize();
    }

    /**
     * Initialize the contents of the frame.
     */
    private void initialize() {
        frame = new JFrame();
        frame.setBounds(100, 100, 477, 604);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().setLayout(null);

        // Здесь создается наша панель MyPanel с рисунком домика
        JPanel panel = new MyPanel();
        panel.setBorder(new BevelBorder(BevelBorder.LOWERED,
            null, null, null, null));
        panel.setBounds(10, 11, 432, 534);
        frame.getContentPane().add(panel);
    }
}

```

Генерирование изображения с использованием цикла

До этого мы в панели MyPanel выводили изображение с заранее просчитанными координатами элементов. Можно сказать, что то изображение было статичным. Сейчас мы создадим панель MyPanel2, в которой будем формировать изображение динамически - во время работы программы. Сейчас мы разберем, как нарисовать такое изображение:



Рис. 2.25. Генерируемый рисунок

Создадим класс `MyPanel2`, расширяющий `JPanel`:

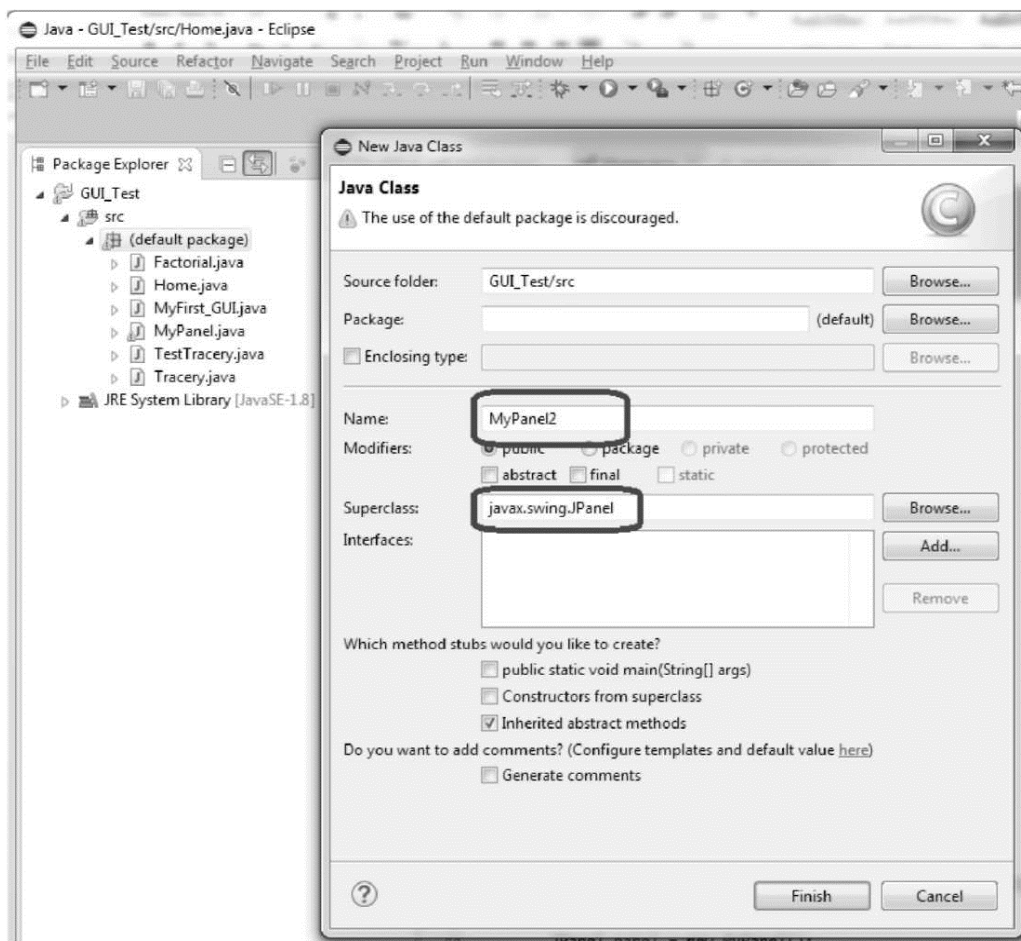


Рис. 2.26. Создание класса панели `MyPanel2`

Добавим в сгенерированный код класса `MyPanel2` метод `paint()` с кодом рисования красной линии:

```
import javax.swing.JPanel;
import java.awt.Graphics;
public class MyPanel2 extends JPanel {
@Override
    public void paint(Graphics g) {
        // отрисовываем элементы наследуемые от JPanel
        super.paint(g);
        // Цвет рисования - красный
        g.setColor(Color.red);
        // рисуем линию от точки (10,50) до точки (200,50)
        g.drawLine(10, 50, 200, 50);
    }
}
```

Теперь перейдем в код класса `Home` и изменим в методе `initialize()` строку создания панели с

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

на

```
JPanel panel = new MyPanel2();
```

После этого изменения код `initialize()` будет выглядеть так:

```
/**
 * Initialize the contents of the frame.
 */
private void initialize() {
    frame = new JFrame();
    frame.setBounds(100, 100, 477, 604);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.getContentPane().setLayout(null);
    // Здесь создается наша панель MyPanel2 с линией
    JPanel panel = new MyPanel2();
    panel.setBorder(new BevelBorder(BevelBorder.LOWERED,
        null, null, null, null)); pan-
    el.setBounds(10, 11, 432, 534);
    frame.getContentPane().add(panel);
}
```

Запустим класс `Home` на выполнение. Увидим такой рисунок на экране:

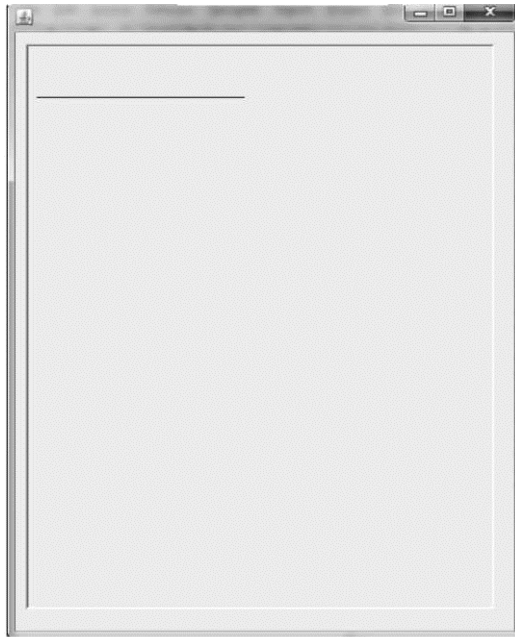


Рис. 2.27. Работа приложения Home с панелью MyPanel2

Вернемся в класс MyPanel2. Изменим метод paint() таким образом, чтобы наша линия от точки (10,50) до точки (200,50) рисовалась по-другому. А именно, координаты точки 1 (10,50) и точки 2 (200,50) зададим с использованием переменных:

```
public void paint(Graphics g) {
    super.paint(g);
    g.setColor(Color.red);
    // координаты первой точки (x1,y1)
    int x1 = 10; int y1 = 50;
        // координаты второй точки ( x2,y2)
        int x2 = 200; int y2 = 50;
        // Рисуем линию между точками (x1, y1) и (x2,y2)
        // Фактически рисуется линия между
        // точками (10, 50) и (200, 50)
        g.drawLine(x1, y1, x2, y2);
//     g.drawLine(10, 50, 200, 50);
}
```

Сохраним изменения в коде. Запустим на выполнение Home и увидим то же изображение, что и на рис.2.27.

Теперь добавим в код цикл while, внутри которого будем изменять значение переменной y2. При каждом выполнении тела цикла y2 будет увеличиваться на 10. В итоге при каждом выполнении тела цикла точка 2 будет смещаться вниз на 10. Посмотрите код, который получился после добавления while:

```
public void paint(Graphics g) {
    super.paint(g);
    g.setColor(Color.red);
    // Задаем начальные координаты точки 1
    int x1 = 10; int y1 = 50;
```

```

// Задаем начальные координаты точки 2
int x2 = 200; int y2 = 50;

// линия будет рисоваться до тех пор, пока
// вторая точки не опустится до y2 >= 150
while (y2 < 150) {
    g.drawLine(x1, y1, x2, y2);
    // значение y2 увеличивается
    // на 10 каждую итерацию
    // в итоге точка 2 опускается
    y2 = y2 + 10;
}
}

```

Запустим класс Home на выполнение сейчас. Вы увидите несколько линий, которые все начинаются в точке 1 с координатами (10,50), но заканчиваются в разных точках 2:

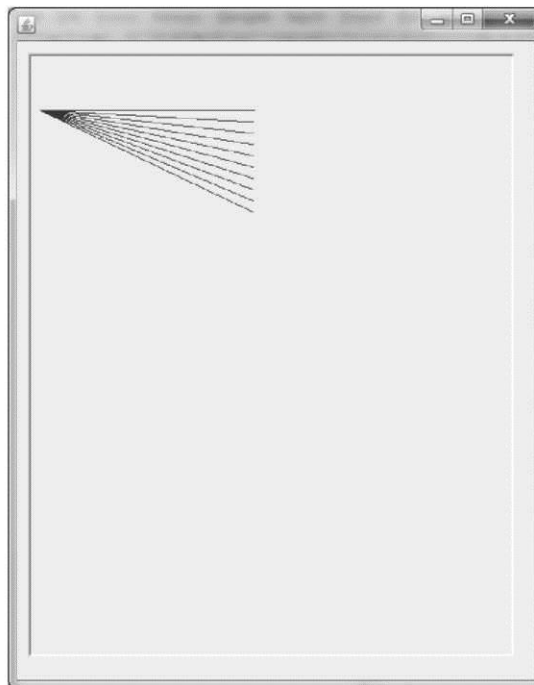


Рис. 2.28. Рисуются линии с общей точкой 1, но с разными точками 2

Добавим в цикл while также изменение координаты y1. Будем уменьшать y1 при каждом выполнении тела цикла на 5. В итоге точка 1 будет подниматься на 5 пикселей при каждом выполнении цикла. Код после изменения будет выглядеть так:

```

public void paint(Graphics g) {
    super.paint(g);
    g.setColor(Color.red);
    // Задаем начальные координаты точки 1
    int x1 = 10; int y1 = 50;
    // Задаем начальные координаты точки 2
    int x2 = 200;
    int y2 = 50;

```

```

// линия будет рисоваться до тех пор, пока
// вторая точки не опустится до y2 >= 150
while (y2 < 150) {
    g.drawLine(x1, y1, x2, y2);
    // значение y2 увеличивается
    // на 10 каждую итерацию
    // в итоге точка 2 опускается
    y2 = y2 + 10;

    // значение y1 уменьшается
    // на 5 каждую итерацию
    // в итоге точка 1 поднимается
    y1 = y1 - 5;
}
}

```

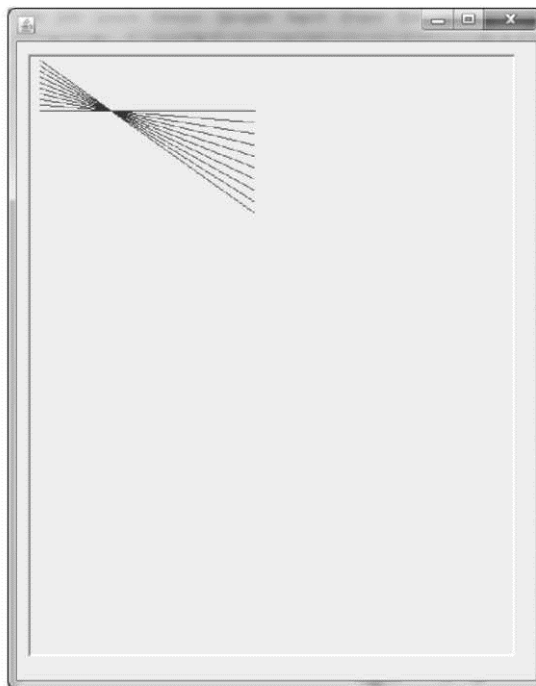


Рис. 2.29. Рисуется узор за счет перемещения точки 1 вверх и одновременное перемещение точки 2 вниз

Запустим класс `Nome` на выполнение сейчас. Вы увидите несколько линий, которые все начинаются в разных точках 1, и заканчиваются в разных точках 2:

Полный код программы будет выглядеть так:

```

import javax.swing.JPanel;
import javax.swing.JPanel;
import java.awt.Color;
import java.awt.Graphics;

public class MyPanel2 extends JPanel {
    @Override

```

```

public void paint(Graphics g) {
    super.paint(g);
    g.setColor(Color.red);

    int x1 = 10; int y1 = 50;
    int x2 = 200; int y2 = 50;
    while (y2 < 150) {
        g.drawLine(x1, y1, x2, y2);
        y2 = y2 + 10;
        y1 = y1 - 5;
    }
}
}

```

Создание сложного изображения из повторяющегося простого

Создадим изображение из множества логотипов Мерседеса:

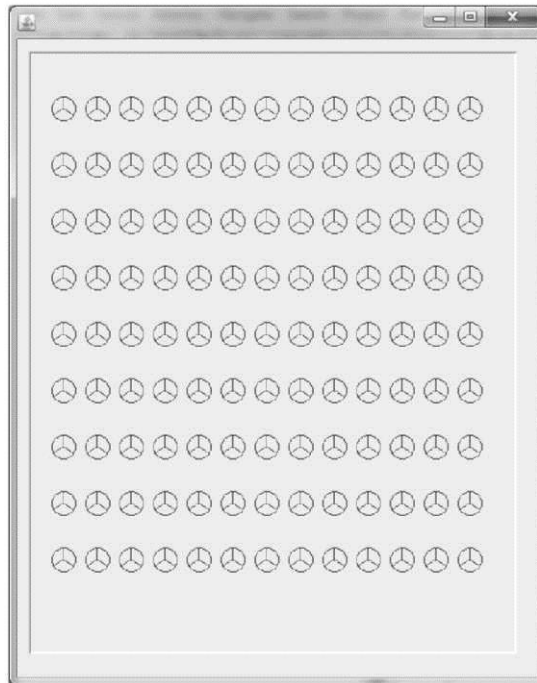


Рис. 2.30. Множество логотипов Мерседес

Для этого сначала создадим класс `MyPanel3`, расширяющий `JPanel`. Затем добавим в сгенерированный код класса `MyPanel3` метод `paint()` с кодом рисования одного логотипа Мерседеса:

```

import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class MyPanel3 extends JPanel {

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(Color.red);
    }
}

```

```

// координаты центра логотипа ( x = 100, y = 100)
int x = 100; int y = 100;
// окружность вокруг центра логотипа
g.drawOval(x - 10, y - 10, 21, 21);

// рисуем три линии внутри окружности
// линия, идущая от верхней части окружности до центра
g.drawLine(x, y, x, y - 10);
// линия, идущая от левой части окружности до центра
g.drawLine(x, y, x - 9, y + 6);
// линия, идущая от правой части окружности до центра
g.drawLine(x, y, x + 9, y + 6);
}
}

```

Обратите внимание на то, что элементы логотипа рисуются, используя относительные координаты: $y-10$, $x-9$, $y+6$ и т.д. Это позволяет нам легко нарисовать изображение логотипа в любом месте экрана. Для этого достаточно изменить координаты x и y .

В классе `Home` изменим в методе `initialize()` строку создания панели с

```
JPanel panel = new MyPanel12();
```

на

```
JPanel panel = new MyPanel13();
```

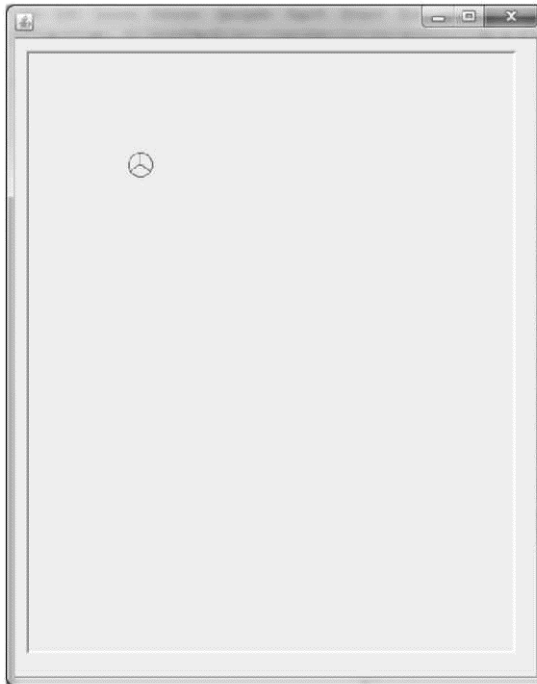


Рис. 2.31. Рисуются один логотип

После чего запустим `Home` на выполнение. Мы увидим такой вид запущившегося приложения:

Чтобы вывести изображение в виде строки из логотипов, нам понадобится цикл. Будем использовать `while`. В начале зададим начальные координаты

центра логотипа и будем продвигать центр вправо, рисуя логотипы один за другим:

```
import java.awt.Color; import
java.awt.Graphics;
import javax.swing.JPanel;

public class MyPanel3 extends JPanel {

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(Color.red);
        // начальные координаты центра логотипа ( x = 100, y =30)
        int y = 100; int x = 30;

        // будем рисовать, пока x не достигнет 400
        while (x < 400) {
            g.drawOval(x - 10, y - 10, 21, 21);
            g.drawLine(x, y, x, y - 10);
            g.drawLine(x, y, x - 9, y + 6);
            g.drawLine(x, y, x + 9, y + 6);
            // увеличиваем значение x на 30 каждую итерацию
            // т.е. следующий логотип будет
            // нарисован правее на 30 пикселей
            x = x + 30;
        }
    }
}
```

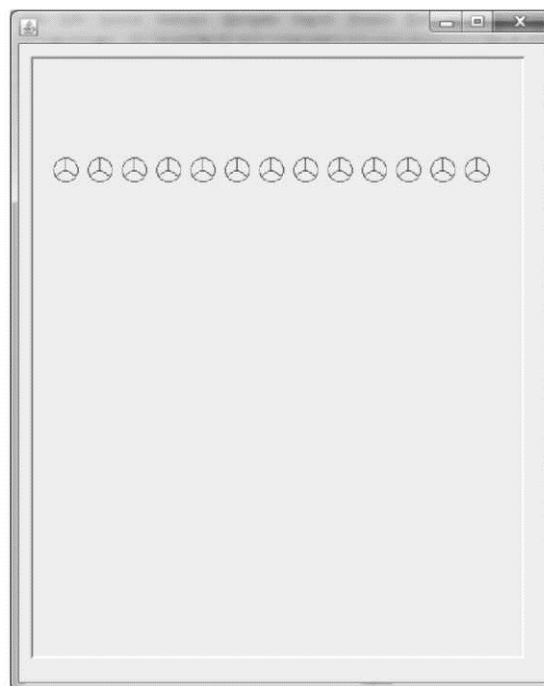


Рис. 2.32. Рисование логотипов вдоль оси x

Чтобы вывести несколько строк логотипов нам понадобится второй цикл. Также будем использовать `while`. Ранее созданный цикл теперь окажется внутри

вновь созданного. В новом цикле мы будем менять переменную *y*, вначале задав ей минимальное значение, и с каждым шагом увеличивая ее, тем самым продвигая логотипы ниже и ниже:

```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JPanel;

public class MyPanel3 extends JPanel {

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(Color.red);
        // начальные координаты центра логотипа (x = 100, y =30)

        int y = 50;
        // будем рисовать строки логотипов пока

        // не опустимся до 450
        while (y <= 450) {

            // x нужно инициализировать при КАЖДОМ
            // новом проходе тела цикла по y
            int x = 30;

            // будем рисовать, пока x не достигнет 400
            while (x < 400) {
                g.drawOval(x - 10, y - 10, 21, 21);
                g.drawLine(x, y, x, y - 10);
                g.drawLine(x, y, x - 9, y + 6);
                g.drawLine(x, y, x + 9, y + 6);
                // увеличиваем значение x на 30 каждую итерацию
                // т.е. следующий логотип будет
                // нарисован правее на 30 пикселей
                x = x + 30;
            }
            // увеличиваем значение y на 50 каждую итерацию
            y = y + 50;
        }
    }
}
```

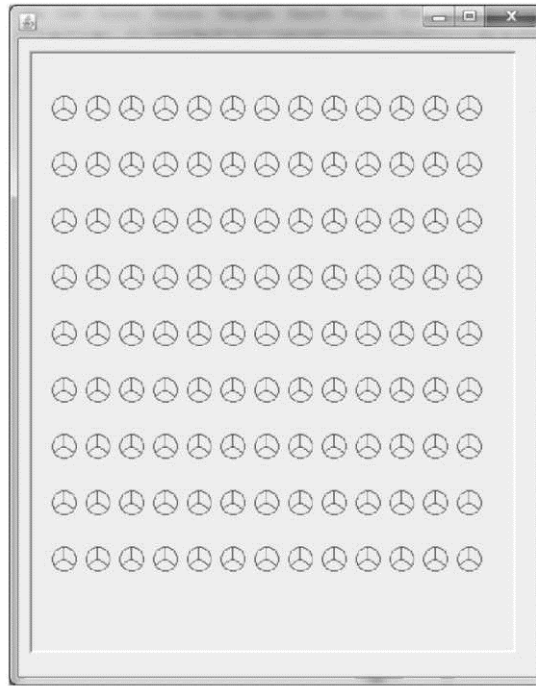


Рис. 2.33. Рисование логотипов вдоль оси x и оси y

Это именно тот вид, который мы хотели получить.

Поздравляем Вас! Вы добрались до самого интересного! Все примеры разобраны, вся информация получена! Теперь Вам стоит приступить к выполнению задач, которые приведены далее. Удачи Вам в освоении программирования GUI приложений на Java!

ЗАДАНИЯ ДЛЯ ПРАКТИЧЕСКИХ РАБОТ

Практическая работа №1 - создание узора из чисел

Создать GUI приложение, формирующее узор из чисел. Варианты взять из задачи С. На входе у приложения число $size$ ($0 \leq size \leq 9$) - задается в элементе JSpinner. На выходе узор в поле JTextArea. При изменении значения в поле ввода, узор должен перерисовываться автоматически. Внешний вид программы должен быть таким:

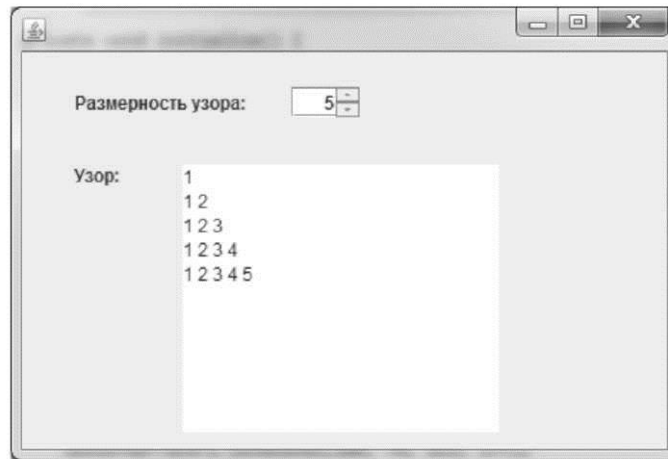


Рис. 3.1. Пример реализации задачи D вариант 0

Таблица 3.1

Вариант	Рисунок	Вариант	Рисунок
0	1 1 2 1 2 3 1 2 3 4 1 2 3 4 5	1	1 2 3 4 5 1 2 3 4 1 2 3 1 2 1
2	5 4 5 3 4 5 2 3 4 5 1 2 3 4 5	3	5 4 3 2 1 5 4 3 2 5 4 3 5 4 5
4	5 5 4 5 4 3 5 4 3 2 5 4 3 2 1	5	1 2 1 3 2 1 4 3 2 1 5 4 3 2 1

Вариант	Рисунок	Вариант	Рисунок
6	1 2 3 4 5 2 3 4 5 3 4 5 4 5 5	7	5 5 5 5 5 5 5 5 5 5 5 5 5 5 5
8	5 5 5 5 5 5 5 5 5 5 5 5 5 5 5	9	5 4 3 2 1 4 3 2 1 3 2 1 2 1 1

Практическая работа №2 - создание статического изображения

Создать окно приложения с собственной панелью. На панели нарисовать статичное тематическое изображение - дом, сад, поезд, ракета и т. п. Ниже приведен пример рисования дома:

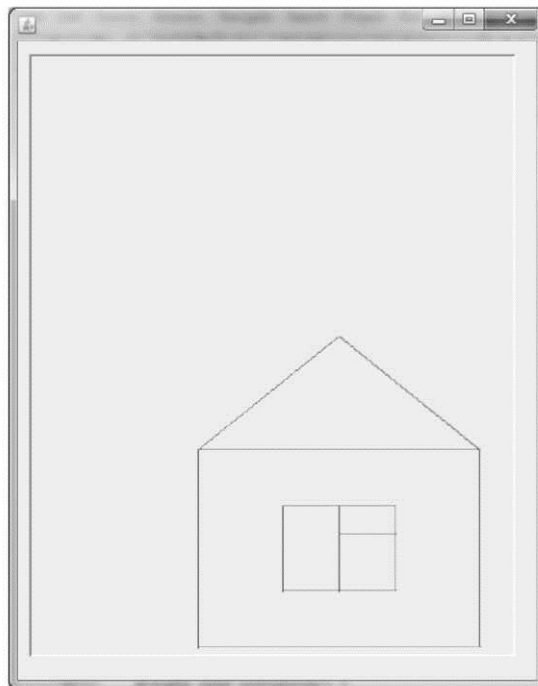


Рис. 3.2. Окно приложения с панелью с отрисовкой домика

Необходимо создать изображение с использованием разных цветов (минимум 3 разных цвета). Изображение должно быть достаточно сложным - в нем должно быть около 25-40 элементов. (На рис. 3.2 всего 12 элементов). Варианты:

0. Домик
1. Дом с садом.
2. Автомобиль на трассе
3. В кабинете программиста
4. Цветочная поляна

5. На поле танки грохотали...
6. Новогодняя елка
7. Стол
8. Стул
9. Шкаф
10. Музыкальный инструмент
11. Молоток и наковальня
12. Принтер
13. Самолет
14. Паровоз
15. Флаги стран
16. Завод
17. Дом многоквартирный
18. Мост
19. Корабль
20. Свободная тема - можно предложить свой собственный рисунок

Задание считается выполненным при наличии бумажного листка с оцифровкой точек картинка.

Практическая работа №3 - создание динамического изображения

Создать окно приложения с собственной панелью. На панели нарисовать динамическое изображение, созданное за счет многократного рисования линии между двумя точками, которые одновременно перемещаются. Ниже приведен пример рисования такого изображения:

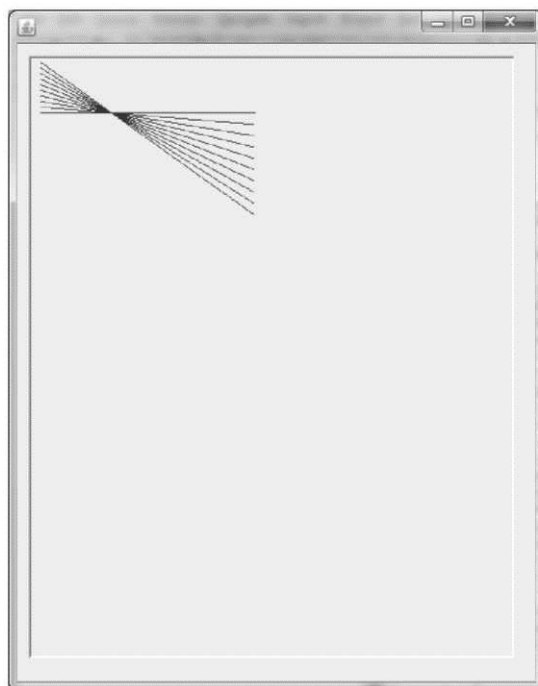
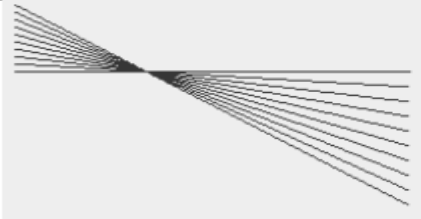
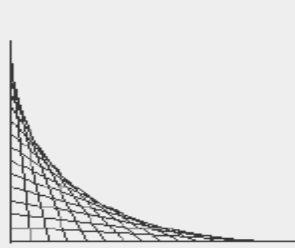
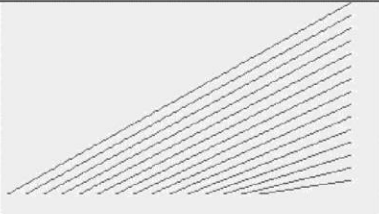
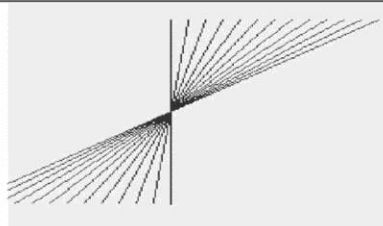
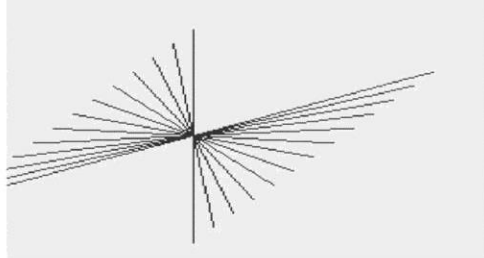
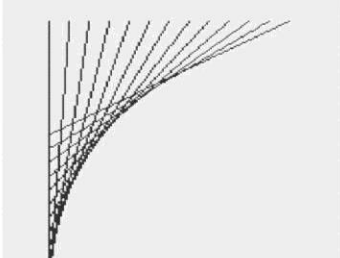
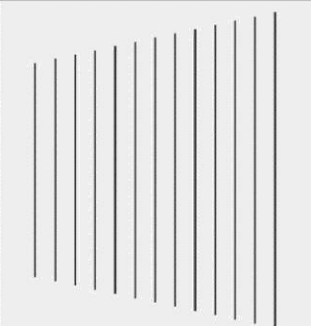
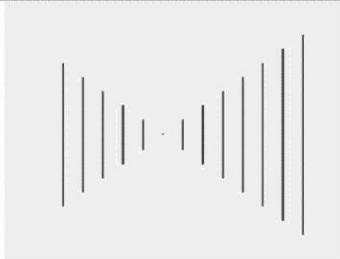
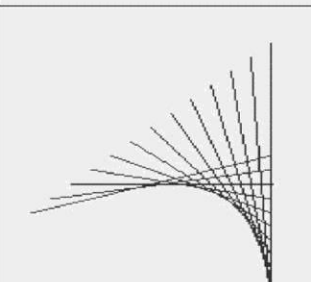
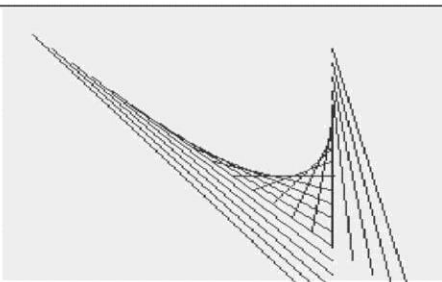


Рис. 3.3. Рисование динамического изображения

Таблица 3.2

Варианты

Вариант	Рисунок	Вариант	Рисунок
0		1	

2		3	
4		5	
6		7	
8		9	
10	Свой собственный рисунок		

Практическая работа №4 - Создание сложного изображения из повторяющегося простого

Нужно нарисовать N рядов изображений логотипов с M логотипами в каждом ряду. N и M задаются в программе. Для рисования нужно использовать

относительные координаты и вложенные циклы. Ниже представлен пример рисования логотипа Мерседеса при $N=6$, $M=16$:

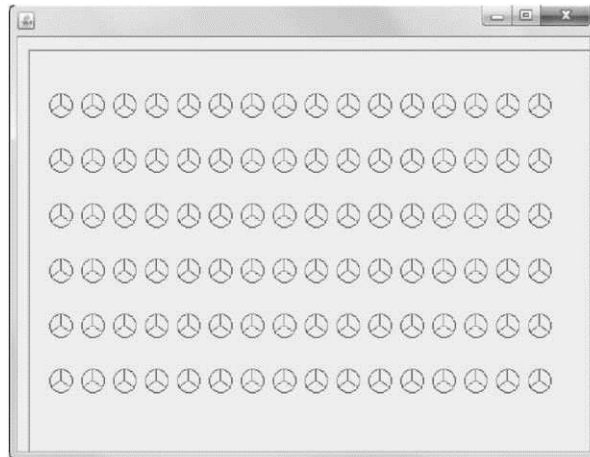


Рис. 3.4. Рисование логотипов

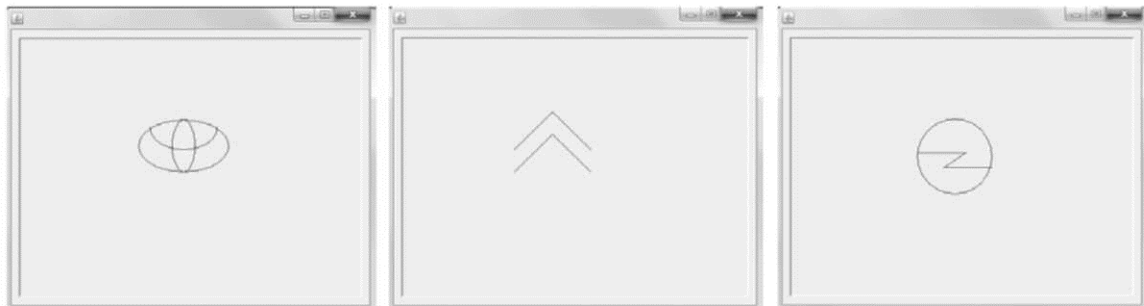


Рис. 3.5. Примеры рисование логотипов

Рисунок логотипа должен быть максимально простым. Ниже приведены примеры рисования логотипов Тойоты, Ситроена и Опеля:

Варианты (0 - 19)

Номер варианта	Логотип по варианту	Номер варианта	Логотип по варианту
0	 Mercedes-Benz	1	
2	 TOYOTA	3	 mazda
4	 Audi	5	 HONDA
6	 LEXUS	7	 HYUNDAI
8	 CITROËN	9	 KIA MOTORS
10		11	
12		13	 CHERY
14		15	 RENAULT
16	 INFINITI	17	 ACURA
18	 SUZUKI	19	

ЗАКЛЮЧЕНИЕ

Огромный дефицит ИТ специалистов в сфере разработки программного обеспечения делает актуальным вопрос наискорейшего освоения новых языков программирования и современных технологий разработки ПО. Разработка программ для конечного пользователя как GUI приложений является сегодня стандартом. Поэтому актуально для изучающего новый язык очень быстро научиться делать на нем GUI приложения. Данные методические указания предназначены именно для этого. Здесь в одно целое собран материал, касающийся знакомства с разработкой GUI приложений на Java.

В методических указаниях представлена минимально необходимая информация по установке и использованию WindowBuilder, по базовым компонентам Swing (панели, кнопки, метки, поля редактирования и др.), созданию генерируемых изображений средствами Graphics.

Приведены задания на 4 работы по созданию GUI приложений с вводом и выводом текста, созданию изображений различной степени сложности средствами Graphics.

Методические указания можно применять как в процессе выполнения лабораторных работ в аудитории, так и для самостоятельного изучения основ разработки GUI приложений на Java.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1 **Блинов, И.Н.** Java 2. Практическое руководство / Блинов, И.Н., Романчик, В.С. - Мн.: УниверсалПресс, 2005. – 400с.

2 **Васильев, А. Н.** Java. Объектно-ориентированное программирование для магистров и бакалавров: базовый курс по объектно-ориентир. программированию : учеб. пособие / А. Н. Васильев. – М. ; СПб. ; Нижний Новгород : Питер, 2013. – 396 с.

3 **Вязовик, Н.А.** Программирование на Java. [электронный ресурс] <http://www.intuit.ru/department/pl/javapl>.

4 Введение в программирование Java [Электронный ресурс]. (<https://www.ibm.com/developerworks/ru/java/newto/>).

5 **Дж. Стивен Перри.** Введение в Java-программирование: Часть 2. Конструкции реальных приложений [Электронный ресурс] (<https://www.ibm.com/developerworks/ru/edu/j-introtojava2/index.html>).

6 **Зыков, С.В.** Программирование. Объектно-ориентированный подход: учебник и практикум для академического бакалавриата / С.В. Зыков. – М. Научная школа: Национальный исследовательский университет «Высшая школа экономики», 2017. – 155 с. [электронный ресурс]. <https://biblio-online.ru/book/E006A65E-B936-4856-B49E-1BA48CF1A52F>.

7 **Николаев, Е.И.** Объектно-ориентированное программирование: учебное пособие. – Ставрополь: Изд-во СКФУ, 2015. – 255 С. [электронный ресурс] <http://www.knigafund.ru/books/200468/>

8 **Тузовский, А.Ф.** Объектно-ориентированное программирование: учебное пособие для прикладного бакалавриата / Тузовский А.Ф. – Томск. Научная школа: Национальный исследовательский Томский политехнический , 2017. – 206 с. [электронный ресурс] <https://biblio-online.ru/book/BDEEFB2D-532D-4306-829E-5869F6BDA5F9>.

9 **Хорстманн К.С., Корнелл Г.** Библиотека профессионала. JAVA 2. Том 1. Основы. 8-е издание. Пер. с англ. – М.: ООО Издательский дом “Вильямс”, 2008. – 816 с.

10 **Хорстманн, К.С., Корнелл, Г.** Библиотека профессионала. Java 2. Тонкости программирования. Том 2. – М.: «Вильямс», 2002. - 1120 с.

11 **Эккель, Б.** Философия JAVA. 4-е издание. – СПб.: Питер, 2009. – 638 с.

12 **Шилдт, Г.** Java. Полное руководство, 8-е изд.: Пер.с англ. – М.: ООО «И.Д. Вильямс», 2012 . – 1104 с.

Интернет-ресурсы

1. Руководство по Eclipse: <http://sotnyk.com/2011/10/09/rukovodstvo-po-eclipse-ide/> (Дата обращения: 27.08.2015).
2. Eclipse - учебное пособие: <http://window.edu.ru/resource/397/58397> (Дата обращения: 27.08.2015).
3. Официальный сайт WindowBuilder -<https://eclipse.org/windowbuilder/> (Дата обращения: 27.08.2015).

Учебное издание

Игнатьева Олеся Владимировна

ОБЪЕКТНО ОРИЕНТИРОВАННОЕ ПРОГРАММИРОВАНИЕ

Печатается в авторской редакции

Технический редактор А.В. Артамонов

Подписано в печать 31.08.17. Формат 60×84/16.

Бумага газетная. Ризография. Усл. печ. л. 4,88.

Тираж экз. Изд. № 9014. Заказ .

Редакционно-издательский центр ФГБОУ ВО РГУПС.

Адрес университета: 344038, г. Ростов н/Д, пл. Ростовского Стрелкового Полка
Народного Ополчения, д. 2.